Decentralized IoT Solution for Smart Agriculture Using Edge Computing

Report submitted to SASTRA Deemed to be University
As per the requirement for the course

CSE400: PROJECT & VIVA VOCE

Submitted by

KISHORE B
(225003069, B. Tech Computer Science and Engineering)

MAY 2025

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SRINIVASA RAMANUJAN CENTRE
KUMBAKONAM, TAMILNADU, INDIA – 612001

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SRINIVASA RAMANUJAN CENTRE
KUMBAKONAM, TAMILNADU, INDIA – 612001

Bonafide Certificate

This is to certify that the report titled **"Decentralized IoT Solution for Smart Agriculture Using Edge Computing"** submitted as a requirement for the course, CSE400:Project & Viva Voce for B.Tech. is a bonafide record of the work done by Kishore B (225003069, B. Tech Computer Science and Engineering) during the academic year 2024-25, in the School of Computing, under my supervision.

Signature of Project Supervisor :
Name with Affiliation : Mr. Jeyapandian M - Asst. Professor-II
Date :

Mini Project *Viva voice* held on _____

Examiner 1 Examiner 2

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SRINIVASA RAMANUJAN CENTRE
KUMBAKONAM, TAMILNADU, INDIA – 612001

Declaration

I declare that report that the project report titled "**Decentralized IoT Solution for Smart Agriculture Using Edge Computing**" submitted by me is an original work done by me under the guidance of **Mr. Jeyapandian M** - Asst. Professor-II, School of Computing, SASTRA Deemed to be University during the final semester of the academic year 2024-25, in the School of Computing. The work is original and wherever I have used material from other sources, I have given due credit and cited them in the text of the report. This report has not formed the basis for the award of any degree, diploma, associateship, fellow or other similar title to any candidate of any University.

Signature of the candidate **:**

Name of the candidate **:** Kishore B

Date **:**

Acknowledgments

I would like to thank our Honorable Chancellor **Prof. R. Sethuraman** for providing us with an opportunity and the necessary infrastructure for carrying out this project as a part of our curriculum.

I would like to thank our Honorable Vice Chancellor **Dr. S. Vaidhyasubramanian** and **Dr. S. Swaminathan**, Dean, Planning & Development, for the encouragement and strategic support at every step of our college life.

I extend our sincere thanks to **Dr. R. Chandramouli**, Registrar, SASTRA Deemed to be University for providing the opportunity to pursue this project.

I extend our heartfelt thanks to **Dr. V. Ramaswamy**, Dean, Srinivasa Ramanujan Centre, and **Dr. A. Allirani**, Associate Dean, Srinivasa Ramanujan Centre.

I exhibit our pleasure in expressing our thanks to **Dr. V. Kalaichelvi**, Associate Professor, Department of Computer Science and Engineering, Srinivasa Ramanujan Centre, for her encouragement during our project work.

My guide **Mr. Jeyapandian M**, Assistant Professor-II, Department of Computer Science and Engineering, Srinivasa Ramanujan Centre was the driving force behind this whole idea from the start. His deep insight into the field and invaluable suggestions helped us in making progress throughout our project work. I also thank the project review panel members **Dr. Arun Kumar S,** Assistant Professor-II and **Dr. Balakrishnan R,** Assistant Professor-II, for their valuable comments and insights which made this project better.

I would like to extend our gratitude to **Dr. S. Ranjeeth Kumar**, Assistant Professor & Project Co-coordinator, Department of Computer Science and Engineering, Srinivasa Ramanujan Centre for organizing and helping us in the completion of the project.

I gratefully acknowledge all the contributions and encouragement from my family and friends resulting in the successful completion of this project. We thank you all for providing me with an opportunity to showcase my skills through this project.

Table of Contents

List of Figures

ABSTRACT

Existing smart irrigation systems face limitations such as dependency on third-party cloud services, inefficient water usage, limited connectivity, and lack of data privacy. These challenges often hinder the effectiveness and scalability of traditional solutions. To address these issues, this project proposes a decentralized smart irrigation system tailored for agricultural fields. By integrating private web-based monitoring and automated irrigation control, the system eliminates reliance on external cloud platforms, ensuring seamless and uninterrupted operation. This approach envisions a future where farmers have full control over their irrigation infrastructure, enabling real-time decision-making and optimizing resource usage. With enhanced autonomy, secure data management, and decentralized operation, it paves the way for more resilient, scalable, and self sufficient agricultural practices, ultimately promoting sustainability and long-term efficiency in farming.

Keywords: Decentralized Smart Irrigation, Private Web-based Monitoring, Automated Irrigation Control, Data Privacy, Resource Optimization, Real-time Decision-making, Scalable Agricultural Solutions, Autonomous Farming Systems, Sustainable Agriculture, Self-sufficient Infrastructure, Resilient Farming Practices, Cloud-independent Operation

CHAPTER 1

Introduction

1.1 Summary of Base Paper

Journal Name: Results in Engineering

Title: IoT-based smart irrigation management system to enhance agricultural water security using embedded systems, telemetry data, and cloud computing [2024]

Author: Abdennabi Morchid, Hassan Qjidaa, Rachid Jebabra, Haris M. Khalid, Mohammed Ouazzani Jamil, Rachid El Alami

Base Paper Abstract:

Agriculture, the key sector for food production, faces challenges exacerbated by the growing global demand for food and the scarcity of water resources. Traditional irrigation methods often lead to inefficient use of water, resulting in wastage. Moreover, unpredictable weather conditions and the need to adopt sustainable farming practices are driving us to develop advanced irrigation systems. This paper proposed a smart irrigation management system using new technology like embedded systems, Internet of Things (IoT), telemetry data, cloud computing, communication protocol, and sensors to collect and process real-time data of the smart agriculture. This new technology and intelligent algorithm are used in this paper to improve agricultural practices. The architecture of the proposed scheme comprises three layers: IoT devices, ThingsBoard cloud, and the dashboard, each playing a pivotal role in ensuring seamless data flow, secure communication, and enhanced user interaction. The algorithm orchestrates system operations, incorporating sensor data reading, JavaScript Object Syntax (JSON) payload creation, and secure telemetry data transmission via Hypertext Transfer Protocol (HTTP) protocol to the ThingsBoard cloud. Pump activation decisions are governed by pre-defined thresholds, preventing an over-irrigation system. The evaluation of system performance involves deploying temperature, humidity, moisture, and water level sensors in a testing field. Further, before data is sent to the cloud, sensor values are calibrated using a functional map. Tests carried out with temperature, humidity, and

water level sensors demonstrate the system's dynamic efficiency. By visualizing and analyzing environ mental information via ThingsBoard, the results provide real-time data on environmental parameters of the system proposed, improving the efficiency of water use and the sustainability of farming practices. In addition, the system features e-mail notifications for alerts. The integration of e-mail notifications reinforces monitoring and management practices by alerting farm owners and users. This study showcases the efficacy of the proposed paper in enhancing sustainability, water usage, and supporting smart agriculture practices. Further, this paper integrates embedded systems, IoT, cloud computing, and advanced algorithms to optimize and enhance crop productivity and contribute to global food and water security.

1.2 Background

Agriculture has remained the backbone of human civilization, providing sustenance, raw materials, and food to billions. Over time, traditional methods of farming have been revolutionized with the aid of mechanization, biotechnology, and information technologies. Yet, modern agriculture still suffers from many serious problems such as wasteful use of resources, water scarcity, unpredictable yields, and growing calls for sustainability.

With the emergence of the Internet of Things (IoT), it is now feasible to monitor and regulate farm operations in real-time. IoT-based systems integrate sensors, actuators, and data communication technologies to provide precise information about soil, water, and crop conditions. Although traditional smart farming models employ cloud platforms for processing and storage, these also present data privacy, high-cost, and third-party service dependency concerns [1][2]. This calls for the shift to decentralized and edge-based smart agriculture solutions to enable real-time, local decision-making and increased autonomy for farmers [6][13].

1.3 Significance of Smart Agriculture:

Smart agriculture utilizes digital technologies—like IoT, automation, and machine learning—to transform crop growth and maintenance. The transition is vital in solving the world's population growth, food security, and climate change challenges.

1.3.1 Key benefits of smart agriculture include:

- Precision Farming: Proper application of water, fertilizers, and pesticides according to sensor data.
- Real-Time Monitoring: Real-time monitoring of crop status, soil moisture, and climatic conditions.

- Automation: Labor is decreased through mechanized fertilization, crop management, and irrigation systems.
- Data-Driven Decision Making: Farmers can make data-driven decisions that optimize yield and reduce resource utilization.

With the shift towards decentralized architecture on the premise of edge computing, smart farming systems are being rendered more secure, cost-effective, and privacy-oriented so that farmers are granted complete control over processes and information [6][13][14].

1.4 Motivation Behind the Project

The growing dependence on third-party cloud platforms in smart agriculture is of great concern regarding data privacy, operational reliance, and periodic expenses [2][6]. Farmers nowadays have little or no control over their own data, as it is stored and processed in centralized servers that are susceptible to breaches, abuse, or service outages [3][7]. Moreover, small farmers in rural regions can experience unstable internet connectivity, rendering cloud-based systems less feasible.

These issues prompted the creation of a decentralized edge-computing-based solution that allows farmers to control irrigation infrastructure directly without depending on constant access to the cloud. Utilizing local devices such as Raspberry Pi and ESP32 nodes to handle data collection and processing, the system provides real-time responsiveness along with enhanced reliability even in low-connectivity or offline scenarios [6][10]. This initiative seeks to provide digital sovereignty to farmers, securing sustainable and self-sufficient smart agriculture practices in the future[13][14].

