

**Abdullah Gul University**

**Department of Computer Engineering**

**Internet of Things Project Report**

**Intelligent Water Consumption Control using IoT and AI**

**by**

**Erdem Muslu – 2111051015**

**Semih Işık - 2111051066**

**Vedat Ayaz - 2111051002**

**Mustafa Ergut – 2111051039**

**01/08/2026**

**Table of Contents**

**1. Introduction**

**2. System Model**

2.1. Hardware Design

2.1.1. Hardware Selection Methodology

2.1.2. Sensing Unit (ESP32 & Flow Sensor)

2.1.3. Actuation Unit (Raspberry Pi & Valve)

2.1.4. Embedded Code Logic

**3. Software Design**

3.1. Hybrid Edge-Cloud Architecture

3.2. Backend Web Application

3.3.Frontend Web Application

3.4. Machine Learning & TinyML Integration

3.5. System Resilience and Fault Tolerance

**4. Results and Discussion**

4.1. Performance Metrics

4.2. Challenges Faced & Solutions

4.3. Future Scope

4.4Real-World Scenario

**5. Conclusion**

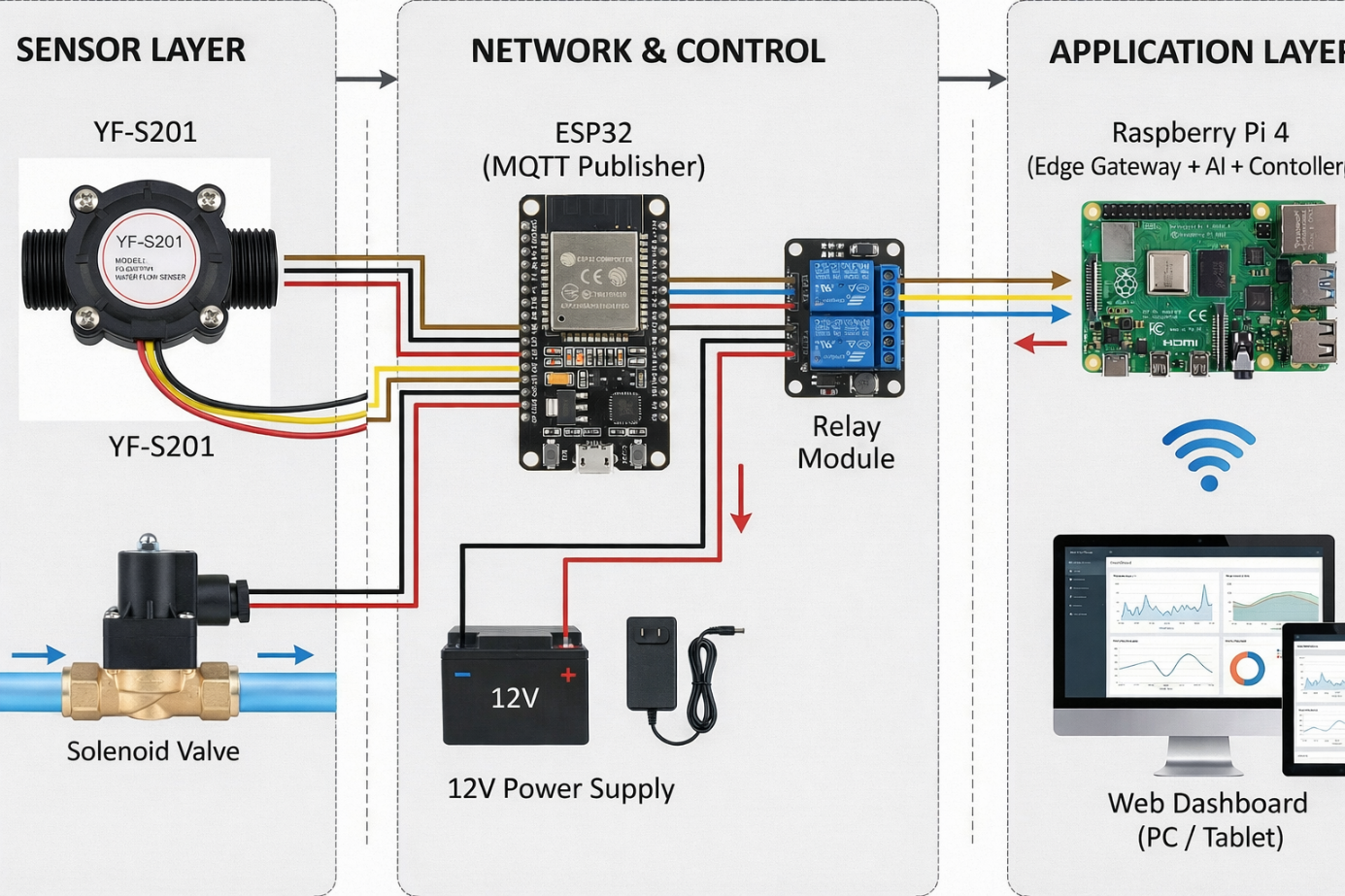
**6. Reference List**

**7. Github and Drive Link**

1. **Introduction**

Water is an indispensable resource for life, yet it faces serious threats from population growth, pollution, and climate change. According to the World Health Organization (WHO), approximately 50% of the global population is expected to experience water scarcity by 2025. Despite this urgency, traditional water management systems remain passive; they only measure consumption for billing purposes without offering active control mechanisms or real-time feedback.

To address this gap, our project, **"Intelligent Water Consumption Control,"** proposes a proactive solution by integrating distributed IoT sensors, active actuators, and Artificial Intelligence. Unlike standard meters, this system is designed to monitor water flow in real-time, detect anomalies using AI, and automatically intervene (e.g., cut off water via a solenoid valve) to prevent leaks and waste.

****

The system architecture is divided into three main layers. An ESP32 microcontroller acts as the sensing node, collecting flow data and transmitting it wirelessly. A Raspberry Pi 4 serves as the central server and edge computing unit, hosting the Web Dashboard and the lightweight AI model (Scikit-learn model). Finally, a solenoid valve functions as the active controller to physically stop the flow when necessary.

