NITRA TECHNICAL CAMPUS, GHAZIABAD

A Project Report

on

IOT AIDED VEGETAL IRRIGATION ROUTINE WITH PLANT DISEASE DIAGNOSIS AGENT

Submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology

in

Computer Science Engineering

By

Amrit Tyagi (1908020100004)

Varnika Tyagi (1908020100031)

Nadimuddin (1908020100011)

Yash Raj (1908020100034)

Vanshika Tyagi (1908020100030)

Under the Guidance of

Asst. Professor Avnish K. Sharma

Asst. Professor Saurabh Jain

Submitted to:

Department of Computer Science Engineering

Dr. A.P.J. Abdul Kalam Technical University, Lucknow, Uttar Pradesh

(2022 – 2023)

CERTIFICATE

This is to certify that the Project Report entitled, **IOT AIDED VEGETAL IRRIGATION ROUTINE WITH PLANT DIAGNOSIS AGENT** by Amrit Tyagi (1908020100004), Nadimuddin (1908020100011), Vanshika Tyagi (1908020100030), Varnika Tyagi (1908020100031), Yash Raj (1908020100034) for the partial fulfillment of the requirements of Bachelor of Technology in Computer Science Engineering Degree of Dr. A.P.J. Abdul Kalam Technical University, Lucknow embodies the bona fide work done by them under my supervision. The matter embodied in this project report is original and has not been given for the award of any other degree.

Mr. Avnish K. Sharma

(Assistant Professor)

Project Guide

Mr. Saurabh Jain

(Assistant Professor)

Project Guide

Mr. B. K. Sharma

Head of Department (Computer Science)

ACKNOWLEDGEMENT

Primarily, we would like to express our gratitude to our Mentors, Prof. Avnish Kumar Sharma, and Prof. Saurabh Jain, who were a continual source of inspiration. They pushed us to think imaginatively and urged us to do this homework without hesitation. Their vast knowledge, extensive experience, and professional competence in Software Engineering and Electrical components enabled us to successfully conduct this project. This endeavor would not have been possible without their help and supervision. We could not have asked for finer mentors in our studies. This initiative would not have been a success without the contributions of every individual. We were always there to cheer each other on, and that is what kept us together until the end.

We would like to thank the HOD of Computer Science, Prof. BK Sharma, and our institute NITRA Technical Campus for providing us with the opportunity to work on the project **IOT AIDED VEGETAL IRRIGATION ROUTINE WITH PLANT DISEASE DIAGNOSIS AGENT**. Finally, we would like to extend our gratitude to our faculty, peers, and family for their invaluable support, and we are deeply grateful to everyone who has contributed to the successful completion of this project.

DECLARATION

We hereby declare that this submission, is our own work and that, to the best of our knowledge and belief, it has no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree.

Signature:

Name: Amrit Tyagi

Roll No.: 1908020100004

Date:

Signature:

Name: Varnika Tyagi

Roll No.: 1908020100031

Date:

Signature:

Name: Nadimuddin

Roll No.: 1908020100011

Date:

Signature:

Name: Yash Raj

Roll No.: 1908020100034

Date:

Signature:

Name: Vanshika Tyagi

Roll No.: 1908020100030

Date:

TABLE OF CONTENTS

TABLE OF FIGURES

ABSTRACT

India diverse ranging from improvised farm villages to develop farms using modern agricultural technologies. Facility in China is expanding and is leading the world however its ecosystem control technology is still immature with low level of intelligence. Promoting applications of modern information technology in agriculture will solve a series of problems faced by farmers. Lack of exact information and communication leads to the loss in production. This system supplies an intelligent monitoring platform framework and system structure for facility agriculture ecosystem based on IOT. The Internet of Things makes everything connected. Over 50 years of independence, India has made immense progress towards food productivity. Modern agricultural practices have a great promise for the economic development of the nation. So, we have brought in an elevated project for the welfare of farmers and for the farmers. There are no day or night restrictions. This is helpful at any time. Thus, due to its good data handling, decision making capabilities for precise water usage, being portable and user-friendly, this system proves beneficial in-home gardens, greenhouses.