1.5 Project Objectives

The main aims of the "Decentralized IoT Solution for Smart Agriculture Using Edge Computing" project are the following:

1.5.1 End Reliance on Centralized Cloud Infrastructure

Develop a system where every agricultural data is processed and stored locally at the edge, eliminating the dependency on centralized cloud providers. This design improves data sovereignty, ensures quicker response times, and provides higher resilience in rural or remote settings [6][13].

1.5.2 Support Real-Time Monitoring and Control

By connecting ESP32 nodes with a web dashboard, users can track soil moisture levels in real-time. This allows for real-time control actions such as the on/off switch of irrigation motors, thus enhancing responsiveness and minimizing labor [10].

1.5.3 Minimize Water Waste

The system utilizes smart irrigation logic that only starts water flow when moisture falls below a specified threshold. This helps avoid overwatering and ensures sustainable water use in agriculture [6].

1.5.4 Sustain Scalability and Flexibility

The architecture is scalable to multiple ESP32 nodes, making it simple to scale to larger farms or multiple fields. Each node operates independently, meaning that farmers can add more nodes without having to redesign the system [10][12].

1.5.5 Empower Data Ownership and Privacy

By processing data and hosting the dashboard locally, farmers have complete ownership of their data. This prevents third-party monitoring or abuse of farm data, which is in line with privacy-first philosophies [7][13].

1.5.6 Low-Cost and Open-Source Adoption

The project uses low-cost components such as Raspberry Pi and open-source platforms like Ngrok to maintain costs at a low level. This makes the solution more accessible across a broad spectrum, particularly to smallholder farmers and schools [6][11].

CHAPTER 2

Problem Statement

2.1 Challenges in Traditional Systems

Conventional smart irrigation systems tend to have a centralized architecture, whereby field sensor data is sent to cloud servers for analysis and decision-making. Although this architecture supports global access and integration with high-capacity analytics platforms, it has some operational and practical issues that pose a challenge to farmers, particularly rural or resource-limited farmers [2][6].

First, the centralized systems need constant internet connectivity. Network infrastructure in most agricultural areas is weak or non-existent, and therefore it becomes hard to ensure real-time data transfer to cloud platforms [10]. Consequently, such systems can be affected by latency or complete failure during network downtimes, which is essential when timely irrigation is required.

Another problem is inflexibility. Traditional systems are mostly bound to proprietary devices or platforms, which can make it difficult for farmers to adapt the installation or include local technologies. They are usually restricted to utilizing certain apps or cloud services that may not exactly address their requirements [7].

Scalability is also an issue. As the farmland gets larger, or as the range of crops expands, it becomes more complicated and costly to add more devices and sensors. Small and medium-sized farmers cannot easily afford to extend such systems [13].

These issues exhibit the shortcomings of centralized irrigation measures. A decentralized approach conducive to local leadership, functional with or without an internet connection, and affordable fits better for mainstream adoption [6][10][13].

2.2 Data Privacy and Security Concerns

In intelligent agriculture, information like soil moisture, irrigation rates, and crop health is very

valuable. But most current systems upload this information to big tech company-controlled centralized cloud servers [3][7].

This has implications for data ownership. Farmers tend to have little insight into how their data is used, processed, or distributed. In most instances, cloud providers will retain the right to analyze or even sell user data to other parties. This decreases the farmer's authority and enhances the risk of abuse [7][14].

Centralized servers also represent a point of failure. A system failure or security breach at the provider's end can compromise all user data or stop operations. For instance, unauthorized access to irrigation schedules by an attacker can trigger or delay watering intentionally, impacting yields [3].

With edge computing, farmers can store and compute data locally on their own systems. This implies that only the farmer possesses access to the data, which greatly enhances control, privacy, and security [13][14].

The project emphasizes giving farmers full control back. It tries to secure sensitive information while ensuring systems remain in full operation even in the absence of internet or cloud connectivity [6][13].

2.3 Dependency on Third-party Cloud Services

Most of today's smart irrigation systems rely heavily on third-party cloud services such as AWS, Google Cloud, or Microsoft Azure. These systems provide data storage, processing, and system control, but this reliance is not without a price [6][8].

Subscription fees, data caps, and service charges can accumulate over time. This poses a long-term financial strain on farmers, particularly small-scale farmers. What begins as a free or low-cost setup would ultimately become an expensive monthly or annual fee [13].

There is also the possibility of disruptions in services. If the cloud provider experiences downtime or alters its policies, farmers can lose access to their systems overnight. At times, an alteration in the terms of service by the provider can compel users to move their data, something that can be costly and challenging [11].

In tightly regulated data regions, the storage of agricultural data on overseas cloud servers can also

incur compliance. Farmers might unintentionally breach regional data laws, exposing them to legal trouble [8][14].

This project puts forward a decentralized system which cuts out third-party cloud service necessities. Every decision and data storage is local, and this puts the farmers entirely in charge of their infrastructure. This model enhances autonomy, cuts down long-term expenses, and eliminates cloud service dependency risk [10][13][14].

CHAPTER 3

System Design & Architecture

3.1 Overall System Overview

The decentralized smart irrigation system seeks to transform the way irrigation is controlled in farm fields through offering a scalable and highly efficient solution that is independent of outside cloud platforms. The system, at its foundation, is built to minimize water consumption while achieving maximum autonomy and data protection for farmers. Through removing dependency on third-party cloud services, the system guarantees consistent performance and complete control over the irrigation process.

The architecture of the system includes a centralized control unit, which is a Raspberry Pi, connected to various sensor nodes scattered over the farm. The nodes have soil moisture sensors and are tasked with gathering real-time information about the moisture content of the soil. The ESP32 microcontroller-powered sensor nodes send their information wirelessly to the central Raspberry Pi, which processes the information and makes decisions regarding irrigation using set moisture levels.

Once the moisture level goes below a specified limit, the Raspberry Pi provides a signal to the concerned sensor node to trigger the sprinkler system connected to it. With this automated mechanism, water is conserved without any wastage while ensuring perfect soil conditions for plant growth.

The Raspberry Pi also runs a web server exposed to the internet via NGROK so that farmers can view real-time moisture levels and remotely control the irrigation system from anywhere. Hosting the system on the internet, the farmers have the advantage of the flexibility to monitor and manage the irrigation process without depending on a centralized cloud service, which also improves data privacy and security.

Farmers can make informed, real-time choices regarding their irrigation methods with this system, lowering labour and water consumption while elevating the general sustainability and efficiency of farming activities. The edge computing integration helps ensure that the system is working effectively without delays from the latency of cloud-based solutions.

3.2 Hardware Components Used

This distributed smart irrigation system is constructed with a well-chosen set of hardware components, all of which are suitable for field-level agricultural conditions. The main objective was to design a robust, modular, and scalable infrastructure that would be able to operate effectively without depending on any third-party cloud platforms.

The initial major component is the soil moisture sensor, and more precisely, the capacitive type. In contrast to resistive sensors, capacitive moisture sensors are not subject to corrosion, thus being much more durable and reliable for extended periods of use outdoors. These sensors constantly measure the water content in the soil and offer real-time information that is the foundation of the irrigation decision-making process.

Then there's the ESP32 microcontroller that is employed as a sensor node at various areas of the field. Every ESP32 takes measurements of moisture levels from its connected sensor and has wireless communication with the master controller. The ESP32 was chosen because of its great inbuilt Wi-Fi feature, dual-core operation, and established efficiency in IoT-based applications. It is space-efficient, cost-effective, and extremely reliable to operate continuously for such projects.

3.3 Software Tools and Technologies

To bring the decentralized irrigation system to life, a variety of software tools and technologies were integrated, each chosen for its simplicity, reliability, and compatibility with edge devices.

Arduino IDE: The Arduino Integrated Development Environment (IDE) was used to program the ESP32 microcontrollers. Custom firmware was written to read data from the soil moisture sensors, format the readings, and transmit them wirelessly to the Raspberry Pi. The ESP32 boards operate on a scheduled loop, sending updated sensor values every few seconds via WiFi to ensure the system stays responsive to changing soil conditions [3].

Python on Raspberry Pi: A Python-based server script runs on the Raspberry Pi to handle incoming data from all the ESP32 nodes. This script listens for incoming JSON payloads, processes the soil moisture readings, logs the data into a CSV file, and makes decisions on whether to activate or deactivate the irrigation system. GPIO pins on the Raspberry Pi are used to send control signals to the appropriate ESP32 nodes when needed.

Flask (Python Web Framework): Flask, a lightweight Python web framework, was used to build the user-facing dashboard. It enables real-time display of sensor data, status indicators for each zone, and manual override options in case the farmer wants to control the irrigation manually. The interface is minimal and fast, designed to work well even on slower connections, while providing essential control and feedback features [11].

NGROK: To make the local Flask web server accessible over the internet, NGROK was used to create a secure tunnel. This allows farmers to access their field's live dashboard from any location, without the need for a public cloud platform or complex network setup. NGROK simplifies the process of remote access while maintaining end-to-end encryption [14].

CSV for Data Logging: Instead of using a traditional database, the system logs all sensor readings in a CSV file on the Raspberry Pi. This approach was chosen for its simplicity and ease of use. The CSV file stores timestamped records of each reading, which can be reviewed later for trend analysis or troubleshooting. It avoids the overhead of database setup while still keeping a reliable history of system activity.

JSON for Communication: Data exchanged between the ESP32 boards and the Raspberry Pi is formatted in JSON (JavaScript Object Notation). This lightweight format makes the communication efficient and easy to parse, ensuring structured and consistent messaging across the system.

The pairing of these technologies makes the system responsive, lightweight, and friendly while still compliant with the main philosophy of the project, which is decentralization and autonomy [13][14].