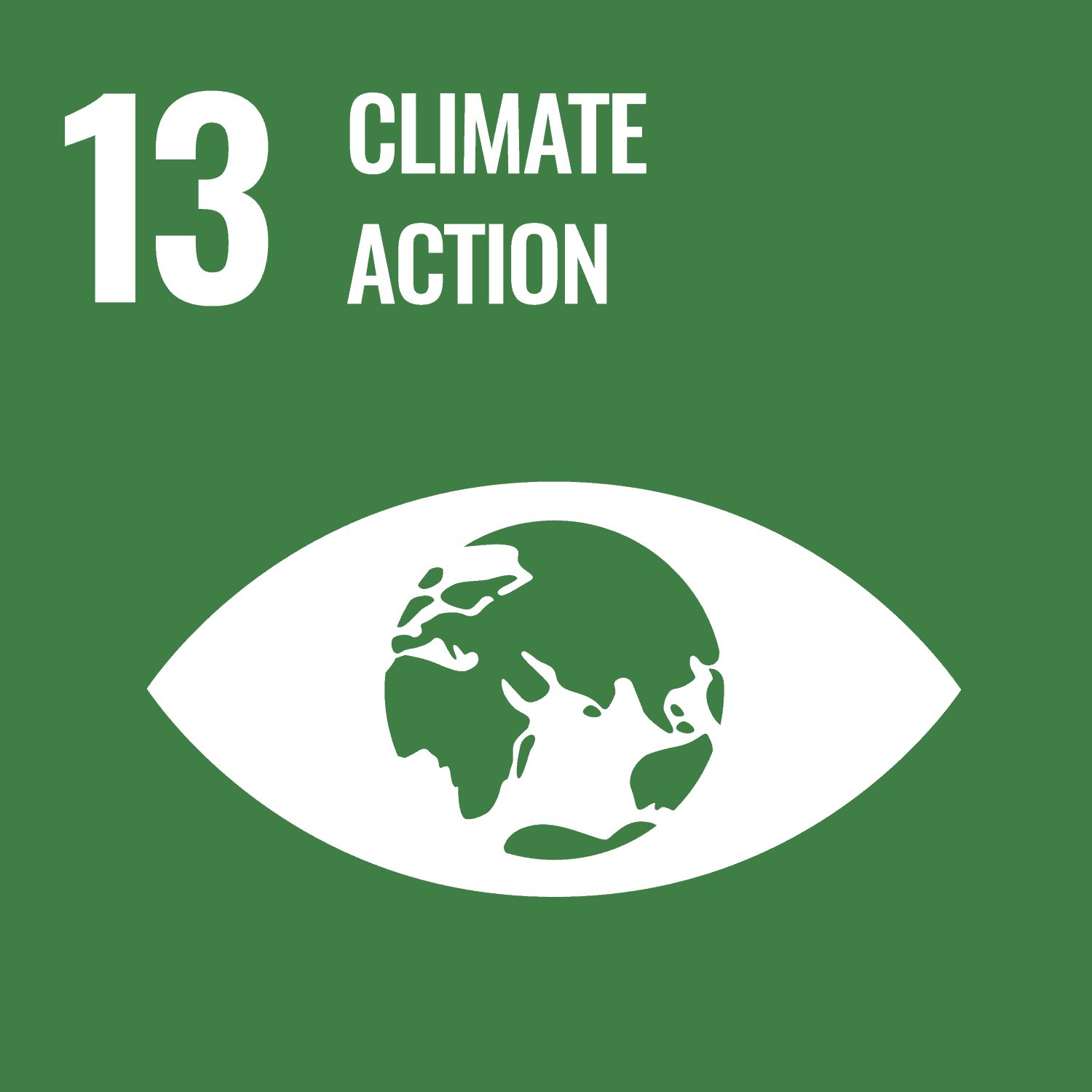
By transforming a passive monitoring process into an intelligent, active control loop, this project directly contributes to the following **UN Sustainable Development Goals (SDGs):**



**SDG 6 (Clean Water and Sanitation):** By preventing leaks, reducing waste, and ensuring efficient water management.



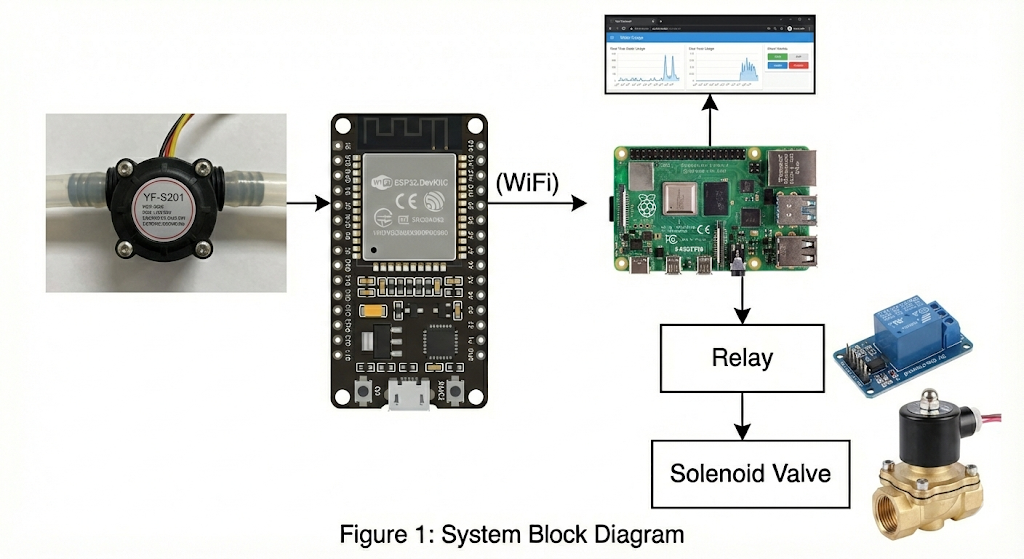
**SDG 12 (Responsible Consumption and Production):** Encouraging conscious water usage habits through data visualization and real-time feedback.



**SDG 13 (Climate Action):** Reducing the energy footprint associated with excessive water processing and transport by minimizing loss.

1. **System Model**

The system is divided into three main layers: The Perception Layer (Sensors), the Network/Control Layer (Microcontrollers), and the Application Layer (Web Interface & AI).