PROLOGUE

CLIMATE CRISIS IN INDIA

Climate crisis has cost India 5 million hectares of crop in 2021.In November, unprecedented rainfall caused huge loss of life and property in south Indian states of Andhra Pradesh, Tamil Nadu, Kerala, and Karnataka. Extreme weather events throughout the year have become the new normal. The decade spanning 2010–2019 was the most turbulent for disasters, Food and Agriculture Organization's (FAO) assessment showed. In India, around 36 mha agricultural area was affected due to hydro-meteorological calamities, including heavy rain and floods since 2016 – 6.65 mha in 2016, 5.08 mha in 2017, 1.70 mha in 2018, 11.42 mha in 2019, 6.65 mha in 2020 and 5.04 mha in 2021. The above data has been aggregated from government replies in the parliament on crop loss and damage in recent years.

This has led to repeated losses for farmers, especially small and marginal ones who comprise over 85 per cent of the total number of farmers in the country. They are being increasingly put to test, as climate change disturbs everything from sowing operations to harvest. *In 2016, damage to crops on 6.65 mha area was estimated to be to the tune of Rs 4,052.72 crore*, the Union government said in the Lok Sabha in 2020. If this is applied to the 36 mha of crop area India has lost since 2016, the figure translates to a whopping Rs 29,939 crore loss for the farmers. However, this would only be an indicative figure as the 2017 evaluation of the crop damage suggests. *In 2017, while the area of crop damage was lower compared to 2016 — 5.08 mha — the value of such damage was more than double – Rs 8,761.39 crore.*

Over a billion people are desperate for a thin layer of soil and a few millimetres of rain to survive. These places do not have enough moisture to plant crops and quench thirsty fields. Drought is looming in every corner of the world, including the densely populated regions of India.

- UP, Bihar, Jharkhand, and West Bengal — the major rice-producing states of India — are in the grips of drought in the absence of enough rain.
- India has now entered the UN's Global Drought Vulnerability Index
- The US announced water cuts as rivers and lakes are hitting dangerous lows.

- England had officially declared drought in many regions.

