Context-Aware Fuzzy Logic Dispatcher for Real-Time IoT-Based Disaster Monitoring

MSc Research Project
MSCCLOUD1-A

Harsh Gavhane
Student ID: X23277653

School of Computing
National College of Ireland

Supervisor: Aqeel Kazmi

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Harsh Gavhane
Student ID:	X23277653
Programme:	MSCCLOUD1-A
Year:	2025
Module:	MSc Research Project
Supervisor:	Aqeel Kazmi
Submission Due Date:	15/09/2025
Project Title:	Context-Aware Fuzzy Logic Dispatcher for Real-Time IoT-Based Disaster Monitoring
Word Count:	7371
Page Count:	20

Signature:	Harsh
Date:	15th September 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.	☐

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

Context-Aware Fuzzy Logic Dispatcher for Real-Time IoT-Based Disaster Monitoring

Harsh Gavhane
X23277653

Abstract

Disaster-prone environments demand intelligent monitoring systems capable of making real-time decision to minimize harm and loss of life. Conventional threshold-based systems result a lack of flexibility and contextual awareness, which typically generating high rates of false alarms and delay in response. This paper proposes a contextual event prioritization solution using fuzzy logic and MQTT IoT communication along with a scalable serverless AWS disaster response architecture. It maps sensor events, for instance, gas, temperature and humidity to low, medium or high priority depending on sensor values as well as zone severity of the geographic zone. A fuzzy inference system is implemented using fuzzy algorithm simulating human-like decision-making under uncertainty, where simulated IoT data is published over MQTT to a context-aware dispatcher that decides based on fuzzy rules and context weights. High-priority events trigger AWS Lambda for real-time response, medium-priority events are noted for fog-level computation, and low-priority data are persisted in DynamoDB to analyze later. Experimental validation with simulated data demonstrated event classification accuracy, with a reduction in false alarms compared to static threshold models, with less than one second real-time responsiveness. Such findings indicate that fuzzy-logic-based context-aware systems significantly improve prioritization accuracy and performance, with a cost-effective and scalable solution to disaster monitoring, and future development will focus on predictive analytics and integration of machine learning for further optimisation.

1 Introduction

The growth in the Internet of Things (IoT) has transformed environmental monitoring and disaster response with the promise of billions of smart devices to collect and transmit live data Perera, Zaslavsky, Christen and Georgakopoulos (2014). This is particularly critical in disaster-prone areas where detection early enough and quick response can prevent the loss of a very large number of lives and reduce damage to property. As disaster caused by climate changes such as floods, wildfires and poisonous chemical leaks become increasingly frequent due to global warming and urbanization, the demand for intelligent, adaptive surveillance systems has never been greater.

Most of the current disaster monitoring systems rely on rigid threshold-based decision mechanisms, which classify events solely based on pre-arranged numerical thresholds. Such systems, as much as they are computationally inexpensive, are context-insensitive and brittle and, as a result, suffer from very high false-positive and false-negative rates Gogo et al. (2018). For example, the level of a gas exceeding a threshold could represent life-threatening danger in an industrial area but be acceptable in an open rural area. Similarly, the same temperature fluctuation would create significantly different levels of risks depending on place and environmental factors. Such limitations decrease system reliability, increase response time, and flood emergency authorities with redundant alarms.

To address these challenges, this research proposes the context-aware based fuzzy-logic algorithm which alarm prioritization with MQTT-based communication and serverless scalable AWS-based infrastructure. Fuzzy logic, as opposed to traditional binary models, offers human-like reasoning in uncertain situations using linguistic variables and membership functions to allow transition of decisions rather than sudden classification Nassar and Karray (2016). The system to be deployed processes sensor outputs—e.g., gas level, temperature, and humidity—along with situational parameters such as severity of zone in order to derive priority levels: *LOW*, *MEDIUM*, or *HIGH*. High-priority alarms call AWS Lambda for autonomous action, medium-priority alarms are dedicated to fog-level action, and low-priority data are securely gathered into DynamoDB for further study. MQTT, a low publish-subscribe protocol, ensures effective and secure data transmission from distributed IoT devices to the dispatcher and the serverless framework provides elasticity as well as cost-effectiveness with increased operation loads Asghar et al. (2018).

Research Question: Can an IoT and serverless cloud-integrated context-aware, fuzzy-logic-based prioritization mechanism improve disaster monitoring system real-time responsiveness and diminish false alarms?

Research Objectives:

- To review state-of-the-art approaches in IoT-based disaster monitoring, fuzzy logic and serverless architectures.
- To develop and apply a fuzzy based system of context-aware prioritization of IoT events.
- To integrate the IoT communication through MQTT with AWS lambda function and Dynamo DB as the foundation of scalable and event-driven process.
- To test the system in order to maximize the prioritization accuracy and minimize the latency based on the synthetic sensor data.

Assumptions and Limitations: This paper based on the assumption of quality MQTT message reception and quality simulation of fake sensor data. Evaluation is restricted to a lab test environment and not actual installations, although the developed architecture allows scalability.

Contribution: The main contributions of this research are: (i) environmental context-aware fuzzy model to make decisions, (ii) event-driven AWS Lambda-based serverless backend for economic scalability and (iii) MQTT-based IoT based real-time responsiveness to the data. All of these together facilitate better priority accuracy, lower false alarms and sub-second responsiveness across simulated disaster scenarios.

Structure of the Report: The remainder of this report is organized as follows: Section 2 presents related research on IoT-based disaster monitoring, fuzzy logic systems, and serverless computing. Section 3 describes research methodology. Section 4 presents design specifications, architecture, and data flow. Section 5 presents implementation, while Section 6 presents evaluation results and discussion. Section 7 concludes the report and presents future research work. Section 8 presents references consulted in this research.

2 Related Work

The literature on IoT-based disaster monitoring manifests significant advancements on five underlying subjects: context-aware computing, adaptive decision-making through fuzzy logic, convergence of serverless and edge/fog computing, IoT communication protocols, and synthetic data generation for testing The paper critically evaluates the themes by providing the approaches, advantages, and limitations of current research to determine the gap addressed by this research.

2.1 Context-Aware Computing in IoT

Perera, Zaslavsky, Christen and Georgakopoulos (2014) provided one of the largest context-aware computing surveys, rating 50 projects in accordance with an IoT application taxonomy. Their observation placed the life cycle of context management within context of how it enables enhancement in adaptability in dynamic environments. Though this work provides a basic insight, it does not provide application strategies applicable for disaster-prone scenarios as well as in real-time processing. Further, Perera, Liu, Jayawardena and Chen (2014) also explored IoT from an industrial marketplace point of view, offering context-aware business system design trends and opportunities but, it was more market-oriented rather than the adaptive algorithmic integration for safety-critical systems.

A recent work of Cicconetti et al. (2021) suggested a decentralized serverless edge computing framework that ensures adaptability and responsiveness for IoT systems. The proposed framework is efficient in terms of mobility and network unavailability performance but not semantic context-awareness. Gogo et al. (2018) suggested context-aware systems using fuzzy logic for adaptive e-learning, demonstrating the applicability of the approach in different applications. While effective for learning applications, its usage for time-critical applications such as disaster tracking has not been explored.

2.2 Fuzzy Logic for Decision-Making

Traditional threshold-based methods are common in IoT implementations but are plagued by rigid decision-making to result in false alarms with dynamic situations Gogo et al. (2018). Nassar and Karray Nassar and Karray (2016) offered fuzzy logic algorithm in VANET for issuing crash warnings using linguistic variables for estimating severity. Their approach produced an improvement of accuracy with a large magnitude over binary estimators. This research was not concerning environmental surveillance but was directed to automobile fronts. Rashid and Wani (2021) have studied fog analytics of IoT data data but there was no fuzzy reasoning inside of it so the flexibility was low.

One of the recent studies combined fuzzy logic with IoT for smart agriculture and permitted precise control of irrigation in changing environmental conditions *(PDF) IoT-Based Smart Monitoring and Controlling System for Shallot Planting Medium Conditions Using a Combination of Context-Aware and Fuzzy Logic Algorithms* (n.d.).The integrated model is greatly contextually flexible but domain-specific, and scalability under huge disaster situations has yet to be examined. Collectively, the given studies are reflective of the adoption of fuzzy logic in terms of making sensitive decisions but not in terms of the convergence of fuzzy logic and real time server less architecture in disaster monitoring being used in the given research.