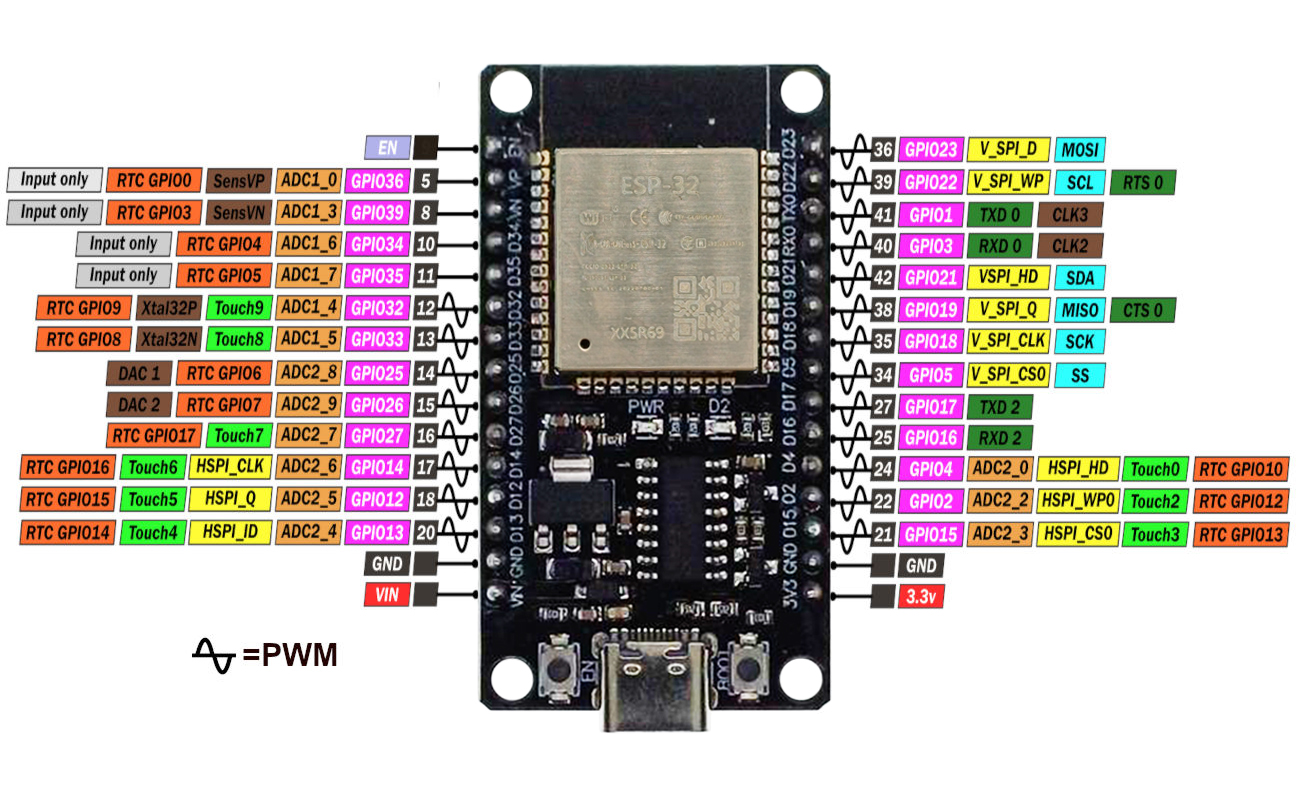


* 1. **Hardware Design**

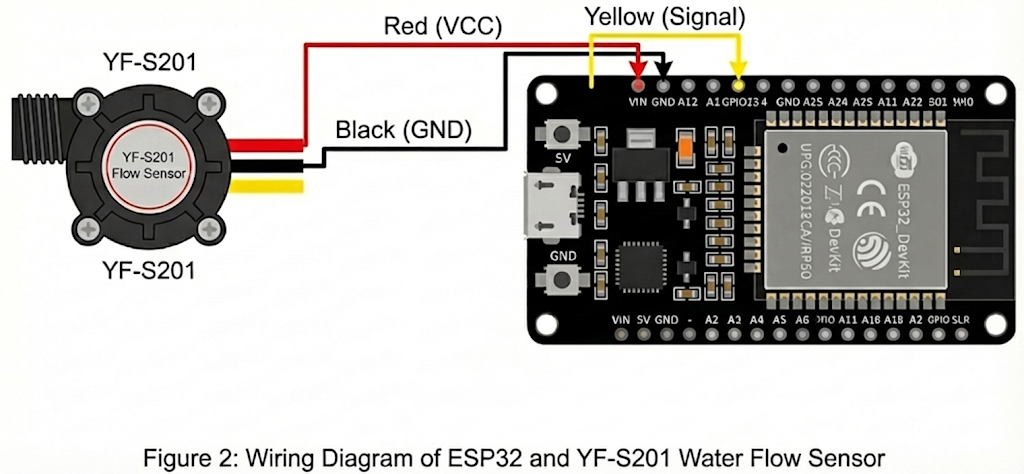
The hardware setup is designed to be modular. We separated the sensing unit from the actuation unit to demonstrate a distributed IoT network.

* + 1. **Hardware Selection Methodology:** In designing the *Intelligent Water Consumption Control* system, specific hardware components were selected to balance performance, cost, and architectural requirements. The rationale behind these choices is detailed below:
* **Why Raspberry Pi 4 instead of a single ESP32? (The Edge Gateway Rationale)** While an ESP32 is capable of simple processing, it lacks the computational power and memory required to host a full-stack solution locally. We chose the Raspberry Pi 4 to act as a localized server because:
  1. **Edge Computing Capability:** It enables running the TensorFlow Lite model and the MQTT Broker locally. An ESP32 would struggle to run a complex LSTM model while simultaneously managing web requests and database operations.
  2. **Data Sovereignty:** By hosting the database and dashboard on the Pi, we eliminate the need for third-party cloud services, ensuring data privacy and reducing latency to <200ms.
  3. **Multitasking OS:** The Linux environment allows for concurrent execution of the Python Backend, Web Server, and Database, which a microcontroller like the ESP32 cannot efficiently handle.
* **Why ESP32 as the Sensing Node? (Distributed Architecture)** Instead of connecting sensors directly to the Raspberry Pi, we utilized the ESP32 as a dedicated sensing node. This decision was driven by:
  1. **Real-Time Precision:** The ESP32 is a microcontroller running on bare-metal (or RTOS), which provides precise hardware interrupts required for accurate pulse counting from the flow sensor. A Raspberry Pi running a general-purpose OS (Linux) may miss high-frequency pulses due to process scheduling.
  2. **Wireless Mobility:** Placing the Raspberry Pi near water pipes is often impractical due to power and space constraints. The ESP32’s built-in Wi-Fi allows the sensing unit to be placed directly on the pipe, transmitting data wirelessly to the central server.
* **Why YF-S201 Flow Sensor?** The YF-S201 was selected for its cost-effectiveness and reliability for non-industrial applications. It operates on a simple Hall Effect principle, providing a digital square wave output that is easily readable by the ESP32 without complex analog-to-digital conversion. Its standard 1/2-inch thread makes it compatible with most household plumbing systems.
* **Why Relay Module & Independent Power Supply?** Directly connecting the 12V Solenoid Valve to the Raspberry Pi or ESP32 is impossible as their GPIOs operate at 3.3V/5V. We incorporated a Relay Module to provide:
  1. **Voltage Level Shifting:** To switch the 12V required by the valve using a 3.3V/5V control signal.
  2. **Galvanic Isolation:** To protect the sensitive microcontrollers from "Back-EMF" (voltage spikes) generated when the solenoid coil de-energizes. Furthermore, as noted in the *Challenges Faced* section, using a dedicated 12V power supply for the valve prevents voltage sag that could reset the logic boards.