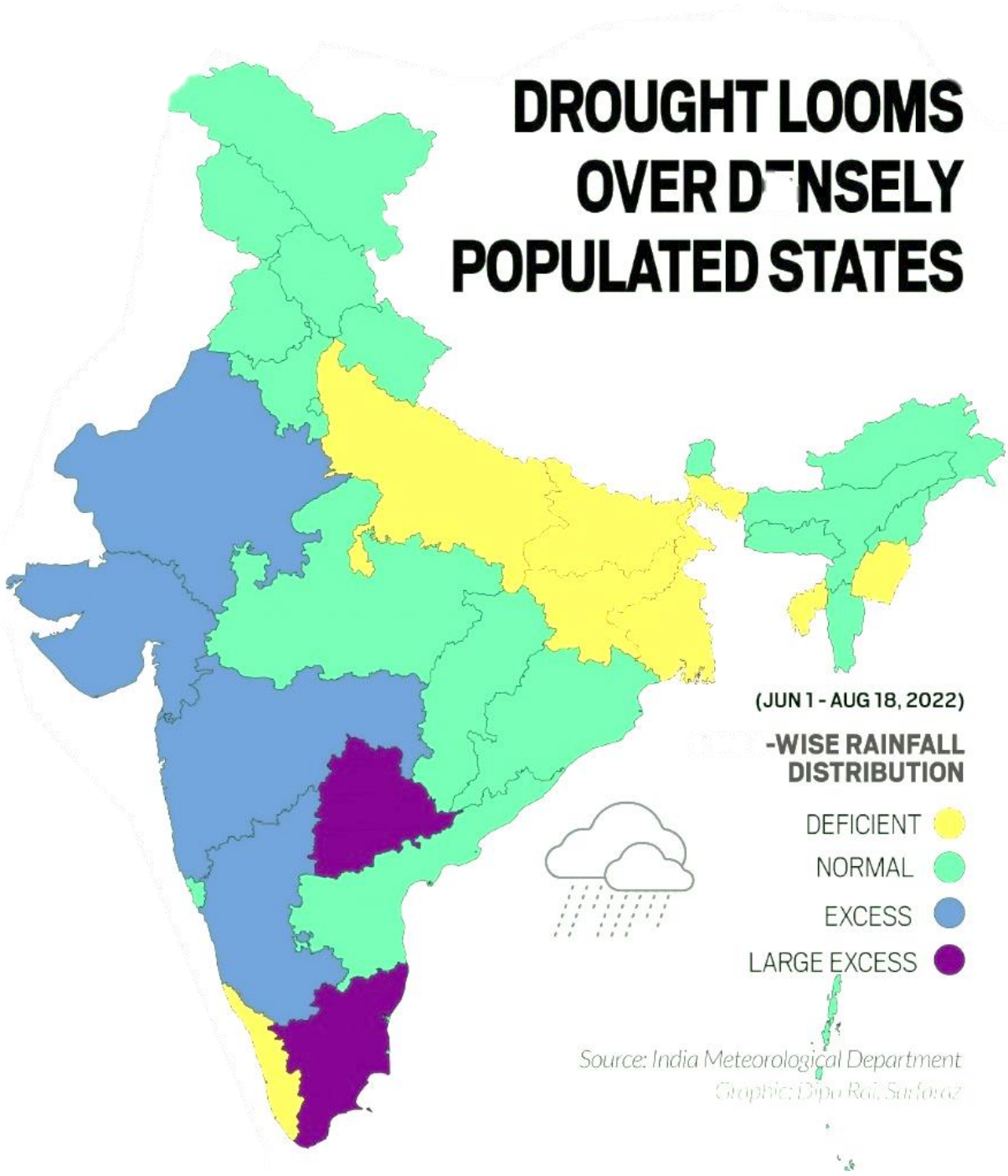

Figure 1 State wise rainfall Distribution

The above figure show Drought becoming a severe problem in the northern states of India.

Dryland farming is largely restricted to these regions having annual rainfall less than 75 cm. Major crops are ragi, bajra, moong, gram, and guar (fodder crops).

Severe drought can also affect air quality by making wildfires and dust storms more likely, increasing health risk in people already impacted by lung diseases, like asthma or chronic obstructive pulmonary disease (COPD), or with heart disease.

When drought causes water and food shortages there can be many impacts on the health of the affected population, which may increase the risk of disease and death. Drought may have acute and chronic health effects, including malnutrition due to the decreased availability of food, including micronutrient deficiency, such as iron-deficiency anemia.