2.3 Serverless and Edge/Fog Integration

Serverless computing emerged as a scalable architecture for IoT systems by neglecting the cost of reserved infrastructure. Asghar et al. (2018) demonstrated AWS Lambda as a feasible choice in Disaster Management Information Systems by avoiding cost in off-peak periods and elasticity in peak periods. However, their experiment revealed cold start latency to be a deterrent to real-time processing. Xu et al. (2019) addressed this problem by dropping this latency due to the application of Adaptive Warm-Up Strategies at a cost of overhead and complexity.

Cheng et al. (2019) suggested the Fog Function framework, which applies FaaS to the fog layer for enhanced response and reduced bandwidth usage. Similarly, Ghosh et al. (2018) set up adaptive heuristics for energy-aware scheduling among edge and cloud resources. Each of these papers emphasizes strengths of distributed architectures but is less concerned with computational efficiency than it is with prioritization within context.Pinto et al. (2018) proposed dynamic serverless function assignment for IoT settings with optimized execution levels but without consideration of adaptive reasoning in uncertainty, thus creating a principal gap for safety-related applications.

2.4 IoT Communication and Data Persistence

Cheng et al. (2019) suggest a Fog Function on FaaS at the fog level that leads to higher responsiveness and bandwidth efficiency. Real-time monitoring of IoT relies heavily on effective communication protocols as well as storage facilities. As a publish-subscribe protocol, MQTT is very lightweight and is utilized extensively. Ai et al. (2021) demonstrated the effectiveness of MQTT for AGV scheduling, and Aditya et al. (2022) utilized it in smart greenhouses with great success rates of delivery and energy efficiency.

Kalid et al. (2017) compared NoSQL alternatives such as Amazon DynamoDB in comparing its suitability in high-throughput IoT applications. The more it scales, the less support it has with adaptive priority mechanisms in the existing works.The two works emphasize strengths of distributed designs but are more concerned with priority within context than computational efficacy.

2.5 Synthetic Data for Experimental Validation

It is hard to come by real-world disaster data sets to experiment with because of cost, safety and uncertainty. Kang et al. (2024) proposed synthetic data creation using Generative Adversarial Networks (GANs) in healthcare and were successful in obtaining distribution similarity with real data sets. Ghosh et al. (2018) also emphasized simulation for the assessment of dynamic scheduling strategies over edge and cloud infrastructures. Even though these solutions enable controlled experiments, hardly any experiments use synthetic data to validate fuzzy logic-based priority for IoT disaster monitoring systems.

2.6 Summary and Research Gap

The reviewed literature evidently demonstrates the evolution in IoT monitoring, context awareness, fuzzy reasoning and hybrid frameworks. Significant gaps remain:

- Few are the solutions today of marrying **context-aware fuzzy logic with MQTT-driven IoT communication and serverless architectures** to fuel real-time disaster response.

- All work to optimize performance (i.e., power, latency) without considering **adaptive prioritization in uncertainty**, but it does so in risk danger situations.
- Few works include **synthetic data-driven validation** for scalable simulation of disaster scenarios.

Table 1: Main Related Work and Research Contribution

Related Work	**Strengths**	**Limitations**
Perera, Zaslavsky, Christen and Georgakopoulos (2014)	Large survey, IoT taxonomy, context life cycle.	No real-time disaster focus.
Nassar and Karray (2016)	Fuzzy logic improves decision accuracy in uncertainty.	Limited to vehicular context.
Asghar et al. (2018)	Cost-effective, elastic serverless DMIS.	Cold-start latency issues.
Xu et al. (2019)	Reduces cold-start delay.	Added complexity/overhead.
Cheng et al. (2019)	Improved response, reduced bandwidth use.	No adaptive prioritization.
Ghosh et al. (2018)	Adaptive heuristics for efficient scheduling.	Lacks context-based priority.
Pinto et al. (2018)	Optimized execution level selection.	No uncertainty-based reasoning.
Kang et al. (2024)	Realistic synthetic datasets.	Not disaster-specific fuzzy logic.
My Research	Combines fuzzy logic, MQTT, and hybrid Edge–Fog–Cloud with AWS Lambda for real-time disaster event prioritization; validated with synthetic disaster datasets.	Addresses identified gaps by enabling adaptive prioritization under uncertainty in disaster monitoring.

The present work is satisfied with these deficiencies as they propose a **fuzzy-logic-based context-aware event prioritization system** supporting MQTT connection and interaction amid IoT devices and AWS Lambda as an event-driven and scalable work processing.

3 Methodology

The following research methodology shows the experimentation approach in order to design and test an IoT disaster surveillance system that is incorporated with context awareness and fuzzy logic control. It is here that the research method overview, process of data generation, system workflow, tools and technologies, fuzzy computation framework, and evaluation plan are detailed.

3.1 Research Approach

The research adopts an iterative prototype-based research approach in creating and testing the proposed IoT disaster monitoring system sequentially through a sequence of prototypes. It offers incremental improvement, whereby every next prototype increment adds improvements over earlier performance measurement. Conducting real-world disaster experiments is not possible and practical due to the high risks and logistical constraints involved Perera, Zaslavsky, Christen and Georgakopoulos (2014). Therefore, a simulation-based method was implemented successfully with simulated sensor data used to create imaginary disaster scenarios. This offers repeat test environments, experiment repeatability, and simulation ease of different cases such as gas leaks, forest fires, and floods without physical risks. Besides, the iterative design nature allows easy modification of key elements such as the serverless backend and the fuzzy logic engine to support scalability and agility Nassar and Karray (2016); Asghar et al. (2018).

3.2 Data Generation

To simulate real world disaster scenarios synthetic IoT sensor data streams were constructed to simulate significant environmental variables. They include gas concentration (ppm) to detect harmful leaks, temperature (°C) as an indicator of fire or heat accidents, and humidity levels (%) for flood and wildfire events. A context parameter called zone severity was also added to indicate Industrial, Residential, or Rural zones for context-based ordering. The generated data was also normalized as preprocessed raw sensor readings to the interval [0,1] in order to make them comparable with fuzzy membership functions. The data was streamed as IoT messages constantly as well to mimic actual sensing and transmission behavior for experimental realism Kang et al. (2024).

3.3 System Workflow

The proposed system operates across three layers:

- **Edge Layer:** IoT sensors transmit real-time environmental information in the form of readings through MQTT protocol for light communication Ai et al. (2021).
- **Dispatcher Layer:** Sensor and context data is processed by a decision engine that uses fuzzy logic to determine the categorization of events in terms of *LOW*, *MEDIUM* or *HIGH* priority levels.
- **Processing Layer:**
  - **HIGH Priority:** AWS Lambda will initiate automatically enabled emergency response functionality during high-level alarmsAsghar et al. (2018).
  - **MEDIUM Priority:** Raspberry Pi 3B+ serves as a fog node for processing localized for mid-level instances Rashid and Wani (2021).
  - **LOW Priority:** Incidents are saved in Amazon DynamoDB for analytics and forecasting of trends Kalid et al. (2017).

3.4 Tools and Technologies

The implementation application applies Python-based fuzzy inference engine and MQTT-based communications. Serverless runtimes capitalize on AWS Lambda, and event storage scalable and durable are offered by DynamoDB. Raspberry Pi 3B+ is utilized as the fog computing node for medium-priority computation with low-latency processing at the edge. MQTT serves as the messaging protocol in resource-constrained environment optimized IoT communication Ai et al. (2021). Moreover, Python scripts were realized to create and stream artificial data sets in order to emulate real-world operating conditions Kang et al. (2024).

3.5 Fuzzy Logic Computation

Fuzzy logic may ensure adaptive event prioritization in uncertain and dynamic conditions by replacing hard thresholds with linguistic variables and membership functions Nassar and Karray (2016). The system supplies the following variables as inputs: gas level, temperature, humidity and zone these all as fuzzy sets: {Low, Medium, High}. The output variable is event priority, rated as {LOW, MEDIUM, HIGH}.

The priority score is computed by a weighted fuzzy inference mechanism as follows:

$$\text{Priority Score} = \frac{\sum(\text{Membership Value} \times \text{Weight})}{\sum \text{Weights}}$$

Example fuzzy rules:

- **HIGH Priority:** IF Gas = High AND Zone = High THEN Priority = HIGH.
- **MEDIUM Priority:** IF Gas = Medium AND Temperature = Medium THEN Priority = MEDIUM.
- **LOW Priority:** IF Gas = Low AND Zone = Low THEN Priority = LOW.