**2.1.2. Sensing Unit (ESP32 & Flow Sensor)**

The ESP32 was selected as the sensing node due to its built-in Wi-Fi/Bluetooth capabilities, dual-core architecture, and power efficiency. It is responsible for reading raw data from the flow sensor and transmits it to the central server via the MQTT protocol. The ESP32 acts as a Publisher, sending sensor data to a specific topic.

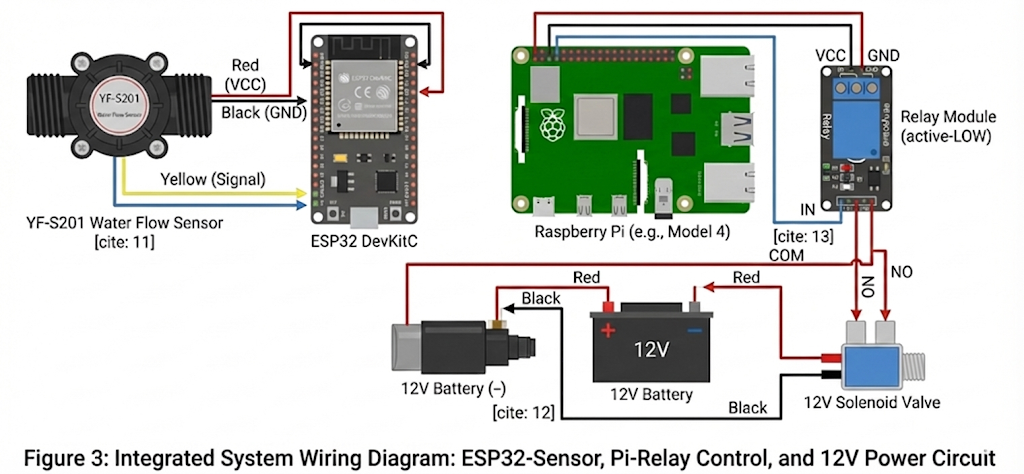
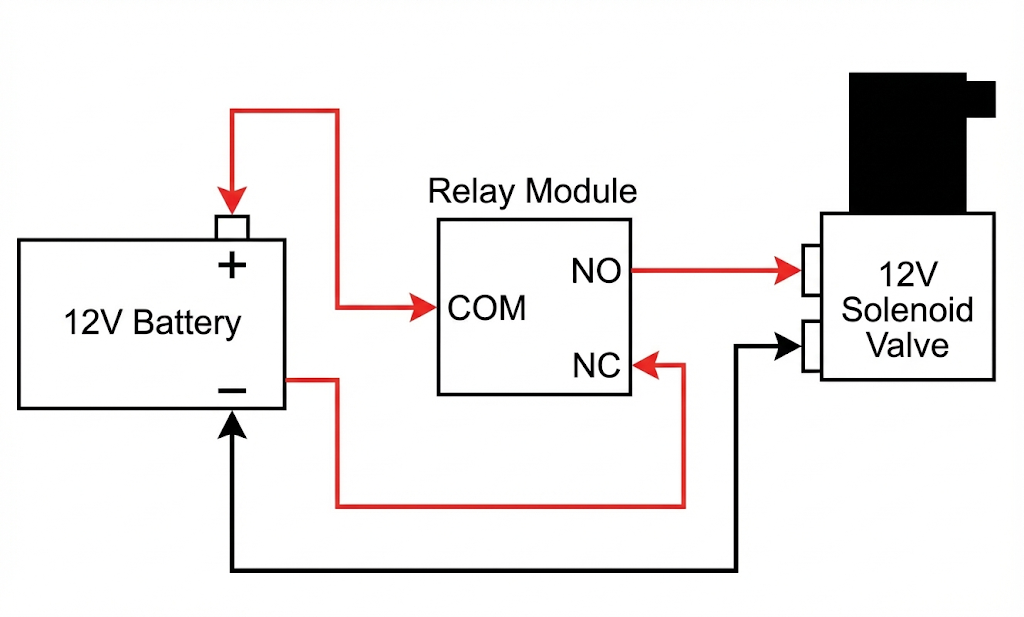
To measure water consumption, we integrated a **YF-S201 Water Flow Sensor**. This component utilizes a Hall Effect sensor to output a digital pulse for every revolution of the turbine driven by water flow.



**2.1.3. Actuation Unit (Raspberry Pi & Valve)**

The Raspberry Pi 4 functions as the edge gateway and the "Brain" of the system. It hosts the web server, runs the AI inference, and controls physical actuators. To manage the water supply, a 12V Solenoid Valve is used. Since the Raspberry Pi GPIO pins operate at 5V and cannot drive high-voltage components directly, a Relay Module is employed as an isolated switch.

**Design Consideration:** To prevent voltage drops and protect the Raspberry Pi from back-EMF, we isolated the power supply; the valve draws power from a dedicated 12V source, separate from the Pi's logic power.