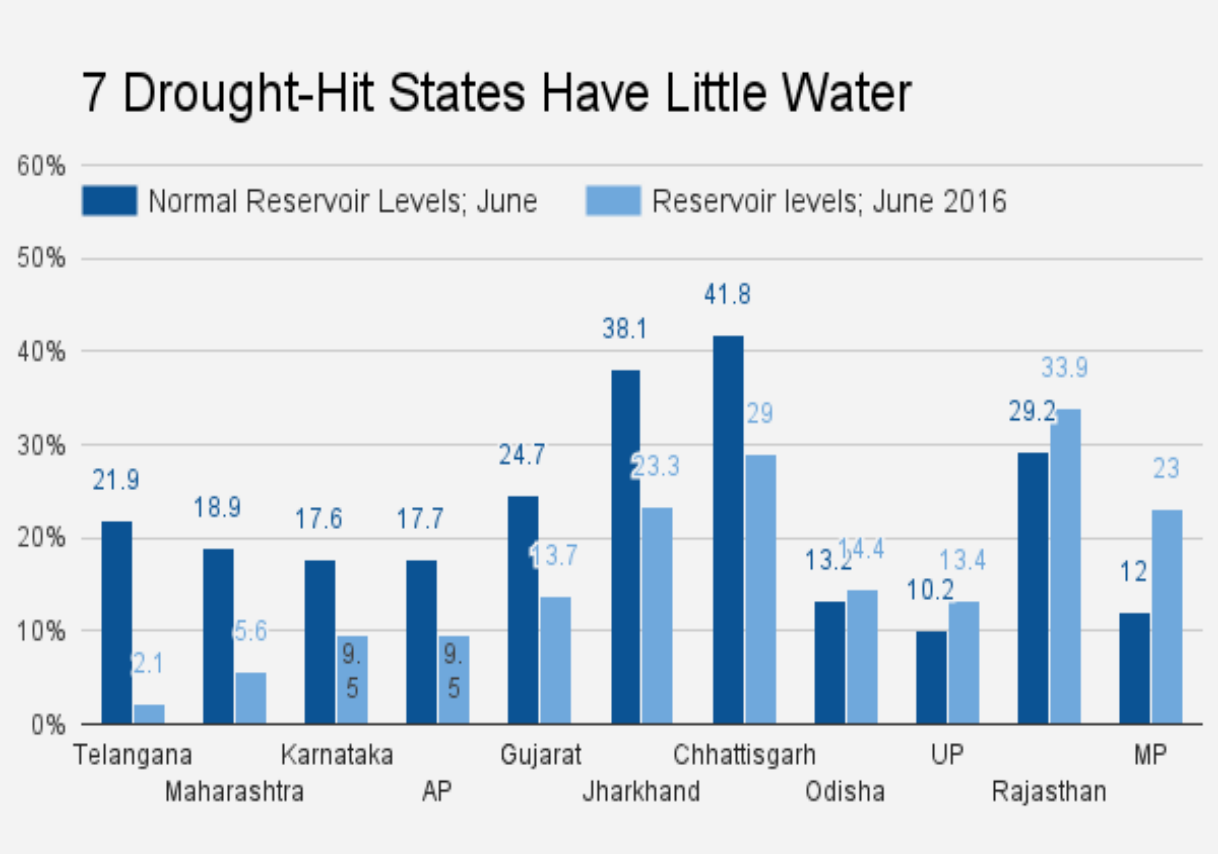

Figure 2 Drought Hit Areas of India

The above graph shows how drought has impacted the water reservoirs in India making agriculture tougher in these areas. Agriculture requires a lot of water as compared to other jobs in the area.

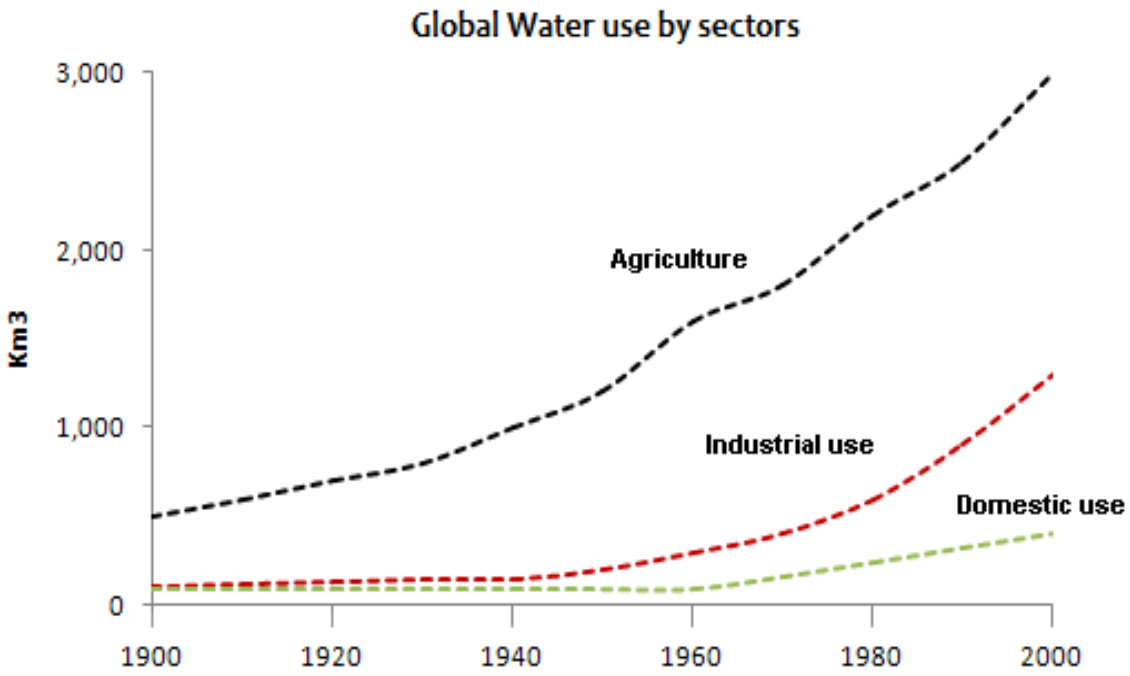

Figure 3 Total Water Usage

The graph shows how water usage has been readily increasing in agriculture. With water's increasing scarcity and increasing usage, it becomes a dire need to conserve water while irrigating the fields. With simple management practices and efficient use of water, homeowners can reduce water usage while maintaining attractive and healthy lawns and landscapes. And some of the most effective techniques for conserving water in the landscape are also the most affordable. Follow these tips to make the most of every drop.