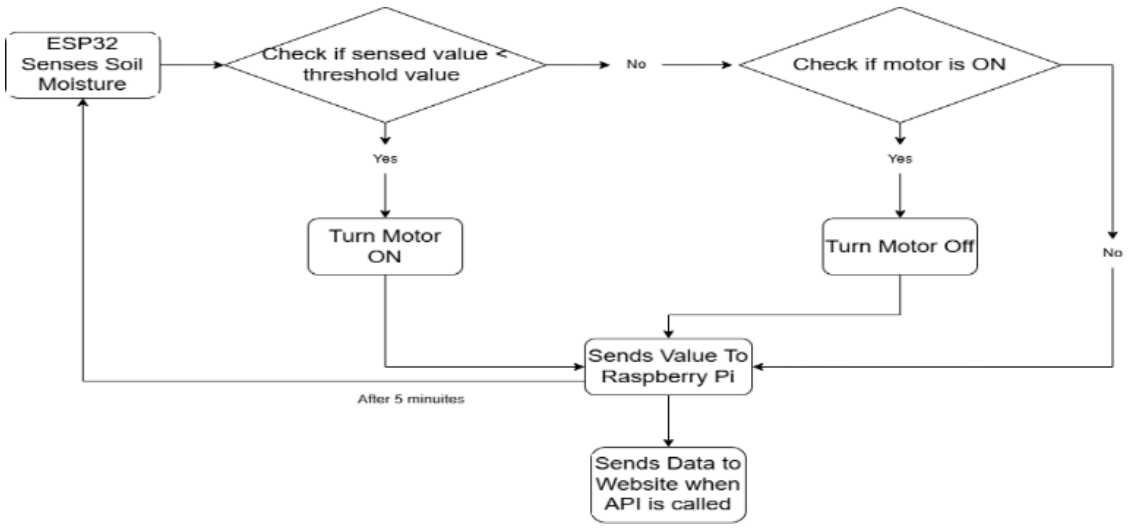

Figure 3.1: Block Diagram

3.4 Communication Flow

The flow of communications in this project is streamlined, modular, and decentralized to enable effective monitoring and control of farming irrigation. It starts from the physical sensor layer and continues right to a web interface accessible globally, with no dependence on conventional cloud platforms. The system is designed to gather, transmit, process, and visualize real-time information through a chain of modules that are interconnected and function in absolute synchrony.

The system starts with soil moisture sensors planted in the ground. The sensors are directly interfaced to the ESP32 microcontroller boards via GPIO pins. The sensors measure the volumetric water content of the soil, giving analog values that correspond to the present levels of moisture. ESP32 is the point of intelligence in the network that senses the analog signals from the sensors at specific intervals. These readings are digitized and processed locally to determine if they should be pushed to the central controller. Each ESP32 executes a light firmware that is programmed in Arduino IDE, which not only processes the sensor reading but also wraps the data into a JSON object that can be transmitted over the network.

When the ESP32 receives the sensor data, it sends this data wirelessly via WiFi to a Raspberry Pi, which acts as the master node and edge server of the system. The ESP32 devices and the Raspberry Pi are members of the same local WiFi network so that the communication is quick and efficient and does not require cellular or external internet connectivity. The information transmitted by the ESP32 boards is picked up by a Python script executed on the Raspberry Pi. The script serves as a data listener and processor, continuously checking for incoming messages. When it gets a new JSON payload from any of the ESP32 nodes, it processes the data, logs it into a CSV file for storage, and checks the moisture levels. Depending on predetermined thresholds, the Raspberry Pi may make GPIO output operations or send control signals to particular ESP32 modules in order to energize or disable the irrigation system such as solenoid valves or relays.

Simultaneously, the Raspberry Pi also contains a web-based interface, coded in Flask, which is an efficient Python web framework. Flask manages the display of real-time sensor readings, shows the current working status of every irrigation zone, and offers the user manual override functions via a graphical interface. The local server runs by default only within the local network, i.e., it can't be

accessed from the outside world unless a secure tunnel is established.

To bypass this limitation and enable the dashboard to be accessed by the landowner or farmer from any internet-enabled device, NGROK is incorporated into the system. NGROK establishes a secure HTTPS tunnel between the local Flask server and the public internet. It assigns a dynamic public URL, which serves as a gateway to the local dashboard. Thus, without the need to set up complicated domain hosting, port forwarding, or deployment to commercial cloud platforms, the system gives complete remote access to field data in real time. The farmer is able to view the dashboard, monitor moisture levels, check irrigation status, and even manually trigger irrigation using a mobile phone, tablet, or computer.

This sensor-to-internet communication architecture provides end-to-end decentralization and total data ownership by the farmer. It reduces latency, enhances reliability, and maintains data privacy through the elimination of third-party cloud storage or processing. The architecture is modular and scalable, making it possible to add more ESP32 nodes in the field with minimal changes in configuration. It is a sound application of edge computing concepts to agriculture, providing a sustainable and autonomous alternative to conventional smart farming platforms.

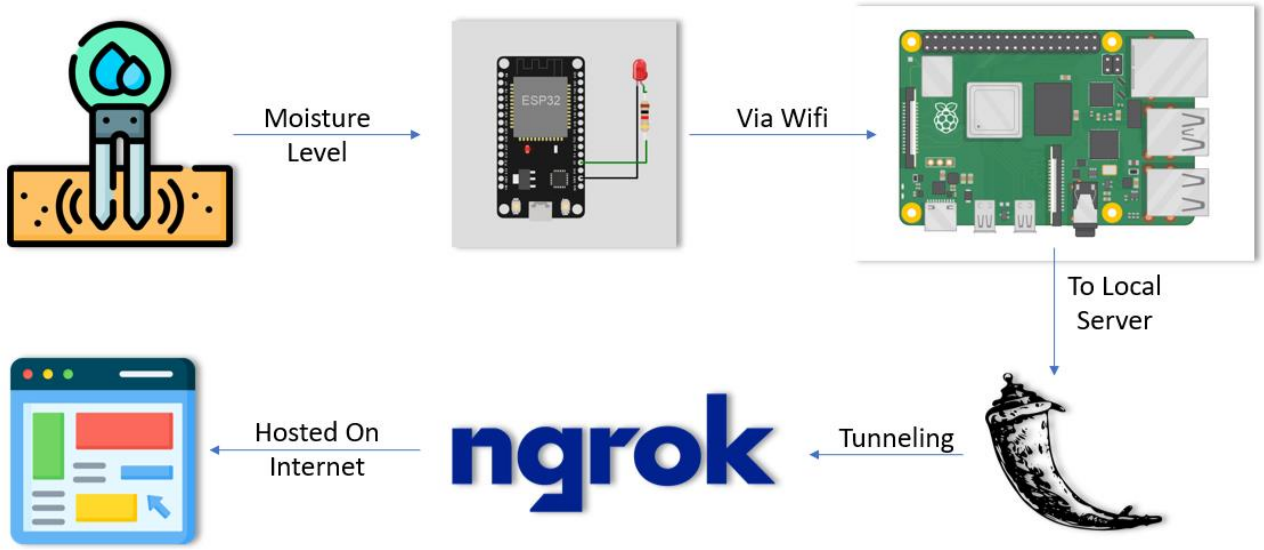

Figure 3.2: Communication Flow

3.5 Architecture Diagram

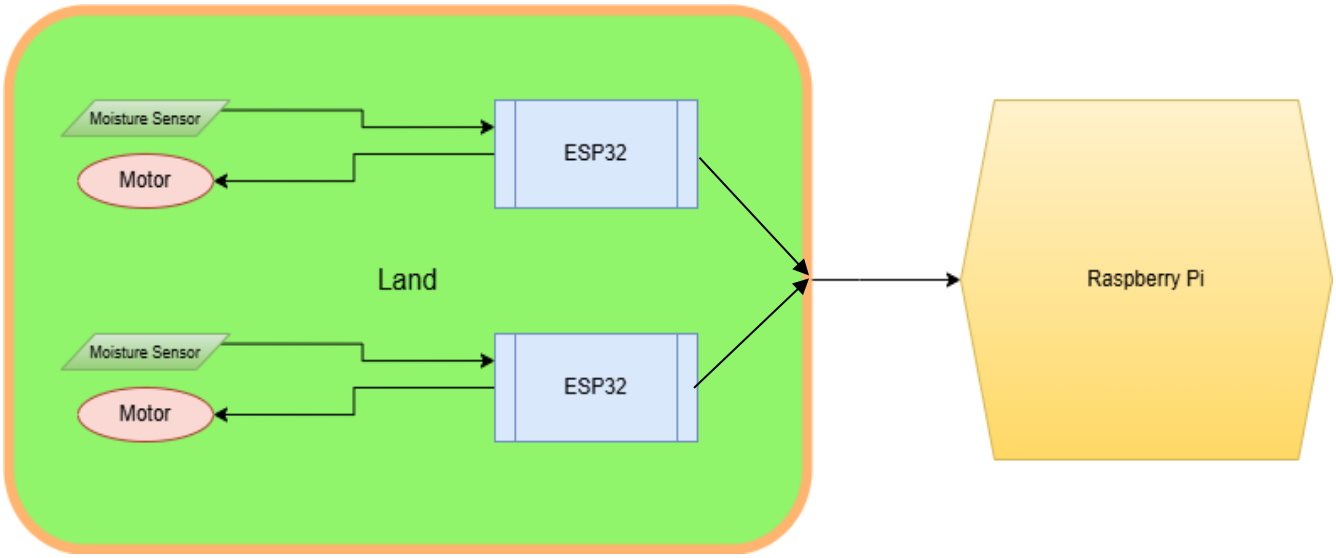

Figure 3.3: Architecture Diagram

CHAPTER 4

Methodology

4.1 Sensor Data Collection and Processing

The basis of the system being proposed is that it can collect soil moisture readings from multiple points throughout the farm and process the data in real-time. Every ESP32 microcontroller has a soil moisture sensor connected to it through GPIO pins. These sensors measure the resistance between two probes buried in the soil. The more dry the soil, the greater the resistance, and the wetter the soil, the lower the resistance. This analog signal is then digitized into a digital value by utilizing the onboard Analog-to-Digital Converter (ADC) of the ESP32.