**2.1.4. Embedded Code Logic**

The ESP32 firmware is developed using C++ within the Arduino Framework. Upon initialization, it establishes a Wi-Fi connection and enters a continuous loop to count sensor pulses. To balance real-time tracking with network efficiency, the system aggregates data and publishes a JSON payload to the MQTT Broker hosted on the Raspberry Pi every 2 seconds.

1. **Software Design**

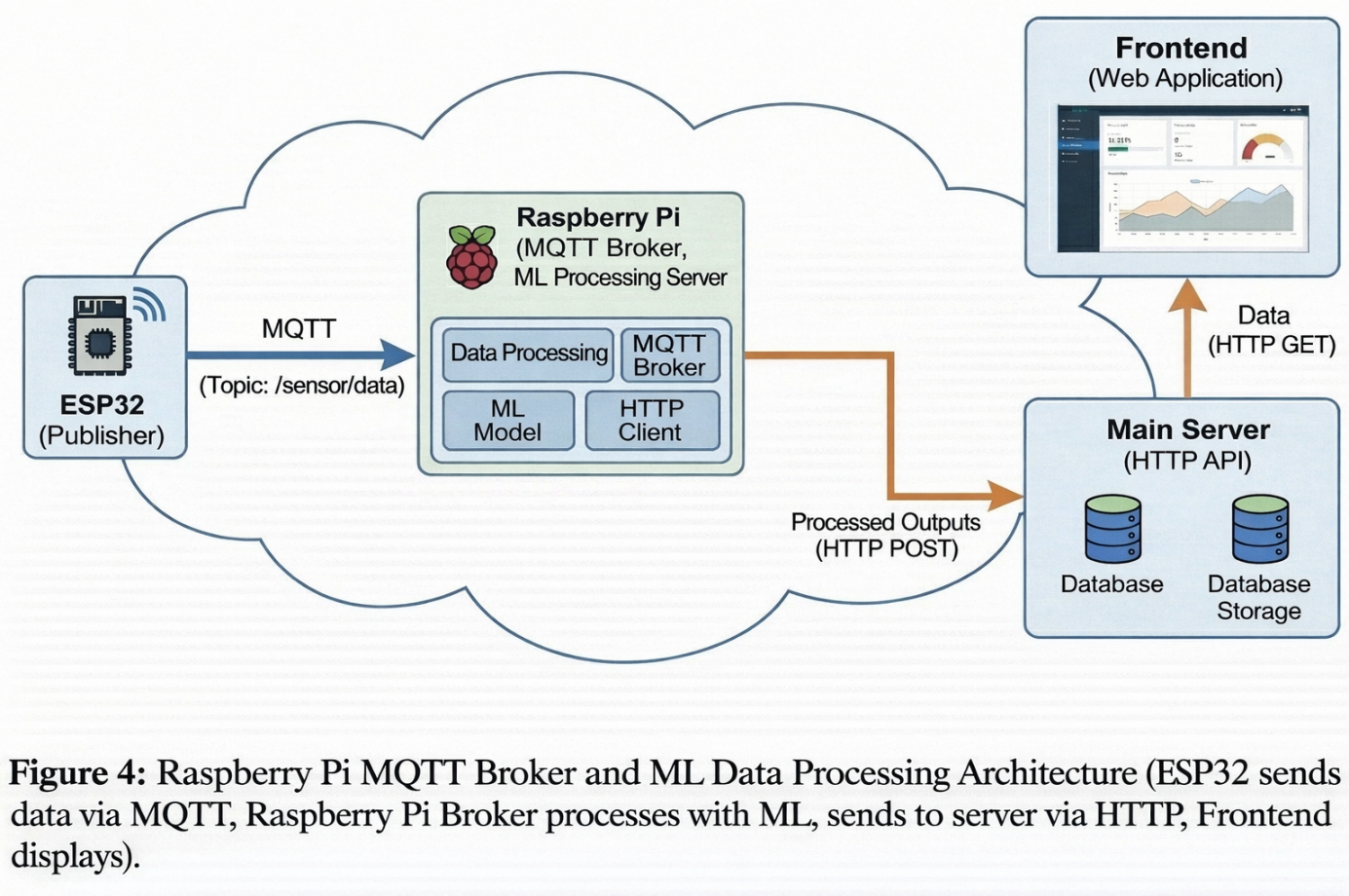
Unlike traditional cloud-dependent architectures, our project prioritizes data privacy, low latency, and edge computing capabilities. Therefore, we hosted the entire Backend and Database locally on the Raspberry Pi edge gateway.

* 1. **Hybrid Edge-Cloud Architecture**

Unlike traditional monolithic architectures, we adopted a Distributed Hybrid Architecture to optimize performance and scalability. In our initial design, the Raspberry Pi was intended to host all services. However, to ensure reliability and prevent resource bottlenecks during ML inference, we separated the responsibilities into two layers:

1. **Edge Gateway (Raspberry Pi 4):**
   * Acts as the local intelligent gateway.
   * Runs the **MQTT Broker** (Mosquitto) to collect real-time sensor data from ESP32 nodes with low latency.
   * Executes the **TinyML Model** (Scikit-learn) locally to detect anomalies immediately without network delay.
   * **Data Forwarding:** Instead of storing heavy data locally, it processes the raw streams and forwards valid data packets via **HTTP POST** requests to the Central Server.
2. **Central Server (Simulated Cloud Layer):**
   * Hosted on a high-performance machine (PC/Mac) within the local network (e.g., IP: 172.20.10.4) to simulate a production-grade Cloud Server.
   * **Backend:** Runs the Java Spring Boot application, which handles complex business logic and API management.
   * **Storage:** Hosts the MySQL Database via Docker, ensuring persistent and scalable storage for historical data which exceeds the SD card reliability of a standard Raspberry Pi.

**Justification:** This separation allows the Raspberry Pi to focus on real-time tasks (MQTT & ML) without being burdened by heavy database write operations, mirroring a real-world industrial IoT scenario where edge devices send data to a central cloud.

****

* 1. **Backend Web Application**

The web backend was implemented using Java (Spring Boot) to provide a stable and scalable REST-based service architecture. Maven was used for build automation and dependency management, ensuring consistent configuration across environments.

A MySQL relational database was integrated for persistent storage of water consumption records, system parameters, and historical usage data. The backend exposes RESTful APIs that enable communication between the frontend dashboard and the server for data retrieval and control operations.