PART I

CHAPTER 1 OVERVIEW AND LITERATURE SURVEY

OVERVIEW

Water conservation is not considered while using a standard irrigation system. Since the water is applied directly to the soil, plants are put under a lot of stress by variations in soil moisture, which reduces plant look and adversely affect it health, conclusively reducing the yield. Inadequate water control is the outcome of the system's lack of automatic control. The population expansion, which is accelerating, is the main cause of these restrictions.

As a result of the current worldwide water problem, managing water shortage has become a critical task. Countries with a lack of water resources and weak economies can see this expansion. Therefore, this is a severe issue in the agricultural sector.

Water conservation is a dire need of modern society. To conserve water and achieve the ready effects of irrigation to support higher yield from the crops, we propose the idea of an IoT aided Smart Irrigation which will work upon ESP32 Microcontroller that operate automatically by sensing the moisture content of the soil and turn ON/OFF the pump without the intervention of farmer and hence, save water. Then the data from the System shall be sent to the user through Wi-Fi, so that the user can manage and run the working of the systems from his mobile device directly.

LITERATURE SURVEY I

I. An intelligent irrigation system is proposed by Gubbi et al. in (Gao et al., 2013) that uses WSN, fuzzy control and monitoring center where the nodes work on solar energy and collect soil and crop growth information for different time periods.

II. Parameswaran et al. in (Parameswaran and Siva Prasath, 2016) have presented a smart drip irrigation system that takes input from various sensors for sensing the soil requirements, alerts the controller and updates status to the server which leads to better productivity.

III. A mobile information system is proposed by Alemu et al. in (Alemu and Nagesh, 2015). Abdullah et al. in (Abdullah et al., 2016) uses a smart agriculture system that can analyze an agricultural environment, deal with agricultural challenges like temperature, humidity, etc. and a similar intelligent farming system with sensor.

IV. A functional architecture of farm management system is also presented in (Kaloxylos et al., 2012) and (Kaloxylos et al., 2013) presents an overall vision for future internet services to revolutionize farming sector.

V. IoT based intelligent irrigation support system for smart farming applications is proposed by Neha Kailash Nawandar and Vishal Satpute (in 2019) uses uses energy compaction property of DCT where, only the most important DCT alphabets are accurately computed, while the least important ones are computed approximately.

VI. An IoT based smart irrigation management system using Machine learning and open-source technologies by Amarendra Goapa in 2018 uses open-source technology based smart system to predict the irrigation requirements of a field using the sensing of ground parameter like soil moisture, soil temperature, and environmental conditions along with the weather forecast data from the Internet.

CHAPTER 2 AGRICULTURE AND TECHNOLOGY

2.1 INTRODUCTION

The way modern farms and agricultural enterprises run differs from how they did a few decades ago, due to technological developments in the form of sensors, machinery, devices, and information technology. Robots, temperature and moisture sensors, aerial photographs, and GPS technology are all often used in modern agriculture. Farming has become more successful, productive, safe, and environmentally friendly thanks to these innovative equipment, robotic systems, and precision agriculture techniques.

2.2 IMPORTANCE OF AGRICULTURAL TECHNOLOGY

Farmers no longer must apply water, fertilizers, and pesticides uniformly across entire fields. Instead, they can use the minimum quantities needed and target specific areas, or even treat individual plants differently. Benefits include:

- Higher crop productivity
- Decreased use of water, fertilizer, and pesticides, which in turn keeps food prices down.
- Reduced impact on natural ecosystems
- Less runoff of chemicals into rivers and groundwater
- Increased worker safety

In addition, robotic technologies enable more reliable monitoring and management of natural resources, such as air and water quality. It also gives producers greater control over plant and animal production, processing, distribution, and storage, which results in:

- Greater efficiencies and lower prices
- Safer growing conditions and safer foods
- Reduced environmental and ecological impact.

Researchers have made outstanding progress towards creating a more productive—and resilient—global food system thanks to developments in data analytics. The uses of these

technologies are limitless, and these digital tools are aiding in opening whole new possibilities in plant breeding, crop protection, and other fields.

Farmers are compelled to think freely when growing crops that require a lot of water or in places where rain does not fall often. To hydrate thirsty plants, irrigation uses groundwater, surface water, and water supplied directly to fields.

Missouri farmers now get guidance on when to irrigate from smart software developed by the University of Missouri. The software analyses weather conditions based on field location, evapotranspiration estimations, and NRCS soil mapping and texture data to aid farmers in better managing moisture.