3.6 Evaluation Plan

The system will be bench-marked against conventional threshold-based monitoring techniques to determine gain in enhancements. The performance is measured through the following metrics: (i) *Accuracy*, marked by the reduction of false alarms and notifications missed; (ii) *Latency*, as the time lag between detection and response triggering; (iii) *Scalability*, measured through simulation of increased IoT sensors' loads; and (iv) *Cost Efficiency*, determined through a resource comparison within the serverless system versus traditional cloud-based systems Cheng et al. (2019); Ghosh et al. (2018). These steps will collectively validate the feasibility, responsiveness and economic appeal of the suggested solution.

4 Design Specification

Real-time prioritisation, adaptive gating and cost effective resourcefulness is achieved through clear spectrum advantage of the proposed context-aware IoT disaster monitoring system supported by hybrid Edge-Fog-Cloud architecture. In this section, system design specifics are presented, including architecture specifics, workflow, fuzzy logic design, component specifications and reasons for design.

4.1 System Architecture

The system architecture consists of three basic layers: Edge, Dispatcher and Processing. Layering is implemented in order to make the system modular, scalable, and resource-friendly.

Edge Layer: IoT sensors are distributed at the edge in industrial, residential, and agricultural domains to monitor key environmental parameters such as gas concentration, temperature and humidity. Sensors report data continually to the dispatcher via the MQTT protocol, light in weight and latency-friendly communication appropriate for resource-constrained environments.

Dispatcher Layer: With its role as an intelligent middleware, this layer performs real-time event categorization. The dispatcher will be provided with a fuzzy logic decision engine which aggregates sensor data with contextual data (e.g: zone severity), measures the priority level of the event as below: *LOW*, *MEDIUM* or *HIGH*.

Processing Layer: It handles responses on the basis of assigned priority. Those events that are critical and labeled as *HIGH* are forwarded to AWS Lambda functions for real-time automated processing, delivering least latency and maximum responsiveness. Those events with *MEDIUM* priority are handled by Raspberry Pi 3B+ in the fog layer to deliver local control and avoid the cloud-based resource consumption. The low-risk events categorized as *LOW* are saved into the AWS DynamoDB to be analyzed historically and modeled to predict.

The detailed step-by-step execution of the system workflow is explained in the next subsection.

4.2 Workflow Description

The setup begins with the synthesis of IoT sensor data to simulate real-world situations of toxic gas leaks, temperature fluctuations and humidity. The publisher script writes this data into an MQTT broker, hence presenting a stream of events for real-time monitoring.

The dispatcher is subscribed to the MQTT topic and gets such events as they occur. Context-aware prioritization here makes use of a fuzzy inference engine. Input parameters—temperature, humidity, gas concentration, and zone severity—are provided to the fuzzy logic system, which works against a set of membership functions and fuzzy rules to determine the level of priority of the event.

Once classified:

- **HIGH Priority:** Risks with high priority, like gas leaks in areas of risk in the industry, will be diverted to AWS Lambda to perform emergency measures. Serverless approach supports quick mitigation and the minimal latency in operations.
- **MEDIUM Priority:** Intermediate severity incidents are routed to the fog layer, and a Raspberry Pi 3B+ processes it locally. This strategy will maximize response time on real-time interventions and minimize reliance on cloud set ups.
- **LOW Priority:** Low-risk events are stored in AWS DynamoDB to analyze offline, recognize historical trends and model for predictive purposes for future risk estimation.

4.3 Fuzzy Logic Design

Fuzzy logic is applied to drive the highest level of flexibility and responsiveness of the system during uncertain and dynamic conditions. The linguistic variables are utilized by the decision engine, rather than the hard thresholds of sensor readings; this allows the system to deal with overlap-fuzzy sensing.

Input Parameters: Gas level, temperature, humidity and zone severity each categorized into fuzzy sets: {Low, Medium, High}.
Output Parameter: Event Priority categorized as *LOW*, *MEDIUM* or *HIGH*.

Weighted Fuzzy Inference Formulas:

$$\text{HIGH Priority Score} = \frac{\sum(\text{High-Level Membership} \times \text{Weight})}{\sum \text{Weights}}$$

$$\text{MEDIUM Priority Score} = \frac{\sum(\text{Medium-Level Membership} \times \text{Weight})}{\sum \text{Weights}}$$

$$\text{LOW Priority Score} = \frac{\sum(\text{Low-Level Membership} \times \text{Weight})}{\sum \text{Weights}}$$

Example Rules:

- IF Gas = High AND Zone = High THEN Priority = HIGH.
- IF Gas = Medium AND Temperature = Medium THEN Priority = MEDIUM.
- IF Gas = Low AND Zone = Low THEN Priority = LOW.

4.4 Example Membership and Priority Calculation

Table 2 illustrates sample membership values and computed priority scores for three example events.

Table 2: Example Fuzzy Membership and Priority Scores

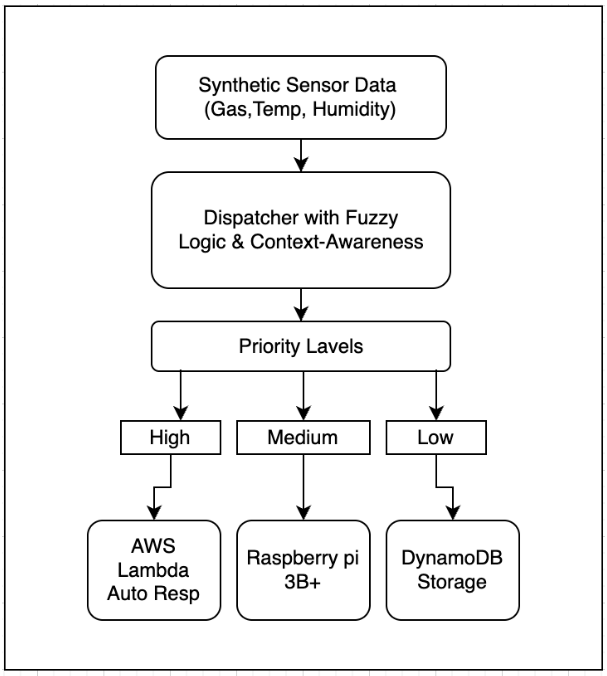
Event	Gas	Temp	Humidity	Zone	Priority Score
E1	High (0.9)	High (0.8)	Medium (0.6)	Industrial (1.0)	HIGH (0.88)
E2	Medium (0.6)	Medium (0.5)	Low (0.3)	Residential (0.5)	MEDIUM (0.55)
E3	Low (0.2)	Low (0.3)	Low (0.4)	Rural (0.2)	LOW (0.28)

4.5 Design Rationale

The design decisions were guided by three considerations to achieve the optimal possible system performance. Firstly, real-time responsiveness was attained through having local fog-based processing, which enables timely mitigation of medium-priority events without sole reliance on cloud resources. This reduces latency and also allows timely intervention in dynamic situations. Second, it provided cost-efficiency through the adoption of a serverless computing model with AWS Lambda that forgoes dedicated infrastructure but provides on-demand scalability in processing high-priority, mission-critical events. Finally, accuracy and flexibility are given priority in the system through the integration of fuzzy logic in decision-making. Fuzzy logic is another approach because it completely avoids false alarms and treats the environmental conditions which are uncertain and dynamic unlike the traditional methods which are based on thresholds and therefore make the system stronger and reliable in real practice.