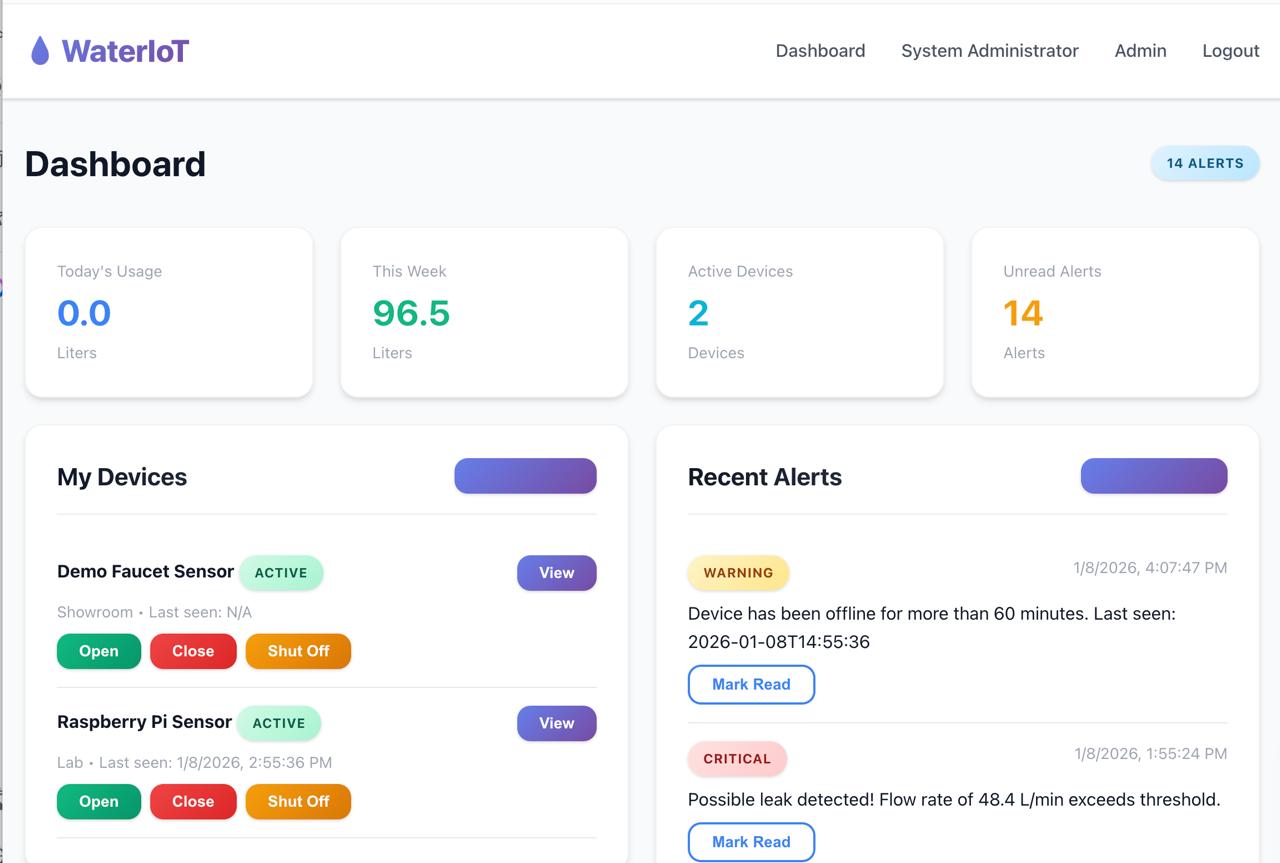
The application is deployed behind an Nginx web server, which operates as a reverse proxy to route HTTP requests to the Spring Boot service, improve performance, and provide a secure access layer.

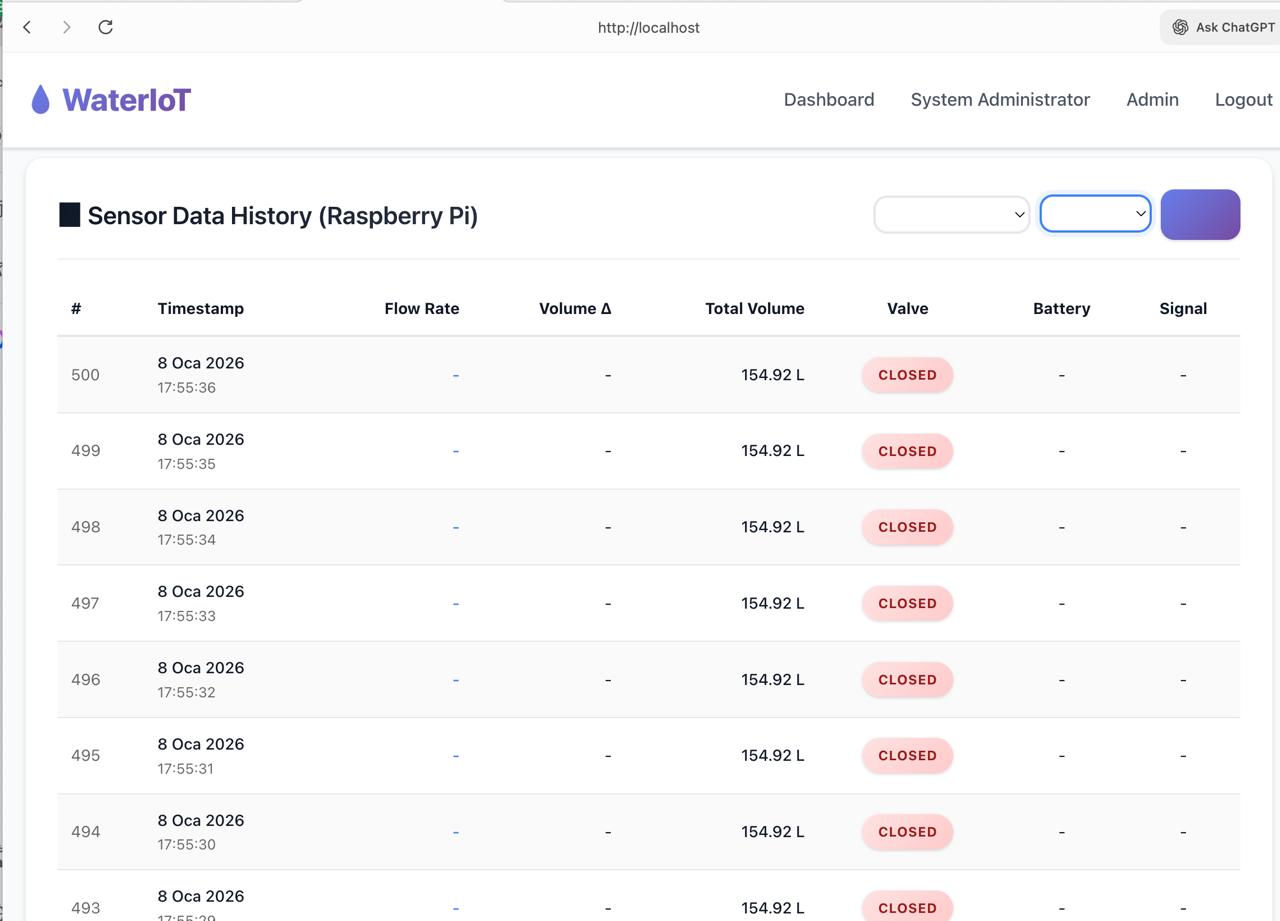
This backend layer is responsible for data management, API services, and coordination with the web interface, complementing the real-time IoT and MQTT-based processing layer.