Each couple of seconds, the ESP32 scans this moisture data and formats it in the right way to be communicated. To ensure consistency and simplicity for reading, the data is processed in a normalized percentage format (for example, 0% for fully dry and 100% for full saturated soil). The data is then packed into a light-weight form using JSON so that it is simple to send via wireless networks.

The ESP32 makes a WiFi connection and transmits the JSON-encoded moisture values to the Raspberry Pi, which serves as the central controller. The Raspberry Pi executes a Python server program that awaits incoming data packets, decodes them, and writes them into a CSV file. This CSV file serves as a lightweight local database, which provides an efficient and streamlined method of data logging without the use of third-party databases or cloud-based service providers.

This unbroken series and first-level processing layer is central to the overall system responsiveness and intelligence. Through the utilization of edge devices to process sensor interaction and preliminary-stage data structuring, the design sidesteps the latency and bottlenecks normally caused by cloud-reliant solutions. It further provides the farmer instant access to the field's real-time status, serving as the foundation for both manual and automated decision-making processes [3][6].

4.2 Automated Sprinkler Activation

Automated sprinkler triggering is the central working function that takes the system from a passive sensor device to an active irrigation management system. As soon as the Raspberry

Pi obtains real-time soil moisture data from multiple ESP32 nodes, it compares the information with a specified threshold value for example, when moisture levels fall below 30 percent, irrigation is required. This threshold can be set according to crop type, soil condition, or seasonal needs, thus making the system flexible for various agricultural scenarios.

Upon comparing the received values with the threshold, the Python script of the Raspberry Pi decides whether a sprinkler should be activated or deactivated for each respective ESP32 node. If the water level at any given sensor falls below the set threshold, the system will immediately send a command to that ESP32 node to turn on its associated sprinkler. The sprinkler stays on until the sensor indicates that the soil water has returned to a healthy level. This is done through a feedback loop to make sure that water is being utilized effectively and only when it is needed.

The exchange of signals between the ESP32 nodes and the Raspberry Pi is done across a WiFi network, enabling rapid and wireless signal transmission without wired infrastructure complexity. Each ESP32 board is pre-programmed through the Arduino IDE with bespoke logic to decode control commands and actuate the sprinkler relay from its GPIO pins. The relay is a switch that opens or closes the flow of water upon receiving digital inputs from the microcontroller.

This entire process happens autonomously and continuously, reducing the need for manual labor and ensuring that crops receive water precisely when they need it. This not only conserves water but also leads to healthier crop growth and improved yield over time. The automation also allows farmers to focus on other tasks, knowing that irrigation is being handled intelligently in the background [3][9].

4.3 Real-Time Web-Based Monitoring

One important aspect of the system is its capability to provide real-time web-based monitoring, enabling landowners or farmers to view the moisture status of their fields remotely anywhere through an internet connection. This is attained through the integration of Python scripts executing on the Raspberry Pi and a Flask web server hosting a private dashboard. The web interface is clean and easy to use, showing live sensor readings, sprinkler statuses, and timestamps, making it easy even for users with minimal technical experience to comprehend the condition of the field at a glance.

As the Raspberry Pi receives moisture readings from the ESP32 sensor nodes, it computes this data and refreshes the display on the dashboard in real time. Flask accomplishes this with dynamic HTML templates that are updated at regular intervals, allowing the user to view updated readings in near real time. In addition to the numeric data, the system also translates the moisture levels into clear indicators like Dry, Optimal, or Wet, based on the defined moisture threshold. This translation assists users in making sound decisions promptly.

The dashboard's responsiveness is the system's greatest strength. Owing to Flask being a lightweight and very efficient web framework, the updates are smooth without overloading the Raspberry Pi. Furthermore, the dashboard may be modified to incorporate manual control switches that allow users to override the automatic sprinkler system in case they need to. This proves particularly handy in maintenance or emergency watering situations where automated reasoning cannot accurately reflect the field's immediate requirements.

The biggest plus point is that this monitoring function does not confine itself to a local network. With NGROK integration, a safe tunnel is formed between the local Flask server and the broader internet so that the dashboard can be remotely accessed. This enables farmers to remain in touch with their irrigation system wherever they are, delivering unprecedented ease to field management.

This real-time monitoring system not only increases transparency but also provides increased user control over the irrigation process. Having live data at hand, farmers have a better chance of managing resources in an efficient manner and adapting to environmental changes in a timely manner [3][11][14].

4.4 Role of NGROK in Web Hosting

In conventional web development, hosting a site on the internet typically involves buying a domain, setting up DNS configurations, and installing the server on a public hosting platform or virtual machine. But for small-scale or prototype-level projects such as this decentralized smart agriculture system, these steps can be convoluted and unnecessarily expensive. This is where NGROK comes into the picture by functioning as a tunneling service that fills in the gap between a local development environment and the world-wide internet.

NGROK enables the Flask server on the Raspberry Pi to be made available on the internet securely. It

achieves this by generating a temporary public URL that forwards all traffic directly to the local server. Consequently, users can access the web dashboard anywhere, using a browser and the given NGROK link, without having to establish conventional hosting infrastructure.

This remote access is particularly crucial for farmers who may not be constantly around the physical installation. With NGROK installed, they can monitor their field's moisture levels, receive live updates, and even bypass sprinkler controls from their phone or laptop. The ease of this solution is perfect for decentralized applications, where cheap and self-hosted hosting is preferable to third-party cloud providers.

Another advantage of employing NGROK is the security it provides. It creates encrypted HTTPS tunnels, and as such, the data being exchanged between the client and server is kept private and secure. This is especially useful in agricultural settings where data privacy is increasingly becoming an issue. Farmers can be confident that their irrigation data is in their hands, which fits perfectly with the decentralized vision of the whole project.

In effect, NGROK turns a web application that runs locally into a publicly accessible one with very little setup. Its implementation in this system eliminates the necessity for cloud hosting centrally and allows field owners to have real-time control and observation of their operations from anywhere in the world virtually [14].

4.5 Data Flow and Decision Logic

Any automated irrigation system succeeds not just because of the hardware employed but also the ease with which data is made to flow through the system and decisions are taken based on the data. In this project, data flow is carefully designed such that every component efficiently plays its part to deliver correct and timely irrigation action with no human intervention.

The operation starts right from the bottom of the system, the soil moisture sensors that are linked to ESP32 boards. The sensors continuously read the moisture levels of the soil and transmit those values to their corresponding ESP32 nodes via GPIO pin connections. The ESP32 boards are configured to read this sensor data at intervals of a few seconds and transmit it to the Raspberry Pi master node over WiFi. The information is sent in a light and standardized form via JSON, which is simple to parse and

store once it arrives at the Raspberry Pi [3].

Once received, a Python script on the Raspberry Pi processes each incoming reading. The information is stored in a CSV file for simple record-keeping, so the landowner can trace back previous readings and monitor field trends over time. Concurrently, the program verifies whether the moisture level for any node is below a threshold value. On detecting a low-moisture condition, a decision is made to trigger the sprinkler corresponding to the node through GPIO control. The system automatically deactivates the sprinkler as soon as soil moisture is regained to an optimum level in order to save water.

Side by side, the Raspberry Pi also refreshes the web dashboard created with Flask. The web dashboard shows the current sensor readings, the status of all sprinklers, and a timestamp on when the data was updated. With the smooth integration with NGROK, the web dashboard can be accessed online, which is the ability to watch these updates in real time from any part of the globe [11][14].

This movement of data and logic produces a closed loop system, sensors gather data, the Raspberry Pi analyzes the data and makes decisions, and then the output is shown in both the physical irrigation as well as the digital dashboard. The design does not include delays and bottlenecks since everything is processed at the edge, near where the data is coming from. Most importantly, it gives complete control and visibility to the end user, allowing for informed farming decisions without relying on any external cloud-based infrastructure.

By maintaining the decision logic close to local and in close association with real-time data computation, the system guarantees reliability, latency, and privacy. It makes what was once a slow, manual process an active, automated mechanism driven solely by live soil conditions.

CHAPTER 5

Advantages of the Proposed System

5.1 Increased Privacy and Security

One of the most important benefits of this intelligent irrigation system is how it maximizes privacy and security by keeping data in the user's hands. In conventional IoT configurations, data ends up being forwarded to outside cloud servers for processing and storage. Although this can be convenient, it also presents critical issues — sensitive farm information may be left vulnerable to breaches, unauthorized entry, or even improper use by outsiders [2][8]. To farmers whose livelihood depends on exact, up-to-the-minute intelligence regarding their own land, that sort of risk can't be underestimated.

This project sidesteps that problem altogether by decentralizing the data stream. All sensor data and control logic remain within the local network, being processed directly by the Raspberry Pi. That means the farmer doesn't need to worry about internet connectivity issues or transmitting private data to commercial cloud providers. It also eliminates the necessity of trusting outside platforms with the farm's operational data, with only the landowner having access and control [6].

In addition, by utilizing NGROK to expose the monitoring dashboard temporarily as required, but not as a permanent hosting service, the system has a minimal digital footprint but also provides convenience and adaptability. NGROK is a tunnel, not storage, so the data privacy stays intact even when accessed remotely.

Security is further augmented by the reduced surface area of attack. Without constant access to outside servers, and with less reliance on internet connectivity, the system naturally circumvents many of the vulnerabilities that afflict standard IoT configurations. Later iterations of the project might include further encryption, authentication, or even blockchain to further protect and log each data transfer, but even as it stands, the methodology is a significant improvement on maintaining agricultural data private and secure.