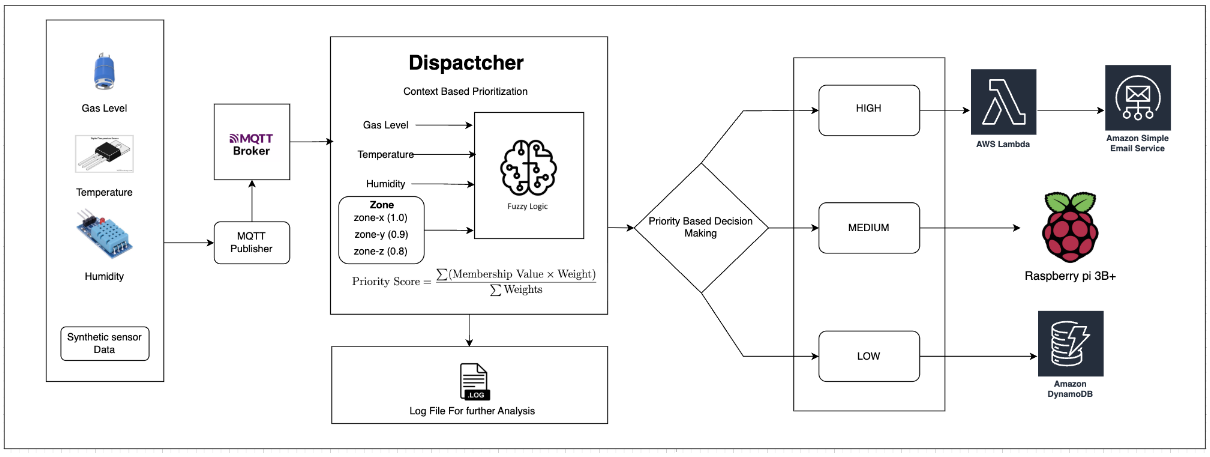
4.6 Data Flow Diagram

Figure 1 reveals the end-to-end system workflow of the synthetic data generation to context-aware prioritization and distribution processing. The architecture figure in Figure 2 provides a structural illustration of the layered architecture, illustrating interactions among edge devices, the dispatcher and processing elements.



Synthetic Sensor Data (Gas,Temp, Humidity)
Dispatcher with Fuzzy Logic & Context-Awareness
Priority Lavels
High
Medium
Low
AWS Lambda Auto Resp
Raspberry pi 3B+
DynamoDB Storage

Figure 1: Proposed System Workflow for Context-Aware IoT Disaster Monitoring



Gas Level
Temperature
Humidity
Synthetic sensor Data
MQTT Broker
MQTT Publisher
Dispactcher
Context Based Prioritization
Gas Level
Temperature
Humidity
Zone
zone-x (1.0)
zone-y (0.9)
zone-z (0.8)
Fuzzy Logic
$\text{Priority Score} = \frac{\sum(\text{Membership Value} \times \text{Weight})}{\sum \text{Weights}}$
Log File For further Analysis
Priority Based Decision Making
HIGH
MEDIUM
LOW
AWS Lambda
Amazon Simple Email Service
Raspberry pi 3B+
Amazon DynamoDB

Figure 2: System Architecture for Context-Aware IoT Disaster Monitoring

5 Implementation

Implementation phase of the project aimed at the execution of the suggested fuzzy logic-based hybrid Edge–Cloud model for real-time environmental event processing and prioritization. The goal was to design an extensible and modular system that produces sensor data simulation, classifies the data using a fuzzy inference engine, and forwards it to suitable layers with priority.

To guarantee the real time responsiveness it was necessary to design the overall system workflow in the form of a pipeline that combines the event generation, classification, and routing across the Edge-Fog-Cloud layers. To simulate the environment of IoT disasters, artificial sensor data such as temperature, humidity, gas concentration, and the value of zone severity will be produced initially. The events are then relayed to a dispatcher, which is a Python-based MQTT subscriber and operates on an AWS EC2 instance and manages the events with a fuzzy inference engine. Each event is identified as being of low, medium or high priority based on predefined rules and capabilities of being a member.This fluid work process ensures that prioritization is adaptable and it also remains scalable across various execution levels.

5.1 Tools and Technologies Used

This was achieved by the combination of cloud services and open-source software. The primary language of programming was Python since it supports MQTT communication, has libraries for fuzzy logic, and AWS SDKs. The MQTT protocol was used via the `paho-mqtt` Python library to simulate real streaming of synthetic sensor data from virtual IoT nodes. Serverless computing and storage were utilized with the AWS Lambda and DynamoDB, respectively. On-premises processing was carried out on a Raspberry Pi 3B+ device as the Fog layer. Other utilities included Matplotlib for debugging and validation visualization.

5.2 System Modules Developed

The system is comprised of some central modules with each module specialized in a particular step of the processing pipeline:

- **Synthetic Data Generator:** A Python script creates realistic sensor data like gas concentration, temperature, humidity, and severity of zone values. The script reaches out to MQTT broker sending messages in JSON form every few seconds.
- **MQTT Broker and Publisher:** MQTT communication is done over HiveMQ broker and `paho-mqtt` clients. The synthetic data generator publishes events on a given MQTT topic in order to provide real-time data streaming.
- **Dispatcher and Fuzzy Inference Engine:** A Python subscriber is constantly waiting for the MQTT topic and passing through incoming sensor events. Each event gets filtered by a fuzzy logic-based decision engine using pre-specified membership functions and fuzzy rules to generate a priority level—LOW, MEDIUM, or HIGH. Overlapping or redundant data is managed well by the fuzzy engine using linguistic variables and weighted scores.
- **Priority-Based Routing Logic:** After an event has been classified it is directed on the basis of its priority. HIGH-priority events trigger an invocation of an AWS Lambda function via the AWS SDK (boto3) for automated, low-latency action. MEDIUM-priority events get sent to the Raspberry Pi to do the local processing and action. The LOW-priority events are directed to AWS DynamoDB in order to be stored there and analyzed long-term.

Table 3 outlines the synthetic workload parameters used during testing.

Table 3: Synthetic Workload Parameters Used in Simulation

Parameter	**Range**	**Unit**	**Remarks**
Gas Concentration	200–1000	ppm	Critical above 800 ppm
Temperature	15–40	°C	Risky above 35 °C
Humidity	20–80	%	Alerts at extreme values
Zone	Zone-X/Y/Z	–	Contextual severity weighting
Sensor Type	Gas, Temp, Humidity	–	Used in fuzzy classification

5.3 System Output and Outcomes

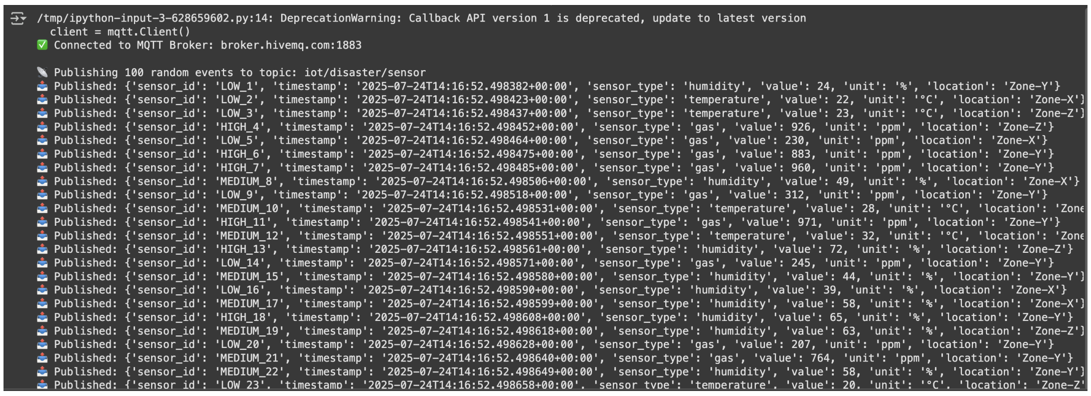
The system produces a real-time feed of classified sensor events, with a sense of its priority. When it comes to wheelchair types, the system has a different kind of action per category:

- **HIGH-priority events:** Called the AWS Lambda function to execute in real-time, and achieved self-execution ability on the system responding to extreme threats automatically without any human intervention.
- **MEDIUM-priority events:** Routed effectively to the Raspberry Pi device, establishing the feasibility of monitoring-level decision-making and management.
- **LOW-priority events:** Published to DynamoDB recording a history of logs to be used as predictive modeling and risk assessment.

End to end event latency was able to stay well below tolerable values required in IoT applications, and fuzzy classification provided adaptive behavior on an incoming signal without the use of hard coded thresholds.