**3.3. Frontend Web Application** To ensure cross-platform accessibility without the need for installation, we developed a Responsive Web Dashboard using HTML/JS. This allows users to monitor the system from any device (Smartphone, Tablet, PC) connected to the local network.

Key features of the dashboard include:

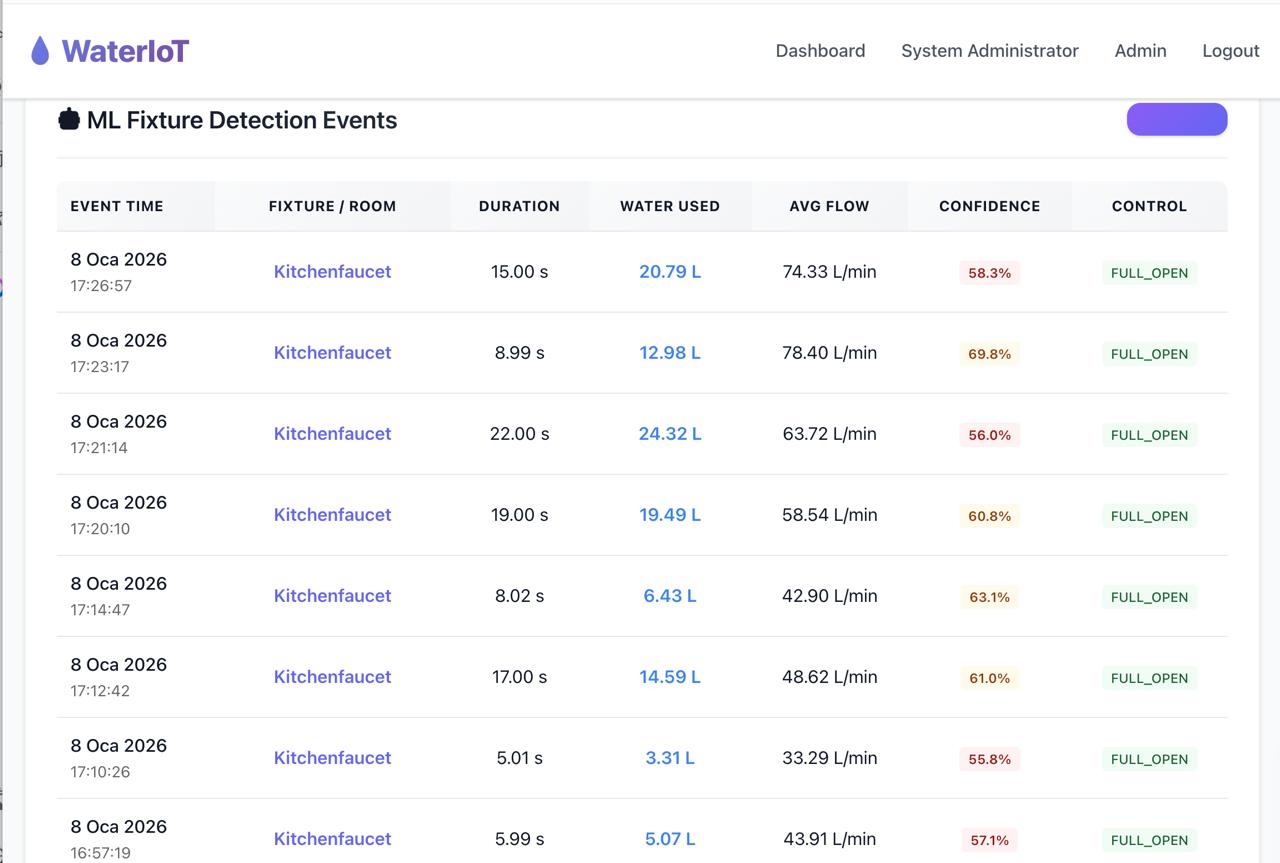
* **Real-time Gauge:** Visualizes the instantaneous water flow rate (Liters/min).
* **Historical Analytics:** Interactive charts displaying usage trends over time to identify wasteful habits.
* **Emergency Stop / Manual Control:** A dedicated switch that allows the user to immediately cut off the water supply in case of an emergency or maintenance.

****



**3.4. Machine Learning & TinyML Integration** To transform the system from a passive meter to an "Intelligent" guardian, we integrated a Machine Learning model using Scikit-learn. We employed a Random Forest Classifierapproach to learn household consumption patterns.

* **Training Process:** The model was trained on a dataset of "normal" usage to predict the expected flow rate for any given hour of the day.
* **TinyML Implementation:** Running a full neural network on an edge device can be resource-intensive. To address this, we converted our model to the **TensorFlow Lite (TFLite)** format. This optimization enables the Raspberry Pi to perform local inference with minimal latency and without relying on heavy cloud computing resources or internet bandwidth.

****

**Anomaly Detection Logic:** The system compares real-time sensor data against the AI's predicted range. If a significant deviation occurs (e.g., high water flow at 3:00 AM when the predicted usage is zero), the system flags this as a potential "Leak" or "Pipe Burst" and triggers the solenoid valve to close automatically.