In an age where data is too frequently commoditized, this system provides a nice change of pace — one where the farmer retains, controls, and profits from their data without sacrificing anything.

5.2 Lowered Operational Expenses

One of the primary objectives of this project was to design a system that's not only intelligent but also economical in the long term. Most of the current smart irrigation systems have ongoing costs associated with them — subscription to cloud platforms, data storage charges, and even licensing fees for analytics software. In the long term, these indirect costs accumulate, and it becomes increasingly difficult for small and medium-sized farmers to implement such technologies [8][13].

By creating a decentralized system that operates locally, this project eliminates the use of third-party cloud services altogether. There is no monthly data hosting fee, no reliance on commercial APIs for making decisions, and no mandatory upgrades based on proprietary software. All of the data is gathered, processed, and responded to directly through the Raspberry Pi and ESP32 network — hardware that is both inexpensive and readily available.

In addition to lowering tech-related costs, the system also reduces water usage through automated, data-driven irrigation. Since sprinklers are triggered only when moisture levels drop below a certain threshold, water is used more efficiently. This not only supports environmental sustainability but also helps farmers save on utility bills and reduce the effort spent on manual monitoring and watering.

The cost of setup itself is low, particularly in comparison to industrial IoT platform scales. All the parts utilized — soil moisture sensors, ESP32 boards, and a Raspberry Pi — are inexpensive yet capable of providing for the demands of an average farm plot. And since it all runs locally, no high-speed internet connections or commercial data plans are required to simply keep the system up and running.

Essentially, the project presents a high-technology solution that is not very expensive. It paves the way for smart farming practices on a larger scale, especially where resources are limited but the urge for innovation is great.

5.3 Scalability and Flexibility

A system is no more useful than its capacity for growth and modification — and that's where this project excels. Scalability was integrated into the architecture from the very start. A farmer with a small

plot of land or an individual running multiple acres has the same fundamental system that can just be grown by incorporating additional ESP32 nodes with sensors and sprinklers.

Because every node operates wirelessly and talks to the Raspberry Pi via WiFi, there is no need for complicated wiring or rearranging infrastructure. If additional areas of the field must be reached, it's simply a matter of installing a few more modules and configuring them to connect to the network. This modular setup maintains flexibility. Farmers can begin small, pilot the system on a small area, and expand incrementally as they become more at ease with the technology or with the passage of time.

Flexibility is not simply with the hardware, however — it's also inherent with the software and control logic. Soil moisture threshold values can be tailored, the data update periods can be modified, and even the web dashboard can be tailored to fit various crops or conditions. That is to say, the system is not set in a single fixed configuration. It can be adjusted to fit the specific requirements of every field or season, enabling greater accommodation of local agricultural practices.

Additionally, utilization of open-source technology such as Raspberry Pi and ESP32 makes it possible for future integrations. Whether incorporating a weather forecasting API, integration with a solar-powered source, or synchronization with a mobile app, the system is adaptable enough to change without the necessity for an entire reboot.

By designing the system to be scalable and versatile, this project guarantees it can remain applicable not only for one season of growth, but for years — regardless of how the needs or space of a farmer may change.

5.4 Real-Time Monitoring and Control

One of the best aspects of this system is its capability for real-time monitoring and control, which significantly boosts a farmer's capacity to take decisions on time. Farmers usually have to make do with timetabled watering sessions or eye-watching while in conventional irrigation systems. It results in inefficiencies like over-watering and under-watering, both of which harm the crops and result in wastage of precious water resources.

With this intelligent irrigation system, growers can track moisture levels in the fields from an internet-based dashboard, refreshed every 5 minutes. When moisture falls below a set threshold, the system can

automatically turn on the sprinkler, so that water is supplied precisely when and where it is required. This dynamic feedback loop lowers water loss, reduces the required amount of intervention, and assures that crops have the appropriate level of water at the appropriate moment.

The virtue of this methodology is its directness and immediacy. Growers don't need to stand in the field to observe what's happening there, nor must they wait for scheduled reports. No matter if they're present or not, they can control the system remotely through the web interface to adjust or just look in. This type of control at their fingertips places farmers in the driver's seat of their irrigation systems, allowing more efficient and proactive use of their resources.

Since the system runs entirely locally and is not dependent on third-party cloud services, it guarantees uninterrupted and reliable real-time control. If a farmer is facing changing weather patterns or surprise shifts in soil moisture, he can count on the system to respond immediately — and most importantly, without lag or reliance on outside servers.

CHAPTER 6

Futuristic Perspective of the Project

6.1 The Changing Role of Cloud Computing

Cloud computing has been instrumental in reshaping the environment of smart agriculture by providing remote access to data, scalable computer resources, and centralized control systems. Conventional cloud-based IoT models provide real-time monitoring and data analysis capabilities, enabling farmers to make educated decisions regarding irrigation timing and resource allocation [6]. Nevertheless, as the magnitude and complexity of IoT networks expand, cloud computing is confronted with escalating challenges with regard to latency, bandwidth utilization, and data privacy [4].

Smart irrigation systems may benefit from third-party cloud platforms at their own detriment in terms of greater operational expenditures and service breakage risks via changes in externally owned policies or outages [8]. Beyond this, there are privacy considerations to be met as a result of the decentralized approach of the cloud architecture since it stores and processes sensitive information from agriculture externally with no farm-direct control or involvement [2].

To overcome these shortcomings, the market is slowly moving toward a more hybrid strategy where cloud computing is supplemented by edge and decentralized solutions. Edge computing, more specifically, enables processing of data closer to the source—on devices such as Raspberry Pi or ESP32—reducing latency and lessening dependency on the cloud [4][6]. As this evolution progresses, cloud computing will increasingly take a supporting role by processing tasks that involve intensive computation or data storage over long periods, with real-time control and monitoring migrating to edge devices for quicker, secure, and localized decision-making [7].

This dynamic model facilitates increased farmer autonomy and mitigates reliance on internet connectivity for mission-critical functions such as irrigation management. Decentralizing computing resources supports the ideal of self-reliant agricultural ecosystems and facilitates the objectives of secure, efficient, and privacy-conscious smart farming solutions [5][9].

6.2 The Future of Cloud Computing

The future of cloud computing is likely to be more adaptive, decentralized, and collaborative, particularly in the case of IoT-based applications such as smart agriculture. Although conventional centralized cloud models have provided tremendous scalability and processing capacity for data, new requirements for privacy, low latency, and offline capabilities are compelling the transformation of cloud services toward a more distributed architecture [6][7].

On farms, the future of cloud platforms is expected to coexist alongside edge and fog computing layers. These designs enable vital information to be analyzed locally on devices like Raspberry Pi or ESP32 modules, with only appropriate summaries or long-term analysis being transferred to the cloud [4]. This blended strategy not only reduces latency but also allows real-time responses—such as the activation of sprinklers—to remain unaffected by internet connectivity.

Furthermore, embedding decentralized technologies, including blockchain and peer-to-peer storage solutions, is on the verge of revolutionizing data storage, dissemination, and validation in the cloud [5][9]. These technologies provide greater transparency, immutability, and data ownership control, which are in perfect conformity with Web 3.0 ideals and escalating interests in data sovereignty [3].

As cloud computing technology advances, it will probably migrate from being the central decision support system to the backend facilitator of big analytics, historical patterns, and forecast modeling. Smart agriculture will then have cloud platforms complementing the edge devices in terms of provisioning services like weather forecasting APIs, long-term analysis of yields, and machine learning-driven irrigation forecast [10][13].

Overall, the future of cloud computing is in its integration with decentralized models, empowering scalable, robust, and farmer-focused agriculture systems that can learn to respond to local conditions while benefiting from global expertise.

6.3 Decentralization in the Context of IoT

Decentralization in the context of the Internet of Things (IoT) represents a paradigm shift away from conventional centralized models towards architectures that give individual devices and users more autonomy, security, and control over their data. In centralized IoT systems, data gathered from edge

devices is usually sent to cloud servers for processing and decision-making. Even though this model simplifies infrastructure management, it poses significant problems in the form of latency, single points of failure, and greater exposure to data breaches [2][5].

Decentralized IoT networks overcome these limitations by dispersing computation, storage, and decision-making capabilities throughout the network. In a decentralized system, edge devices such as ESP32 nodes or Raspberry Pi units perform data processing locally and operate autonomously according to predetermined logic, minimizing reliance on ongoing internet connectivity or third-party cloud services [4][6]. This not only improves system dependability but also guarantees that time-critical operations—such as irrigation control—are not hindered by network bottlenecks or server crashes.

In addition, decentralization ensures greater data privacy and ownership. Since the farm data is retained at local nodes or within farmer-owned infrastructure, the threat of unauthorized access or exploitation is much lower [2][3]. Technologies such as blockchain take it a step further by providing immutable, auditable records of transactions, making information sharing secure without requiring a central point of control [5].

The decentralized method also enhances scalability. Sensors and actuators can be introduced into the system with minimal reconfiguration, making it suitable for large or geographically dispersed farms [7]. With ongoing research and development in this field, decentralized IoT systems are likely to be at the core of emerging smart agriculture solutions, as part of the larger vision of Web 3.0 and self-sovereign digital economies [3][9].

6.4 The Evolution of the Web

The World Wide Web has evolved in a very impressive way since its beginning, progressing through unique phases that portray changing user needs, technological breakthroughs, and social changes. The first phase, referred to as Web 1.0, was mainly static and read-only. The user could read information, but there was limited interaction, and the content was largely managed by a few institutions or content producers.

Next was Web 2.0, with interactivity and user-generated content. This phase saw the development of

social networking sites, blogs, and collaboration applications where the users were now active contributors. Yet, even as Web 2.0 opened up innovation and connectivity, it also opened up the way to data centralization. Some big companies controlled much of the information online, and concerns arose regarding privacy, surveillance, and data monopolies [2][3].