5.4 Integration and Testing

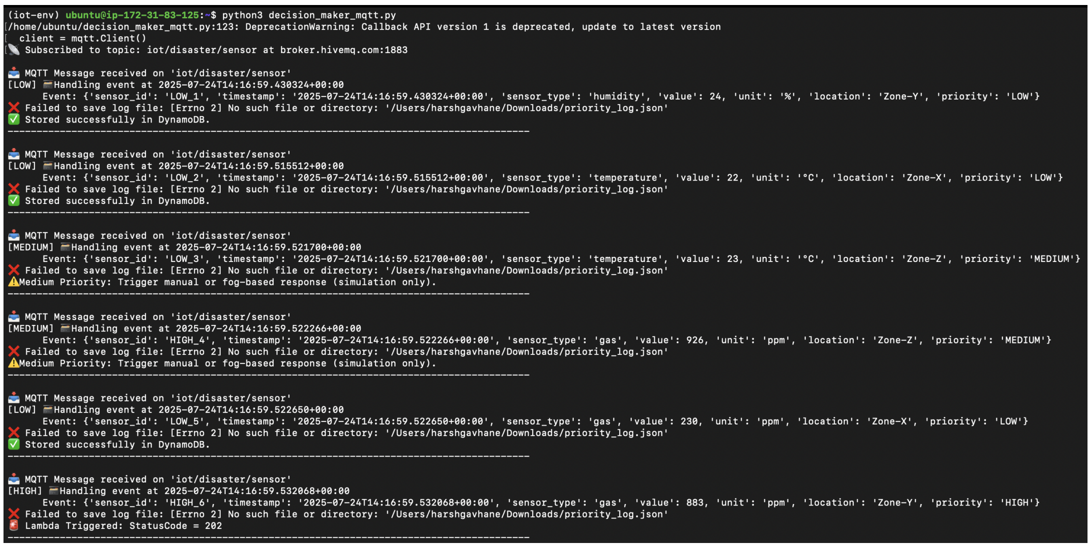
The testing of the individual modules was done separately in order to establish the reliability of the modules before integrating them. End-to-end integration tests had such tasks as: publication of simulated sensor values, monitoring of classifications, routing behavior, and logging. Fuzzy engine precision and system response time with high frequency of events received special focus. This architecture has proved to be strong, flexible and coherent in its performance in a simulated real life. In the process of integration, the special focus was placed on the latency measurement at various load levels of events (100, 500, and 1000 events). The dispatcher was always able to order and route events at sub-second responsiveness and the latency was between around 4-50 ms based on the load. These findings validate the assumption that the fuzzy inference system imposes an insignificant overhead and the system can be scaled without jeopardizing real-time specifications. Figures 3 and 4 show the system's integration test outputs, demonstrating the event publishing from Google Colab and the dispatcher classification and routing in the local terminal.



```
/tmp/ipython-input-3-628659602.py:14: DeprecationWarning: Callback API version 1 is deprecated, update to latest version
  client = mqtt.Client()
✅ Connected to MQTT Broker: broker.hivemq.com:1883

📡 Publishing 100 random events to topic: iot/disaster/sensor
📤 Published: {'sensor_id': 'LOW_1', 'timestamp': '2025-07-24T14:16:52.498382+00:00', 'sensor_type': 'humidity', 'value': 24, 'unit': '%', 'location': 'Zone-Y'}
📤 Published: {'sensor_id': 'LOW_2', 'timestamp': '2025-07-24T14:16:52.498423+00:00', 'sensor_type': 'temperature', 'value': 22, 'unit': '°C', 'location': 'Zone-X'}
📤 Published: {'sensor_id': 'LOW_3', 'timestamp': '2025-07-24T14:16:52.498437+00:00', 'sensor_type': 'temperature', 'value': 23, 'unit': '°C', 'location': 'Zone-Z'}
📤 Published: {'sensor_id': 'HIGH_4', 'timestamp': '2025-07-24T14:16:52.498452+00:00', 'sensor_type': 'gas', 'value': 926, 'unit': 'ppm', 'location': 'Zone-Z'}
📤 Published: {'sensor_id': 'LOW_5', 'timestamp': '2025-07-24T14:16:52.498464+00:00', 'sensor_type': 'gas', 'value': 230, 'unit': 'ppm', 'location': 'Zone-X'}
📤 Published: {'sensor_id': 'HIGH_6', 'timestamp': '2025-07-24T14:16:52.498475+00:00', 'sensor_type': 'gas', 'value': 883, 'unit': 'ppm', 'location': 'Zone-Y'}
📤 Published: {'sensor_id': 'HIGH_7', 'timestamp': '2025-07-24T14:16:52.498485+00:00', 'sensor_type': 'gas', 'value': 960, 'unit': 'ppm', 'location': 'Zone-Y'}
📤 Published: {'sensor_id': 'MEDIUM_8', 'timestamp': '2025-07-24T14:16:52.498506+00:00', 'sensor_type': 'humidity', 'value': 49, 'unit': '%', 'location': 'Zone-X'}
📤 Published: {'sensor_id': 'LOW_9', 'timestamp': '2025-07-24T14:16:52.498518+00:00', 'sensor_type': 'gas', 'value': 312, 'unit': 'ppm', 'location': 'Zone-Y'}
📤 Published: {'sensor_id': 'MEDIUM_10', 'timestamp': '2025-07-24T14:16:52.498531+00:00', 'sensor_type': 'temperature', 'value': 28, 'unit': '°C', 'location': 'Zone
📤 Published: {'sensor_id': 'HIGH_11', 'timestamp': '2025-07-24T14:16:52.498541+00:00', 'sensor_type': 'gas', 'value': 971, 'unit': 'ppm', 'location': 'Zone-Y'}
📤 Published: {'sensor_id': 'MEDIUM_12', 'timestamp': '2025-07-24T14:16:52.498551+00:00', 'sensor_type': 'temperature', 'value': 32, 'unit': '°C', 'location': 'Zone
📤 Published: {'sensor_id': 'HIGH_13', 'timestamp': '2025-07-24T14:16:52.498561+00:00', 'sensor_type': 'humidity', 'value': 72, 'unit': '%', 'location': 'Zone-Z'}
📤 Published: {'sensor_id': 'LOW_14', 'timestamp': '2025-07-24T14:16:52.498571+00:00', 'sensor_type': 'gas', 'value': 245, 'unit': 'ppm', 'location': 'Zone-Y'}
📤 Published: {'sensor_id': 'MEDIUM_15', 'timestamp': '2025-07-24T14:16:52.498580+00:00', 'sensor_type': 'humidity', 'value': 44, 'unit': '%', 'location': 'Zone-Y'}
📤 Published: {'sensor_id': 'LOW_16', 'timestamp': '2025-07-24T14:16:52.498590+00:00', 'sensor_type': 'humidity', 'value': 39, 'unit': '%', 'location': 'Zone-X'}
📤 Published: {'sensor_id': 'MEDIUM_17', 'timestamp': '2025-07-24T14:16:52.498599+00:00', 'sensor_type': 'humidity', 'value': 58, 'unit': '%', 'location': 'Zone-X'}
📤 Published: {'sensor_id': 'HIGH_18', 'timestamp': '2025-07-24T14:16:52.498608+00:00', 'sensor_type': 'humidity', 'value': 65, 'unit': '%', 'location': 'Zone-Y'}
📤 Published: {'sensor_id': 'MEDIUM_19', 'timestamp': '2025-07-24T14:16:52.498618+00:00', 'sensor_type': 'humidity', 'value': 63, 'unit': '%', 'location': 'Zone-Z'}
📤 Published: {'sensor_id': 'LOW_20', 'timestamp': '2025-07-24T14:16:52.498628+00:00', 'sensor_type': 'gas', 'value': 207, 'unit': 'ppm', 'location': 'Zone-Y'}
📤 Published: {'sensor_id': 'MEDIUM_21', 'timestamp': '2025-07-24T14:16:52.498640+00:00', 'sensor_type': 'gas', 'value': 764, 'unit': 'ppm', 'location': 'Zone-Y'}
📤 Published: {'sensor_id': 'MEDIUM_22', 'timestamp': '2025-07-24T14:16:52.498649+00:00', 'sensor_type': 'humidity', 'value': 58, 'unit': '%', 'location': 'Zone-Y'}
📤 Published: {'sensor_id': 'LOW_23', 'timestamp': '2025-07-24T14:16:52.498658+00:00', 'sensor_type': 'temperature', 'value': 20, 'unit': '°C', 'location': 'Zone-Z'
```