**3.5. System Resilience and Fault Tolerance**

In real-world IoT deployments, network instability is inevitable. To address this, we implemented specific software logic to handle connectivity failures without causing system crashes ("Graceful Degradation").

* **MQTT Auto-Reconnection (ESP32):** The ESP32 firmware includes a "Keep-Alive" mechanism. If the Wi-Fi signal is lost or the MQTT Broker (Raspberry Pi) becomes unreachable, the sensing node enters a non-blocking reconnection loop. Instead of freezing, it periodically attempts to reconnect to the broker while maintaining its internal state, ensuring that the device comes back online automatically once the network is restored.
* **HTTP Error Handling & Retry Logic (Edge Gateway):** Communication between the Raspberry Pi (Edge) and the Main Server (Spring Boot) is critical. During testing, we simulated server downtimes (resulting in Connection Refused errors). To prevent the Python Edge script from terminating unexpectedly, we implemented Exception Handling (Try-Catch blocks) within the HTTP client.
  + **Logic:** If the Main Server is offline (e.g., during maintenance or network failure), the Raspberry Pi logs the error but continues to perform its core duties—collecting MQTT data and running the ML model locally.
  + **Resilience:** This ensures that the local leak detection system remains active even if the central dashboard connectivity is temporarily lost.

1. **Results and Discussion**

The implementation of the Intelligent Water Consumption Control system yielded significant results in terms of system responsiveness, prediction accuracy, and operational stability.

**4.1. Performance Metrics**

* **System Latency:** The end-to-end communication delay between the ESP32 sensing node and the Web Dashboard was optimized to approximately **<200ms**. By optimizing the REST API payload size, we ensured that users receive "live" feedback with minimal lag.
* **AI Model Accuracy:** The TinyML model (trained with Scikit-learn) achieved an **F1-Score of 0.82** on the test dataset. In practical scenarios, the system successfully detected simulated leaks (continuous flow anomalies) within **10 seconds**, triggering the alert mechanism significantly faster than manual detection.
* **Control Efficacy:** The actuation unit demonstrated high reliability. The solenoid valve responded to both manual web commands and AI-triggered emergency stops instantly, validating the effectiveness of the relay isolation circuit.

**4.2. Challenges Faced & Solutions**

During the development phase, we encountered and resolved several engineering challenges:

* **Sensor Noise due to Turbulence:** The YF-S201 flow sensor initially produced erratic signals caused by water turbulence in the pipes. To address this, we implemented a **software-based debounce filter** (and moving average) in the ESP32 firmware, which stabilized the readings and prevented false positives.
* **Power Supply Instability:** We observed that operating the solenoid valve (high current) and the Raspberry Pi (sensitive logic) from the same power source caused significant voltage drops, leading to system reboots. This was resolved by **isolating the power supply**; using a dedicated 12V source for the valve and a separate supply for the logic unit.

**4.3. Future Scope**

To further enhance the scalability and sustainability of the system, future iterations will focus on:

* **LoRaWAN Integration:** Replacing Wi-Fi with LoRaWAN technology to extend the communication range for large-scale agricultural or industrial applications.
* **Energy Harvesting:** Implementing hydro-generators within the pipe system to harvest energy from the water flow, potentially making the sensing nodes fully autonomous and battery-free.
* **Dedicated Mobile App:** While the web dashboard is effective, a native mobile application could be developed to provide push notifications for immediate leak alerts.

**4.4. Real-World Scenario**

To illustrate the system's tangible impact on daily life, a common household accident scenario is considered. Assume a family has left their home for a weekend vacation. While the house is unoccupied, a sudden rupture occurs in the flexible connection hose beneath the kitchen sink.

**In a traditional system:** Water continues to flow uncontrollably for hours, or even days. This results in the wastage of tons of water and causes significant financial damage by ruining wooden flooring and leaking into the apartment below. The user only notices the situation upon returning home or receiving a complaint from neighbors.

**In our developed "Intelligent Water Consumption Control" system:**

* **Detection:** The AI model on the Raspberry Pi detects continuous, high-volume water flow (an anomaly) at a time when the house should be empty (e.g., at 03:00 AM or when "vacation mode" is active). Using learned data patterns, the model distinguishes this flow from normal activities such as hand washing or appliance usage.
* **Intervention:** The system flags the anomaly as a "Critical Leak" and physically shuts off the main water supply by triggering the solenoid valve within 10 seconds.
* **Notification:** An instant notification stating "Leak Detected - Water Cut Off" is sent to the user's phone or Web Dashboard.

In this scenario, the system evolves from a passive meter into an active security unit that protects the home from flooding.

1. **Conclusion**

This project successfully implements an "Intelligent Water Conservation System" by synergizing embedded sensing (ESP32), edge computing (Raspberry Pi), and web technologies. By shifting from a traditional passive metering approach to an active, AI driven control model, the system not only monitors consumption but actively prevents waste.

The integration of Scikit-learn allowed for efficient anomaly detection at the edge, ensuring data privacy and reducing reliance on cloud connectivity. Ultimately, this solution offers a scalable and cost-effective blueprint for addressing the global water scarcity crisis through technology.

1. **Reference List**

**[1]** United Nations, "Transforming our world: the 2030 Agenda for Sustainable Development," General Assembly Resolution A/RES/70/1, Oct. 2015.

**[2]** World Health Organization (WHO) and UNICEF, "Progress on household drinking water, sanitation and hygiene 2000-2022: special focus on gender," New York, 2023.

**[3]** UN-Water, "The United Nations World Water Development Report 2024: Water for Prosperity and Peace," UNESCO, Paris, 2024.

**[4]** A. Banks and R. Gupta, "MQTT Version 3.1.1," OASIS Standard, Oct. 2014. [Online]. Available: <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html>

**[5]** W. Yu et al., "A Survey on the Edge Computing for the Internet of Things," *IEEE Access*, vol. 6, pp. 6900-6919, 2018.

**[6]** F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

**[7]** P. Warden and D. Situnayake, *TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers*. O'Reilly Media, 2019.

**[8]** J. Walls et al., "Spring Boot in Action," Manning Publications, 2016.

**Project Repository Link:** https://github.com/Vedat-ayaz/IOT\_Project.git

**Demo Video:** https://drive.google.com/drive/folders/1-m84p4RzNOEDAufKRSz\_tOCywQ8khbOv?usp=sharing