Today, the internet is moving into a new era commonly known as Web 3.0. Web 3.0 is characterized by decentralization, trustless messaging, and empowerment of the user. Rather than data being stored and owned by centralized websites, Web 3.0 is based on systems where users own their data and communicate via peer-to-peer networks, typically supported by blockchain technology [5][9]. It's not a technical revolution — it's a change of mindset. The thought is to empower people by giving them power to have transparent and secure interactions without the need for a single entity.

In IoT-based systems such as smart agriculture, this change has serious consequences. Devices can communicate directly with one another, make local decisions, and keep data in decentralized ledgers when necessary. This not only minimizes dependency on cloud services and internet connectivity but also perfectly aligns with the ethics of privacy, autonomy, and resilience [3][6]. The project presented in this report is evidence of this shift, bringing the concepts of Web 3.0 into a practical application that directly impacts farmers.

As Web 3.0 unfolds, it holds the potential to reimagine not only how we navigate or exchange content, but how we construct systems, handle data, and interact across digital and physical domains.

6.5 Project Alignment with Web 3.0 and Decentralization Principles

This project highly resonates with the fundamental vision of Web 3.0 and the general vision of decentralization. In its essence, Web 3.0 is all about empowering users with control over data, minimizing dependence on central authorities, and facilitating secure peer-to-peer interactions. These concepts are exactly mirrored in the architecture of the proposed smart irrigation system.

By removing third-party cloud services and making all data processing and decision-making local to the Raspberry Pi and ESP32 nodes, the system values user autonomy and data ownership. Farmers are no longer reliant on third-party platforms or internet connectivity to control their irrigation schedules. Rather, they have complete control over their infrastructure, including real-time monitoring and local

decision logic — a definite step toward decentralized empowerment [2][6].

Also, the system is an example of Web 3.0's focus on edge computing and trustless operation. Each sensor node runs semi-autonomously, detecting moisture levels in the soil and turning on sprinklers without the requirement of centralized authorization, thereby replicating the decentralized, trust-reduced mechanisms common in blockchain systems [4][5]. Blockchain isn't used here directly, although the architecture can easily support augmentations in the future like immutable logging of data or the use of smart contracts for automation, both of which are core to the Web 3.0 initiative [9].

At a larger scale, the project helps usher in the transition to sustainable, scalable, and secure digital agri-ecosystems. It imagines a world where each farm can be an autonomous smart unit, immune to network outages and vendor lock-in. This autonomy doesn't only provide technical advantages — it also gives back control to farmers, aligning with the social and ethical goals of decentralized technology [3][5].

In short, the project not only takes up Web 3.0 technologies — it takes up its ethos. It's a concrete demonstration of how decentralized values can be brought down to earth and used to solve real-world issues, making smarter, more equitable, and more sustainable systems.

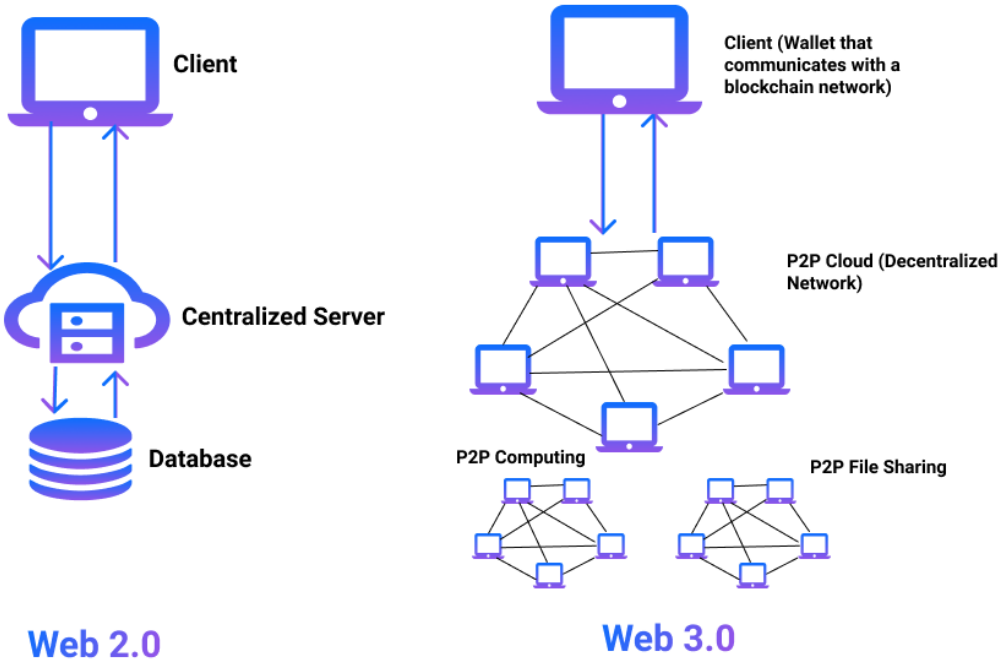

Figure 6.1: Web Phases

CHAPTER 7

Experimental Results and Analysis

7.1 ESP32 Moisture Detection and Output

For checking the moisture sensing function of the ESP32, a simple but efficient method was employed. The soil moisture sensor was put into a small water bowl to represent wet conditions and taken out to represent dry conditions. This simple setup enabled us to see how the sensor reacts to different levels of moisture.

In the wet condition, the sensor read a high level of moisture and transmitted the corresponding signal to the ESP32. This was also indicated in the real-time data presented on the web interface, indicating the moisture reading as being above the threshold level. Conversely, when the sensor was taken out of the water and subjected to dry air, the moisture reading decreased, which sent a low signal to the ESP32. This habit was also readily exhibited on the web dashboard, indicating that the soil was parched and that irrigation could be required.

This test configuration proved the sensor's capability to clearly differentiate between wet and dry states. It was an important step in ascertaining the system's functionality prior to integrating it into a more extensive field-based application. Through this straightforward testing approach, we were able to verify the ESP32's capability to accurately detect moisture levels and initiate the required irrigation processes when necessary.

And the output can be viewed in any device wifi IP 192.168.4.1 which is connected to ESP32 node:

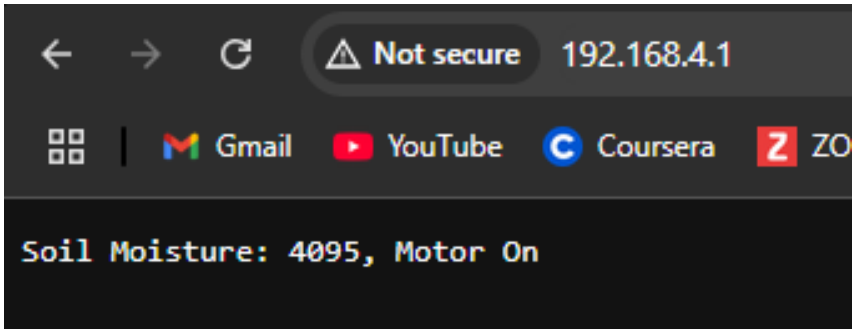

Figure 7.1: Single Node Output

7.2 Python Script for Raspberry Pi Data

The Python script deployed for data retrieval from the ESP32 sensors is meant to retrieve soil moisture data on a regular interval from all the sensors and record it into a CSV file. The script connects to the Wi-Fi network of every ESP32 sensor, extracts the moisture data using an HTTP request, and logs the data with the timestamp.

Firstly, the script accesses the given Wi-Fi network of every ESP32 module. After connection, an HTTP request is sent to the ESP32's IP address where the data from the moisture sensor can be accessed. Then, the moisture value is fetched and printed to the console for confirmation. After every data gathering cycle, the Raspberry Pi disconnects from the Wi-Fi network for the next cycle.

Data is logged in a CSV file named data.csv, which records the moisture values as well as the associated timestamp for every reading. This configuration is essential to maintain the record of soil moisture levels over time, so future analysis and decision-making can be made.

The program executes in a loop, reading data from all the sensors every five seconds, and continues to do so indefinitely. It provides continuous monitoring of soil conditions without requiring manual intervention.

The output is as follows:

```
>>> %Run wifi.py
Connecting to ESP_SoilSensor_1...
Device 'wlan0' successfully activated with 'd454cd84-67e7-4f69-a8ca-1db5d691d390'.
Soil Moisture from 192.168.4.1: Soil Moisture: 45%
Device 'wlan0' successfully disconnected.
Collected data: Soil Moisture: 45%

Connecting to ESP_SoilSensor_2...
Device 'wlan0' successfully activated with '1933ff20-9940-46f5-8d7e-a803f41686b8'.
Soil Moisture from 192.168.4.1: Soil Moisture: 2154, Motor Off
Device 'wlan0' successfully disconnected.
Collected data: Soil Moisture: 2154, Motor Off
Waiting for 5 minutes before next cycle...

Connecting to ESP_SoilSensor_1...
```

Device 'wlan0' successfully activated with 'd454cd84-67e7-4f69-a8ca-1db5d691d390'.
Soil Moisture from 192.168.4.1: Soil Moisture: 45%
Device 'wlan0' successfully disconnected.
Collected data: Soil Moisture: 45%

Connecting to ESP_SoilSensor_2...
Device 'wlan0' successfully activated with '1933ff20-9940-46f5-8d7e-a803f41686b8'.
Soil Moisture from 192.168.4.1: Soil Moisture: 4095, Motor On
Device 'wlan0' successfully disconnected.
Collected data: Soil Moisture: 4095, Motor On
Waiting for 5 minutes before next cycle...