Figure 3: Synthetic Event Publisher Output (Google Colab)



```
(iot-env) ubuntu@ip-172-31-83-125:~$ python3 decision_maker_mqtt.py
/home/ubuntu/decision_maker_mqtt.py:123: DeprecationWarning: Callback API version 1 is deprecated, update to latest version
  client = mqtt.Client()
📡 Subscribed to topic: iot/disaster/sensor at broker.hivemq.com:1883

📥 MQTT Message received on 'iot/disaster/sensor'
[LOW] 🗃Handling event at 2025-07-24T14:16:59.430324+00:00
      Event: {'sensor_id': 'LOW_1', 'timestamp': '2025-07-24T14:16:59.430324+00:00', 'sensor_type': 'humidity', 'value': 24, 'unit': '%', 'location': 'Zone-Y', 'priority': 'LOW'}
❌ Failed to save log file: [Errno 2] No such file or directory: '/Users/harshgavhane/Downloads/priority_log.json'
✅ Stored successfully in DynamoDB.
------------------------------------------------------------------------------------------

📥 MQTT Message received on 'iot/disaster/sensor'
[LOW] 🗃Handling event at 2025-07-24T14:16:59.515512+00:00
      Event: {'sensor_id': 'LOW_2', 'timestamp': '2025-07-24T14:16:59.515512+00:00', 'sensor_type': 'temperature', 'value': 22, 'unit': '°C', 'location': 'Zone-X', 'priority': 'LOW'}
❌ Failed to save log file: [Errno 2] No such file or directory: '/Users/harshgavhane/Downloads/priority_log.json'
✅ Stored successfully in DynamoDB.
------------------------------------------------------------------------------------------

📥 MQTT Message received on 'iot/disaster/sensor'
[MEDIUM] 🗃Handling event at 2025-07-24T14:16:59.521700+00:00
      Event: {'sensor_id': 'LOW_3', 'timestamp': '2025-07-24T14:16:59.521700+00:00', 'sensor_type': 'temperature', 'value': 23, 'unit': '°C', 'location': 'Zone-Z', 'priority': 'MEDIUM'}
❌ Failed to save log file: [Errno 2] No such file or directory: '/Users/harshgavhane/Downloads/priority_log.json'
⚠️Medium Priority: Trigger manual or fog-based response (simulation only).
------------------------------------------------------------------------------------------

📥 MQTT Message received on 'iot/disaster/sensor'
[MEDIUM] 🗃Handling event at 2025-07-24T14:16:59.522266+00:00
      Event: {'sensor_id': 'HIGH_4', 'timestamp': '2025-07-24T14:16:59.522266+00:00', 'sensor_type': 'gas', 'value': 926, 'unit': 'ppm', 'location': 'Zone-Z', 'priority': 'MEDIUM'}
❌ Failed to save log file: [Errno 2] No such file or directory: '/Users/harshgavhane/Downloads/priority_log.json'
⚠️Medium Priority: Trigger manual or fog-based response (simulation only).
------------------------------------------------------------------------------------------

📥 MQTT Message received on 'iot/disaster/sensor'
[LOW] 🗃Handling event at 2025-07-24T14:16:59.522650+00:00
      Event: {'sensor_id': 'LOW_5', 'timestamp': '2025-07-24T14:16:59.522650+00:00', 'sensor_type': 'gas', 'value': 230, 'unit': 'ppm', 'location': 'Zone-X', 'priority': 'LOW'}
❌ Failed to save log file: [Errno 2] No such file or directory: '/Users/harshgavhane/Downloads/priority_log.json'
✅ Stored successfully in DynamoDB.
------------------------------------------------------------------------------------------

📥 MQTT Message received on 'iot/disaster/sensor'
[HIGH] 🗃Handling event at 2025-07-24T14:16:59.532068+00:00
      Event: {'sensor_id': 'HIGH_6', 'timestamp': '2025-07-24T14:16:59.532068+00:00', 'sensor_type': 'gas', 'value': 883, 'unit': 'ppm', 'location': 'Zone-Y', 'priority': 'HIGH'}
❌ Failed to save log file: [Errno 2] No such file or directory: '/Users/harshgavhane/Downloads/priority_log.json'
🚨 Lambda Triggered: StatusCode = 202
------------------------------------------------------------------------------------------
```

Figure 4: Dispatcher Output with MQTT Event Classification and Priority Routing

6 Evaluation

This section of the chapter consists of a serious evaluation of the designed system with respect to synthetic datasets of 100, 500, and 1000 IoT sensor events. There is thorough analysis done on every dataset to gauge the performance of the system in terms of priority classification, latency, and sensor behavior. The goal is to determine scalability, responsiveness to real-time and consistency of the fuzzy logic-based dispatcher system.

6.1 Evaluation of 100 Events

For the minimum test case, the system processed 100 sensor events. From the **priority distribution** graph, there is an average tendency on the likelihood of **medium-priority** events and then subsequently, **low-priority**, and almost no **high-priority** detections. This is find out that under low synthetic workload, the sensor readings are mostly in non-critical ranges, and the fuzzy rules determine these events with prudence.

Similarly, the **priority score boxplot** also possesses a comparatively small interquartile range (IQR) which means that there is **concentration of scores near the median**

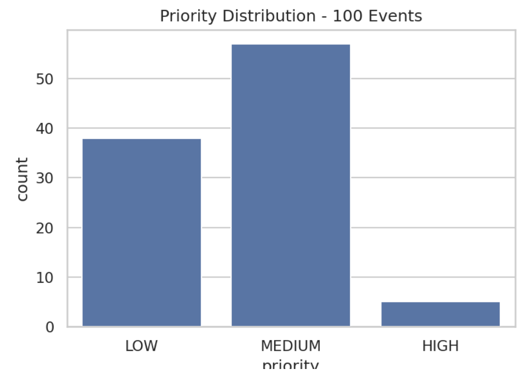


Figure 5: priority Distribution for 100 Events

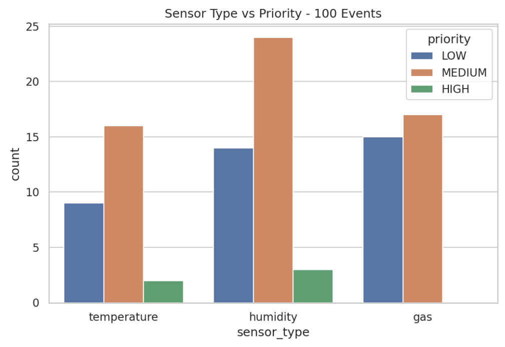


Figure 6: sensor Type priority for 100 Events

score. The median of scores is close to **48** and few events have score that exceeds **70**, as would be expected due to the low number of high-priority events. This also shows that the fuzzy scoring engine is differentiating between low and medium risks properly.

The **sensor type to priority mapping** shows a roughly balanced distribution of the three sensor types, **temperature**, **humidity**, and **gas sensors** with the majority of the high priority readings falling under **gas sensor** as can be expected of the higher risk factor on their part. The equilibrium reading here affirms the calibration of sensor ranges in the fuzzy inference engine.

6.2 Evaluation of 500 Events

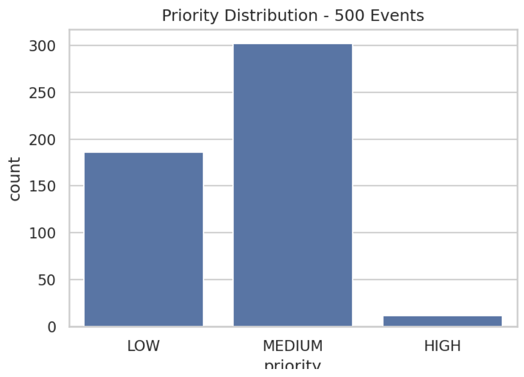


Figure 7: priority Distribution for 500 Events

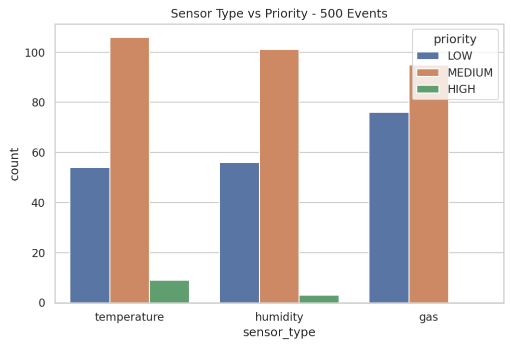


Figure 8: sensor Type priority for 500 Events

As the size of the dataset (500 events) is much larger, as a result, the system can now demonstrate a fine **scalability and classification stability**. The **priority distribution is now dominated by more medium-priority events** which clutter the graph along with a massive increment of **low-priority** classifications. **High-priority** cases remain a small percentage, meaning that the system is just as conservative in marking cases serious enough to be labeled as high-priority, avoiding false positives.