Investments in effective irrigation are essential for reducing the stress placed on the environment and the world's agricultural systems by the increased demand for food. Additionally, they result in noticeable improvements in farmers' earnings and living standards, particularly for small farmers who are the main producers to agriculture in developing nations but are often the most vulnerable and impoverished members of society. To alter the lives of small farmers and advance highly productive yet environmentally friendly agriculture, IFC is looking to collaborate with financial institutions, funders, governments, farmers' groups, and equipment manufacturers to make efficient irrigation technologies more accessible.

CHAPTER 3 PROPOSED METHODOLGY

3.1 INTORDUCTION

This IoT aided Irrigation System is based on ESP32 microcontroller. This prototype checks the amount of soil moisture content in soil. A predefined value of soil moisture is set and can be varied with crops. In case the soil moisture of the soil deviates from the specified range, the watering system is turned ON/OFF. In case of dry soil, it will activate the irrigation system, pumping water for watering the plants.

In simulation, pin2 and pin3 are used as an input pin for solenoid valve and switch, respectively. This system can be implemented on a large scale for farming purposes, which can further prove to be more helpful. Owing to prevailing conditions and water shortages, the best irrigation schedules should be decided especially in farms to conserve water.

3.2 SYSTEM USE CASE DIAGRAM

Use-case diagrams describe the high-level functions and scope of a system. These diagrams also identify the interactions between the system and its actors. The use cases and actors in

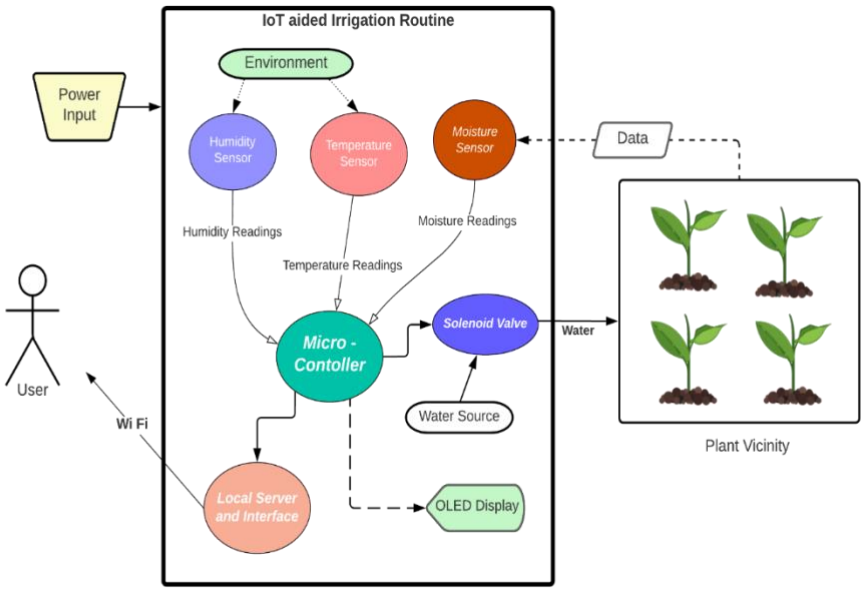

Figure 4 Use Case Diagram for the IoT aided Vegetal Irrigation Routine

use-case diagrams describe what the system does and how the actors use it, but not how the system operates internally.

The uses cases in the above diagram are as follows –

- **User** – The user's job is to turn on the system by connecting it to any power supply and water source. The user will receive the information through Wi-Fi, sent by the Micro-Controller. The user can manually operate the system using his/her mobile device as well.
- **Plant Vicinity** – It is the hospice for the subjects i.e., plants which are connected to the system using the soil moisture sensors which will extract the data from the plants. After processing the data, the system will decide and supply the water to the plant vicinity.
- **System** – It acts as the self-reliant mediator acting between the user and the plant vicinity, governing the data extracted by the sensors, processing it, and communicating it to the user and the other elements in the system. The extracted data is passed through the program which dictates the irrigation of the plant.
- **Use Cases** – Represent the specific functionalities or actions that the user can perform within the system.
    - Set Irrigation parameters: The user can set the irrigation parameters, specifying the days, time, and duration of irrigation.
    - Adjust Irrigation Settings: The user can adjust the irrigation settings, such as the watering frequency or the amount of water to be supplied.
    - Monitor Soil Moisture: The user can monitor the soil moisture levels in real-time using sensors connected to the system.
    - Receive Notifications: The user can receive notifications or alerts regarding system status, weather conditions, or any other relevant information.