7.3 Website HTML and Outputs

The website interface was created using simple HTML and CSS, making it clean and responsive across devices. This page is designed to be hosted locally on the Raspberry Pi and acts as a dashboard for landowners to view their soil moisture data.

When the moisture data is logged into the CSV file by the Raspberry Pi, it is also read by the backend and displayed dynamically using a templating engine. The page is styled in a calm green theme to reflect the agricultural context. It includes a centered container with a title, a welcome message, and a data table that updates with real-time values.

If the soil data exists, it is shown in a table with the moisture value and the timestamp of when it was recorded. If no data is available yet, the page displays a message indicating that.

Below is the HTML code used to display the website:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Agricultural Website</title>
</head>
<body>
   <div class="container">
      <h1>Your Agricultural Website</h1>
      <p>Welcome to your locally hosted site for modern farming and innovation.</p>
```

```
    {% if soil_data %}
    <table>
      <tr>
        <th>Moisture Value</th>
        <th>Timestamp</th>
      </tr>
      {% for entry in soil_data %}
      <tr>
        <td>{{ entry.moisture }}</td>
        <td>{{ entry.timestamp }}</td>
      </tr>
      {% endfor %}
    </table>
    {% else %}
    <p>No data available yet.</p>
    {% endif %}
  </div>
</body>
</html>
```

7.4 Local Hosting with Flask

To make the moisture data accessible via a browser, we used **Flask**, a lightweight Python web framework. Flask was used to host the HTML page on the Raspberry Pi locally, enabling users on the same network to view real-time soil moisture readings.

The backend reads the data stored in the CSV file and passes it to the front-end HTML page using the render_template function. This creates a dynamic webpage that updates every time the data changes.

This script does the following:

- Initializes a Flask app
- Defines the homepage route /
- Opens and reads the data.csv file
- Skips the header and stores each entry in a list
- Passes the list to the index.html template
- Runs the app locally on port 5050

When this server runs, visiting http://localhost:5050/ in the browser displays the moisture readings with timestamps.

7.5 Ngrok Tunneling and Output

After getting the Flask server to run locally at port 5050, accessing the website over the internet would be the next step. That is where Ngrok comes very handy.

Ngrok is a simple and powerful tool that creates a secure tunnel between a local server and the internet. Essentially, it provides a temporary public URL that forwards requests to your locally running web server. This allows others—whether testers, collaborators, or clients—to access your web app from anywhere, even if it’s running only on your laptop and not deployed to the cloud.

To turn on this tunnel, we issue the following command:

ngrok http 5050

This instructs Ngrok to direct all incoming internet traffic from a public URL to port 5050 on your local machine, where your Flask app is listening.

How Ngrok Works

Behind the scenes, Ngrok establishes a secure TLS tunnel between your local machine and the Ngrok cloud service. When you execute the above command, Ngrok connects to its cloud infrastructure and establishes a publicly accessible URL such as https://random-id.ngrok.io. This URL is then mapped directly to your machine's localhost at the given port. Any request to this public URL is forwarded securely to your local development environment [12].

The primary advantage here is that there is no requirement to set up firewalls, port forwarding, or host your application on a public server while developing. This is very useful for testing APIs, showcasing web projects, or remotely invoking IoT-based scripts without having to host them permanently in the cloud [9].

Yet, let us keep in mind that this approach is usually designed for development and demonstration purposes. In production deployments, cloud hosting or edge computing models are more secure and scalable [6], [14].

Here is the screenshot of the internet-hosted website using Ngrok:

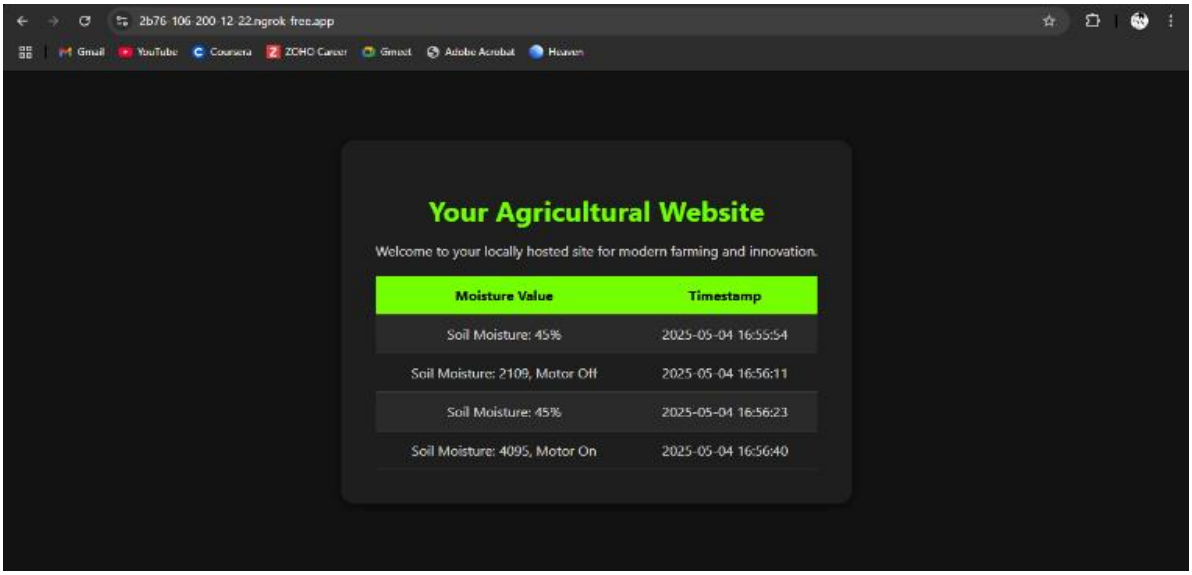

Figure7.2: Ngrok Final Output

Chapter 8

Challenges and Limitations

8.1 Connectivity Constraints:

Among the challenges faced in implementing a decentralized smart agriculture environment is one of connectivity, particularly considering the use of WiFi-based wireless communication between elements such as ESP32 modules and Raspberry Pi. Although WiFi may be readily deployable and economically viable for local small-scale scenarios, it entails some fundamental negatives in field uses.

The main limitation is limited range. Typical WiFi modules like those found in ESP32 boards provide stable connectivity up to about 30 meters inside and 50 to 100 meters outside under optimal conditions. But for agricultural settings with sensors dispersed over large fields, stable WiFi connectivity proves to be a problem. Trees, atmospheric conditions, and landscape further attenuate this effective range.

Additionally, attempting to get around this drawback by utilizing extra routers, range extenders, or setting up mesh networks will greatly add cost and deployment complexity. Not only do such solutions demand more hardware, they will also need constant power supply and setup carefully, which is impossible in distant rural farming communities.

Another option such as LoRa or Zigbee can provide long-distance communication, but they need different hardware modules and integration logic, which was out of the scope and budget of this implementation. Cellular modules are yet another option but introduce recurring data charges and are network-dependent.

Thus, although WiFi offers a simple and quick method of prototyping and testing the system at a small scale, it is not suitable for large-scale agricultural implementation where there are large distances and dispersed sensors. Future development can look into hybrid systems that optimize cost, range, and power efficiency.

8.2 File Handling Between Multiple Processes

In a multi-process system, controlling access to common resources such as files may be difficult. Although opening a file in append mode ('a') permits writing to the file without truncating its content, it

may create issues in certain environments. For instance, in Windows, if a file is opened in append mode, the file becomes locked, i.e., no other process can open it at the same time. This locking process is a Windows file system's intrinsic behavior to avoid data corruption.

In contrast, Linux operating systems treat file access differently. Opening a file in append mode under Linux does not lock the file, and different processes can read from it concurrently. This implies that even while a process writes to the file, other processes can access it for reading purposes, making shared resources more easily handled within a multi-process environment.

This behavior difference is something to keep in mind when developing systems that depend on concurrent file access. It's important to know the operating system-specific features so that conflicts can be avoided and operations can be smooth across platforms.

8.3 Multiple WiFi Connections for Concurrent Tasks

Having multiple WiFi connections running concurrently on a single Raspberry Pi is not easy. In my installation, I needed to handle two connections concurrently: one for data retrieval from the ESP sensors and the other for hosting the site on the internet via Ngrok. The native WiFi on the Raspberry Pi was utilized to access the ESP devices and fetch sensor data. In the meantime, I required a dedicated connection to the internet in order to execute Ngrok for remote access to the website hosted locally.

To resolve this problem, I bought a small WiFi dongle costing only 200 INR. This dongle enabled me to create an additional WiFi connection (wlan1) in the Raspberry Pi. The native WiFi was kept connected with the ESP devices, and the dongle was used for internet connectivity with Ngrok hosting. This solution was efficient and budget-friendly, letting me execute both processes at the same time without any interference.

The functionality and output were smooth, courtesy of the incorporation of the WiFi dongle, and this addressed the problem of few connectivity options on the Raspberry Pi. It also facilitated the concurrent handling of both the local data fetching and the public access of the website without interruption.

Chapter 9

Conclusion

9.1 Project Summary

This project aimed to develop a decentralized IoT solution for smart agriculture, with a focus on sustainability and effective resource utilization. The system was intended to monitor real-time soil moisture levels through ESP32-based sensors, which send information to a central server for analysis and display on a local website. The main goal was to do away with the necessity of cloud-based storage via centralization, rather using edge computing methods to process and save data locally on users' devices. Flask was used for hosting locally over the web and Ngrok for enabling the site's accessibility over the internet. This way, it facilitated farmers to remotely check moisture levels on their land, thereby advising them on irrigation schedules. The solution also covered issues of hardware connectivity challenges, energy reliance, and data privacy, making the system effective, secure, and scalable for use in actual agricultural application.

9.2 Key Takeaways

Several important lessons were learned during the course of developing this project.

- Decentralization and Privacy: One of the key takeaways from this project was understanding that decentralization offers greater data privacy and security control. Hosting the data on local devices and steering clear of cloud-based methods lowered the potential for data breaches and kept users' information under their control. Ngrok was used to facilitate temporary public access while maintaining the privacy of local machine data.

- Scalability Issues: One of the critical issues was to make sure that the system scaled across devices. We solved this by optimizing the process of handling files between multiple processes. This was especially tough on Windows, where the 'a' append mode caused file locking, while Linux systems dealt with the reading and writing of files more accommodatingly.

- Connectivity and Power Solutions: The integration of multiple WiFi connections by using a small WiFi dongle resolved the issue of utilizing one WiFi for data download and another for Ngrok hosting. Further, the use of solar-powered nodes and weather forecast APIs were also seen as possible solutions for ensuring energy-efficient and sustainable operations in the field.

- Practical Application in Agriculture: The project also pointed to the practicality of IoT in enhancing agricultural productivity. With the automation of irrigation decisions from real-time soil moisture levels and external environmental conditions, farmers can minimize water usage, prevent crop damage, and enhance yield. The system is powerful but straightforward, showing how IoT can transform conventional agricultural practice.

9.3 Final Remarks on Decentralization and Sustainability

This project has given very insightful information on the role played by decentralization in data privacy and system responsiveness. By shunning centralized cloud servers and opting for edge computing and local storage, we not only increased the security and privacy of the data of the users but also ensured the system responsiveness. The storage of the data in a local form enabled us to process things faster and in a more effective manner.

On the sustainability side, incorporating solar-powered nodes is a primary strategy for ensuring that the system can be deployed in rural locations without concerns over constant power supply. This measure makes the system more independent and environmentally friendly, minimizing dependence on external power supplies and making it more feasible in off-grid environments.

The prospects for IoT in agriculture are rosy, particularly when it is combined with decentralized methods, low-energy systems, and sophisticated technology such as weather forecasting APIs. The success of the project indicates the potential for large-scale, sustainable IoT solutions to enhance agricultural productivity while helping to tackle major global issues such as water scarcity and climate change. With more advancements, this system may be increased in terms of sensors and capabilities, like monitoring plant health, to allow farmers to know even more about their crops' requirements. In the end, this project has shown how the blend of technology, decentralization, and sustainability can

make a more efficient, more secure solution for today's agricultural sector.

Chapter 10

Future Enhancements

10.1 Adding More Features for Each Node

Each node may be fitted with extra sensors to monitor the health of plants more in-depth. A leaf sensor, for instance, might be able to sense signs of stress, disease, or nutrient deficiencies in plants, allowing for more accurate interventions. With such sensors added, the system might give farmers more detailed information about the health of their crops, enhancing yields and minimizing waste of resources.

10.2 Solar-Powered Nodes

By incorporating solar power into every node, the system would be more sustainable, particularly in off-grid or remote locations. Solar-powered nodes would decrease reliance on conventional power sources and provide constant operation, even in the event of a power outage or in areas where electricity is not reliable. This would not only make the system more environmentally friendly but also lower operational costs, making it suitable for long-term deployment in agricultural environments.

10.3 Weather Prediction API Integration

The addition of weather prediction functionality can be another aspect to improve the functionality of the system. By incorporating weather prediction APIs, the system can make irrigation patterns adaptable according to future weather conditions. For example, in case of expected rain, the irrigation system can be suspended, conserving water and avoiding over-watering. This integration would enable the system to become more sensitive to current environmental conditions, optimizing water usage and facilitating smart farming.

REFERENCES

[1] R. G. Smith et al., “Smart Irrigation Systems: Challenges and Approaches,” IEEE Internet of Things Journal, vol. 6, no. 3, pp. 3971–3983, 2019.

[2] M. A. Jabbar et al., “A Survey on Data Privacy in Cloud-Based IoT Systems,” Journal of Network and Computer Applications, vol. 136, pp. 62–80, 2019.

[3] T. Berners-Lee et al., “Solid: A Platform for Decentralized Data Ownership,” MIT CSAIL White Paper, 2018. [Online]. Available: https://solidproject.org

[4] K. Sharma and D. P. Sharma, “Edge Computing in IoT: A Review,” Procedia Computer Science, vol. 132, pp. 362–369, 2018.

[5] S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System,” 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[6] K. Sangaiah et al., “Energy-Efficient Smart Agriculture using IoT and Edge Computing,” Future Generation Computer Systems, vol. 86, pp. 200–211, 2018.

[7] M. Gheisari et al., “Decentralized Cloud Storage: State-of-the-Art and Future Research Directions,” Journal of Cloud Computing, vol. 9, no. 1, pp. 1–20, 2020.

[8] N. Mohanty et al., “A Review on Cloud Hosting Technologies for IoT Applications,” Materials Today: Proceedings, vol. 49, no. 5, pp. 1651–1657, 2021.

[9] C. Assuncao et al., “Web 3.0 and the Internet of Value: The Role of Decentralization,” IEEE Access, vol. 8, pp. 183655–183674, 2020.

[10] Raspberry Pi Foundation, “Raspberry Pi Documentation,” [Online]. Available: https://www.raspberrypi.org/documentation/

[11] Espressif Systems, “ESP32 Technical Reference Manual,” [Online]. Available: https://www.espressif.com/en/products/socs/esp32/resources

[12] NGROK, “Secure Introspectable Tunnels to localhost,” [Online]. Available: https://ngrok.com

[13] M. Shrestha et al., “IoT-Based Smart Irrigation: Challenges, Applications, and Future Directions,” Sensors, vol. 22, no. 3, 2022.

[14] M. Tran et al., “Smart Agriculture: Technologies and Trends,” Agricultural Systems, vol. 190, pp. 103089, 2021.

[15] N. Z. Naqvi and M. Ilyas, “Privacy-preserving IoT Systems: A Comparative Review,” ACM Computing Surveys (CSUR), vol. 54, no. 2, pp. 1–34, 2021.

Appendices

The Embedded C code with Arduino Libraries that is used for sensing the moisture value is given below:

```
#include <WiFi.h>
#include <WebServer.h>

const char *ssid = "ESP_SoilSensor_2";
const char *password = "12345678";
WebServer server(80);

String getSoilMoistureStatus(int soilMoisturePin, int motorPin) {
   pinMode(motorPin, OUTPUT);
   int moistureValue = analogRead(soilMoisturePin);
   String motorState;
   if (moistureValue < 2500) {
      digitalWrite(motorPin, LOW);
      motorState = "Off";
   } else {
      digitalWrite(motorPin, HIGH);
      motorState = "On";
   }
   return "Soil Moisture: " + String(moistureValue) + ", Motor " + motorState;
}

void handleRoot() {
   server.send(200, "text/plain", getSoilMoistureStatus(34, 4));
}

void setup() {
   Serial.begin(115200);
   Serial.println("Sensor 2");
   WiFi.softAP(ssid, password);
   server.on("/", handleRoot);
   server.begin();
}

void loop() {
   server.handleClient();
   delay(10);
}
```

The Python script for data retrieval from the ESP32 sensors is provided below:

```
import os
import time
import requests
import csv
from datetime import datetime
ESP_SENSORS = [
    {"ssid": "ESP_SoilSensor_1", "ip": "192.168.4.1"},
    {"ssid": "ESP_SoilSensor_2", "ip": "192.168.4.1"}
]
homewifi="Airtel_Kishore Wifi"
def connect_to_esp(ssid):
    print(f"\nConnecting to {ssid}...")
    os.system(f"nmcli device wifi connect {ssid} password 12345678")
    time.sleep(5)
def get_soil_moisture(ip):
    url = f"http://{ip}/"
    try:
        response = requests.get(url, timeout=20)
        if response.status_code == 200:
            print(f"Soil Moisture from {ip}: {response.text}")
            return response.text
    except Exception as e:
        print(f"Failed to get data from {ip}: {e}")
    return None
def log_data_to_csv(data):
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    with open("data.csv", "a", newline='') as csvfile:
        writer = csv.writer(csvfile)
        writer.writerow([data, timestamp])
if not os.path.exists("data.csv"):
    with open("data.csv", "w", newline='') as csvfile:
        writer = csv.writer(csvfile)
        writer.writerow(["Moisture Value", "Timestamp"])
while True:
    for esp in ESP_SENSORS:
        connect_to_esp(esp["ssid"])
        moisture = get_soil_moisture(esp["ip"])
        os.system("nmcli device disconnect wlan0")
        print(f"Collected data: {moisture}")
        if moisture:
            log_data_to_csv(moisture)
    print("Waiting for 5 minutes before next cycle...")
```

```
    time.sleep(5)
```

Below is the HTML code used to display the website:

```
<body>
    <div class="container">
        <h1>Your Agricultural Website</h1>
        <p>Welcome to your locally hosted site for modern farming and innovation.</p>

        {% if soil_data %}
        <table>
            <tr>
                <th>Moisture Value</th>
                <th>Timestamp</th>
            </tr>
            {% for entry in soil_data %}
            <tr>
                <td>{{ entry.moisture }}</td>
                <td>{{ entry.timestamp }}</td>
            </tr>
            {% endfor %}
        </table>
        {% else %}
        <p>No data available yet.</p>
        {% endif %}
    </div>
</body>
```

Below is the Python code used to host the website locally using Flask:

```
from flask import Flask, render_template
import csv
app = Flask(__name__)
@app.route('/')
def home():
    data = []
    try:
        with open('data.csv', newline='') as csvfile:
            reader = csv.reader(csvfile)
            next(reader)
```

```
            for row in reader:
                data.append({"moisture": row[0], "timestamp": row[1]})
    except FileNotFoundError:
        data = []
    return render_template("index.html", soil_data=data)
if __name__ == '__main__':
    app.run(debug=True, port=5050)
```