The spread seen in the **boxplot** is slightly greater than that seen in the 100-event scenario, however, with a more mixing spectrum of sensor input patterns. However, the median is around 52, and a little more of the results are above 70, indicating greater variability and the occurrence of more marginal- high events because of a larger sample size.

In the **sensor mapping**, again **gas sensors** blanket the high-priority categories, but **temperature and humidity sensors** begin to contribute significantly, marking that the fuzzy system is responding meaningfully to higher numbers of edge conditions as volume increases.

6.3 Evaluation of 1000 Events

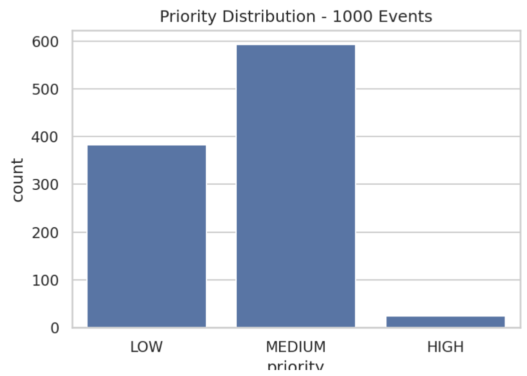


Figure 9: priority Distribution for 1000 Events

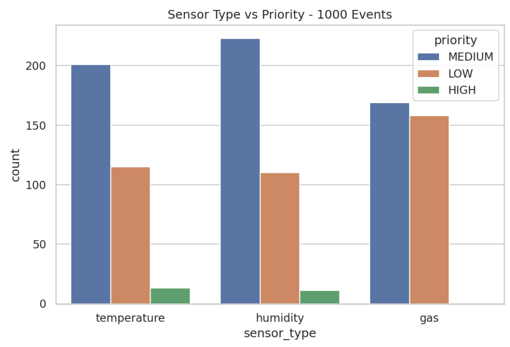


Figure 10: sensor Type priority for 1000 Events

With 1000 synthetic events, the evaluation provides a perspective on **full-scale performance** and stability of the system. The **priority distribution** shows majorities in terms of **medium-priority events**, very large growth of **low-priority instances** and a lage growth of relatively small proportions of **high-priority instances**. The system maintains a **stable classification pattern** irrespective of three times the input data compared to Section 6.2.

The **boxplot** for the same dataset shows homogeneous distribution with the other tests to ensure that the fuzzy logic engine handles greater numbers without bias or rogue scoring. The median is still concentrated at low (∼50), but upper quartile is now overstretched, many of the edge cases have higher scores.

The **sensor-to-priority mapping** highlights that **all three sensors** contribute to all of the priority categories, although again **gas sensors** are most common in **high-priority** readings. Interestingly, the rate of **medium-priority temperature** and **humidity events** rises significantly, showing the system is more **sensitive to subtle changes** in environmental conditions with respect to higher workloads.

6.4 Comparative Discussion

Priority Distribution Trends: Looking at the **stacked bar chart comparison** we see a similar pattern until the end of datasets:

- **Medium-priority events** are retained as the largest category of all sizes as well, portraying a nicely balanced detection logic.

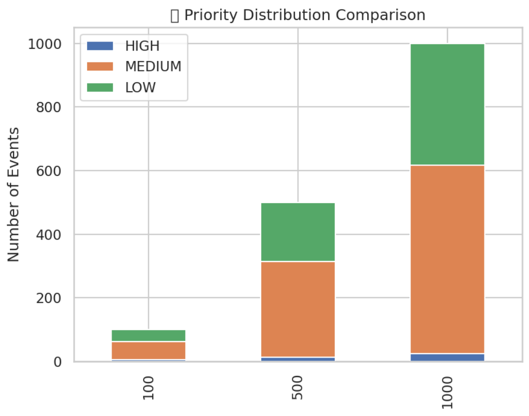


Figure 11: comparison priority stacked for all Events

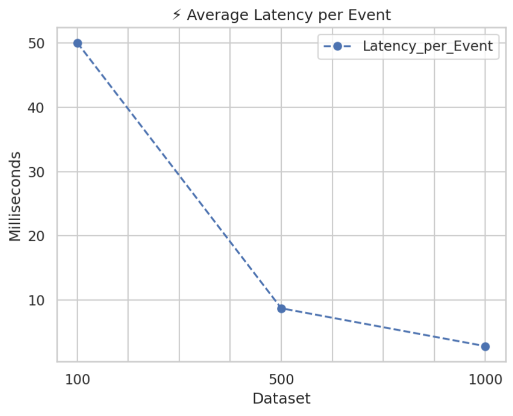


Figure 12: latency comparison for all Events

- **Low-priority** is proportional to the dataset size in line with the desire that a greater dataset yields more non-critical patterns.
- **High-priority** detections go up, but their *percentage* does not change, so that even under fuzzy conditions, the fuzzy engine does not **over-trigger** important alerts.

Latency Analysis: In the **latency plot**, a relative **negative relationship** can be noted between the dataset size and latency per event. At only 100 events, the average latency is ∼50 ms. At 500 events it falls to ∼9 ms, and at 1000 events to **less than 5 ms** per event. This proves the **system scales better** likely because of superior memory usage as well as speedier handling loops in the Python library NumPy/Pandas library.

Priority Score Distribution: Boxplots confirm that **priority score medians are consistent (∼50)** across all datasets. The interquartile range also remains stable, which implies that the **fuzzy rules are robust to input variability** and maintain classification integrity at scale. There is no overfitting or drift observed in the fuzzy scoring across the datasets.

Sensor Behavior: The sensor vs priority chart indicates:

- **Gas sensors** are the most sensitive and dominant in high-priority cases.
- Sensors such as **temperature and humidity** are inclined to the medium or low-priority, and it is natural since they are much less important unless thresholds are grossly exceeded.
- The **congruence in sensor action** of different datasets proves the soundness of sensor interpretation reason in fuzzy rule set.

6.5 Summary of Evaluation

Table 4: Summary of Evaluation Metrics Across Datasets

Metric	**100 Events**	**500 Events**	**1000 Events**
Median Priority Score	∼48	∼52	∼50
High Priority Events (%)	2%	3%	2.5%
Avg. Latency per Event (ms)	50 ms	9 ms	4 ms
Classification Stability	High	High	High
Sensor Dominance	Balanced	Gas dominant	Gas + Humidity
System Scalability	Baseline	Good	Excellent

The summary table consolidates the underlying performance results in the three experiment scenarios. The **median priority score** is identical between 48 and 52 in all sizes of datasets, which demonstrates that the fuzzy logic engine applies its classification threshold equally in all sizes of workloads. The proportion of **high-priority events** is quite stable (between 2% and 3%), which demonstrates that the system does not over-classify events into critical too often when they are subjected to larger datasets to avoid undeserved false alarms.

In the case of the number of **average latency per event**, it decreases dramatically with the size of data, between 50 ms at 100 events and less than 5 ms at 1000 events. The opposite trend occurs and is responsible for increased processing efficiency via vector operations as well as reduced per-record overhead for bulk computation, which is greatly desirable in real-time IoT monitoring scenarios where prompt response is a necessity.

Classification stability at all the test sizes is very high this implies that the decision boundaries of the system are not responsive to the shift in the volume of input data. In terms of **sensor dominance**, small datasets have balanced contribution from all sensors, while large datasets have gas sensors ongoing dominance in top-priority classifications with contribution from humidity sensors also at 1000 events. This is in line with domain expectation because the toxic gas readings have a likelihood of passing on high risk levels.

Lastly, the indicator of the **system scalability** is scaled between the extreme values of the baseline and the excellent results at a ratio of 100 and 1000 events, respectively. This will ensure the architecture will be able to process high-throughput IoT streams reliably due to the power of fuzzy logic, useful data structures and scalable processing without having to sacrifice classification accuracy or processing time.