3.3 ACTIVITY DIAGRAM

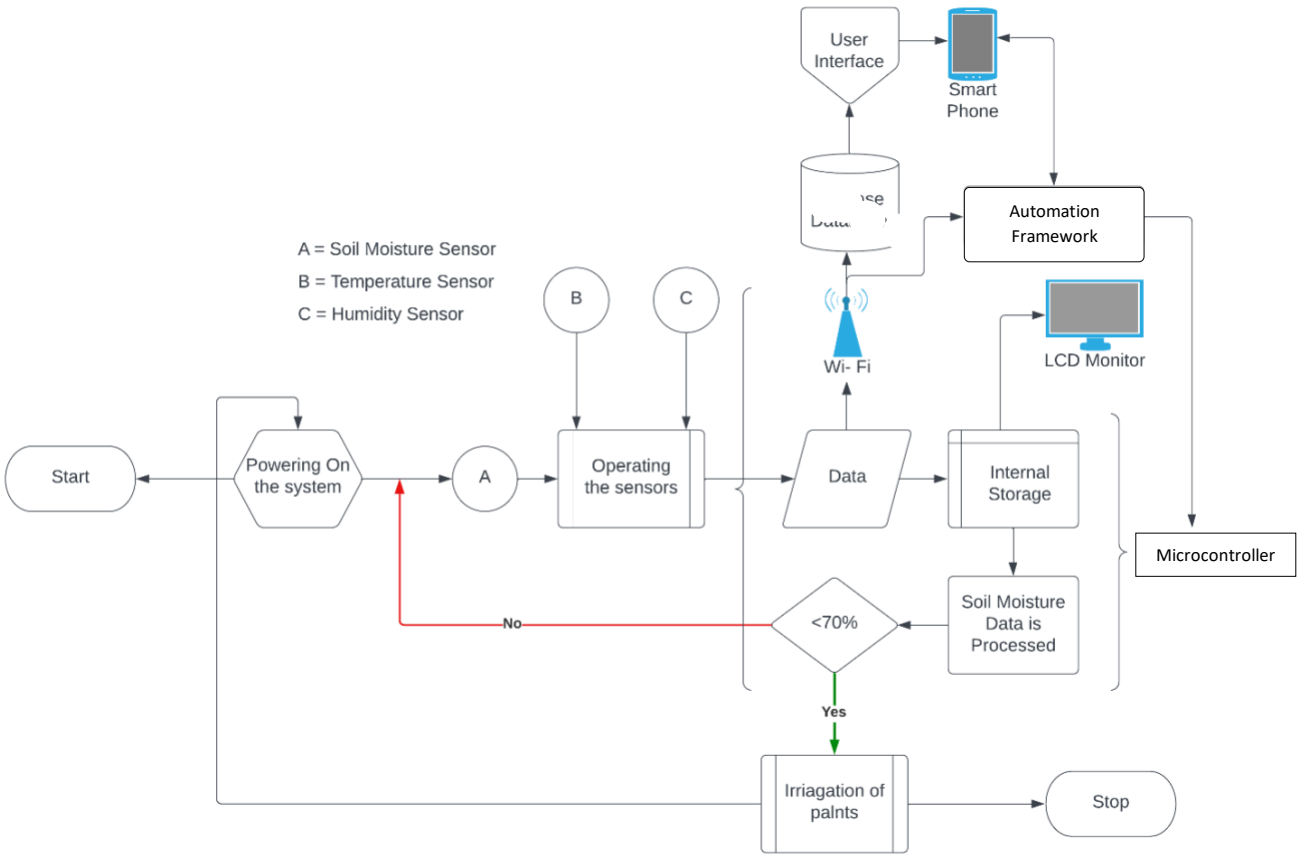

Figure 5 Activity Diagram of the System

Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system. The control flow is drawn from one operation to another. This flow can be sequential, branched, or concurrent.

This activity diagram represents the high-level flow of activities in the IoT aided Vegetal Irrigation Routine, showcasing the key steps involved in collecting data, analyzing it, determining irrigation requirements, adjusting parameters, activating the system, monitoring progress, updating status, and repeating the process.