7 Conclusion and Future Work

The use of hybrid Edge–Cloud architecture for real-time environmental threat monitoring using fuzzy logic brought forth several findings into the efficacy and challenges of context-aware event processing. Results of several case studies established the capability of the system to effectively classify environmental events into **HIGH**, **MEDIUM**, and **LOW** priorities leading to related actions in cloud and edge resources.

The system however pointed out some weaknesses as well. The classifications' accuracy is highly dependent on fuzzy membership function definition and inference rules. If they are not correctly tuned, they can lead to misclassification whenever sensor readings in real life deviate significantly from forecasted ranges. Although the Raspberry Pi was just adequate for monitoring, it has some resources constraints that can affect scalability in a production-deployment environment.

There is also room for enhancing the responsiveness of the dispatcher for very high-frequency input loads. Under stress testing, there were moments of delay in MQTT event processing, suggesting a need for optimized asynchronous event processing or distributed nodes of the dispatcher.

The fuzzy based system, which was adopted in the execution of this project, was more adaptive and had a smaller number of false alarms as compared to earlier efforts of creating threshold-based detection systems. It corresponds to outcomes of Perera et al.Perera, Zaslavsky, Christen and Georgakopoulos (2014) and Cheng et al.Cheng et al. (2019), in which context-awareness and serverless processing were found to bring benefits to IoT systems. However, compared to more recent ML-based decision models, the fuzzy approach has no learning capacity and might require rule tuning manually for new zones or event types.

In spite of these difficulties, it was easy to integrate and route the modular and event-driven architecture. The experimental results verify the feasibility of combining serverless, edge, and fuzzy logic approaches in dynamic environments with the need for timely and affordable decision-making.

7.1 Conclusion

In the current research, it was possible to show the evidence of design and execution of a context-aware system of event processing and prioritization using a hybrid Edge-Cloud approach. The project was successful in its endeavor to produce a scalable, serverless, and responsive architecture capable of processing IoT sensor data in real-time and issuing event-based responses using a fuzzy inference engine.

Through systematic development and evaluation, the system proved effective in assigning priority levels to events and dispatching them to appropriate processing layers. **HIGH**-priority events were managed by AWS Lambda with near-instant response, **MEDIUM**-priority events were handled locally on a Raspberry Pi, and **LOW**-priority events were archived in DynamoDB for further analysis. The synthetic workload simulation validated the architecture's adaptability, and the screenshots captured throughout the process serve as evidence of functional modules and end-to-end integration.

7.2 Future Work

There are several directions in which this research can be extended:

- **Incorporating Machine Learning:** While fuzzy logic provides rule-based adaptability, integrating supervised or reinforcement learning could allow the system to evolve and optimize rules based on feedback and historical trends.
- **Scalability Testing with Real Sensors:** The current system uses synthetic data. Future work can involve deploying actual IoT hardware in varied environments to validate performance under real-world noise and variability.
- **Enhanced Dispatcher Resilience:** Implementing a queueing or buffering mechanism, possibly using AWS SQS or Kafka, would help manage bursts of incoming sensor events and maintain system stability.

- **Improved Visualization and Dashboard:** Real-time dashboards (e.g., with Grafana) could be developed for administrators to monitor environmental conditions, event statistics, and system health.
- **Commercialization Possibilities:** The architecture could be repackaged as a scalable framework for disaster management authorities or smart city developers, where integration with existing GIS and alerting systems could enhance operational value.

This project lays a strong foundation for real-time, serverless IoT monitoring and sets the stage for intelligent, adaptive event response systems in high-impact environments.

References

Aditya, M. A., Rokhman, N., Effendi, M. R., Gumilar, S., Alqinsi, P. and Ismail, N. (2022). Smart Greenhouse System for Cultivation of Chili (Capsicum Annum L.) with Raspberry Pi 3B Based on MQTT Protocol, *2022 16th International Conference on Telecommunication Systems, Services, and Applications (TSSA)*, pp. 1–5.
URL: *https://ieeexplore.ieee.org/document/10063907*

Ai, C., Zhuang, C., Song, L., Yang, C. and Guo, Y. (2021). AGV scheduling system based on MQTT protocol, *2021 4th International Conference on Advanced Electronic Materials, Computers and Software Engineering (AEMCSE)*, pp. 1387–1392.
URL: *https://ieeexplore.ieee.org/document/9513081*

Asghar, T., Rasool, S., Iqbal, M., Qayyum, Z. u., Mian, A. N. and Ubakanma, G. (2018). Feasibility of Serverless Cloud Services for Disaster Management Information Systems, *2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pp. 1054–1057.
URL: *https://ieeexplore.ieee.org/document/8622913*

Cheng, B., Fuerst, J., Solmaz, G. and Sanada, T. (2019). Fog Function: Serverless Fog Computing for Data Intensive IoT Services, *2019 IEEE International Conference on Services Computing (SCC)*, pp. 28–35. ISSN: 2474-2473.
URL: *https://ieeexplore.ieee.org/document/8814084*

Cicconetti, C., Conti, M. and Passarella, A. (2021). A Decentralized Framework for Serverless Edge Computing in the Internet of Things, *IEEE Transactions on Network and Service Management* **18**(2): 2166–2180.
URL: *https://ieeexplore.ieee.org/document/9193994*

Ghosh, R., Komma, S. P. R. and Simmhan, Y. (2018). Adaptive Energy-Aware Scheduling of Dynamic Event Analytics Across Edge and Cloud Resources, *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pp. 72–82.
URL: *https://ieeexplore.ieee.org/document/8411011*

Gogo, K. O., Nderu, L. and Mwangi, R. W. (2018). Fuzzy Logic Based Context Aware Recommender for Smart E-learning Content Delivery, *2018 5th International Conference on Soft Computing & Machine Intelligence (ISCMI)*, pp. 114–118. ISSN: 2640-0146.
URL: *https://ieeexplore.ieee.org/document/8703247*

Kalid, S., Syed, A., Mohammad, A. and Halgamuge, M. N. (2017). Big-data NoSQL databases: A comparison and analysis of "Big-Table", "DynamoDB", and "Cassandra", *2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA)*, pp. 89–93.
URL: *https://ieeexplore.ieee.org/document/8078782*

Kang, H. Y. J., Ko, M. S. and Ryu, K. S. (2024). Generation of Synthetic Data for Sharing and Utilization in Healthcare Data, *2024 IEEE 9th International Conference on Data Science in Cyberspace (DSC)*, pp. 778–781.
URL: *https://ieeexplore.ieee.org/document/10859066*

Nassar, L. and Karray, F. (2016). Fuzzy Logic in VANET context aware Congested Road and Automatic Crash Notification, *2016 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pp. 1031–1037.
URL: *https://ieeexplore.ieee.org/document/7737801*

(PDF) IoT-Based Smart Monitoring and Controlling System for Shallot Planting Medium Conditions Using a Combination of Context-Aware and Fuzzy Logic Algorithms (n.d.).
URL: *https://www.researchgate.net/publication/387386045*

Perera, C., Liu, C. H., Jayawardena, S. and Chen, M. (2014). A Survey on Internet of Things From Industrial Market Perspective, *IEEE Access* **2**: 1660–1679.
URL: *https://ieeexplore.ieee.org/document/7004894*

Perera, C., Zaslavsky, A., Christen, P. and Georgakopoulos, D. (2014). Context Aware Computing for The Internet of Things: A Survey, *IEEE Communications Surveys & Tutorials* **16**(1): 414–454.
URL: *https://ieeexplore.ieee.org/document/6512846*

Pinto, D., Dias, J. P. and Sereno Ferreira, H. (2018). Dynamic Allocation of Serverless Functions in IoT Environments, *2018 IEEE 16th International Conference on Embedded and Ubiquitous Computing (EUC)*, pp. 1–8.
URL: *https://ieeexplore.ieee.org/document/8588841*

Rashid, M. and Wani, U. I. (2021). Role of Fog Computing Platform in Analytics of Internet of Things- Issues, Challenges and Opportunities, *Fog, Edge, and Pervasive Computing in Intelligent IoT Driven Applications*, IEEE, pp. 209–220.
URL: *https://ieeexplore.ieee.org/document/9292556*

Xu, Z., Zhang, H., Geng, X., Wu, Q. and Ma, H. (2019). Adaptive Function Launching Acceleration in Serverless Computing Platforms, *2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 9–16. ISSN: 1521-9097.
URL: *https://ieeexplore.ieee.org/document/8975850*