3.4 PROCESS FLOW DIAGRAM –

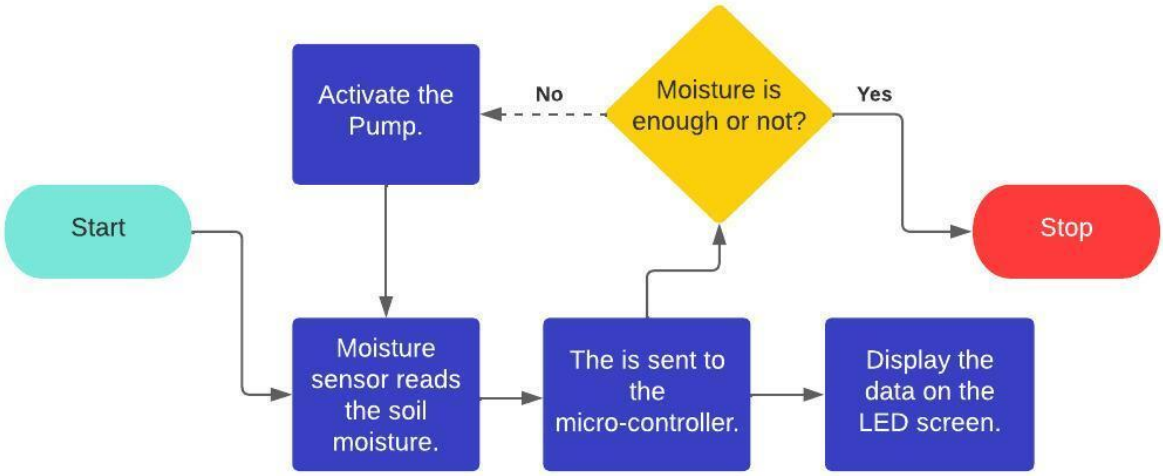

Figure 6 Process Flow Diagram

A flow diagram is a graphic representation of the physical route or flow of people, materials, paperwork, vehicles, or communication associated with a process, procedure plan, or investigation. In the second definition the meaning is limited to the representation of the physical route or flow.

The above diagram shows how the data is being manipulated by the microcontroller.

CHAPTER 4 PRELIMINARY ANALYSIS

The purpose of this preliminary analysis is to evaluate the feasibility and potential benefits of implementing an IoT aided Vegetal Irrigation Routine. This system utilizes advanced technologies such as Internet of Things (IoT), data analytics, and automation to optimize water usage and enhance the efficiency of irrigation processes in agricultural or landscaping settings.

4.1 MAIN OBJECTIVES

- Water Conservation: To reduce water wastage by delivering the right amount of water to plants based on their specific needs and environmental conditions.
- Increased Efficiency: To improve the overall efficiency of irrigation systems by automating the process and minimizing manual intervention.
- Cost Savings: To reduce operational costs associated with water usage, labor, and maintenance by optimizing irrigation schedules and minimizing water waste.

4.2 COMPONENTS OF A IOT AIDED VEGETAL IRRIGATION ROUTINE:

- Sensors: Soil moisture sensors, weather sensors, and moisture level detectors are used to collect data on soil moisture, temperature, humidity, rainfall, and other environmental parameters.
- Data Collection and Transmission: Sensor data is collected and transmitted to a central control system using wireless communication protocols such as Wi-Fi, Zigbee, or LoRaWAN.
- Data Analytics: The collected data is processed and analyzed to determine the precise water requirements of plants based on their growth stage, soil type, and weather conditions.s
- Automated Control System: An intelligent control system regulates the irrigation process, activating or deactivating the irrigation system based on the analyzed data and predefined parameters.

- Mobile/Web Application: A user-friendly interface allows farmers or gardeners to monitor and control the irrigation system remotely, set irrigation schedules, and receive notifications or alerts.

4.3 BENEFITS OF IOT AIDED VEGETAL IRRIGATION ROUTINE:

- Water Conservation: By delivering water based on actual plant needs, the system minimizes overwatering and eliminates water wastage, resulting in significant water conservation.
- Improved Plant Health: Smart irrigation ensures that plants receive the right amount of water at the right time, promoting healthier plant growth, reducing stress, and preventing diseases caused by over or under-watering.
- Cost Savings: Efficient water usage leads to reduced water bills and lower operational costs. Additionally, automated control systems save labor and minimize maintenance expenses.
- Environmental Sustainability: By conserving water resources, an IoT aided Vegetal Irrigation Routine contributes to environmental sustainability and helps mitigate the impact of drought or water scarcity.

4.4 CHALLENGES AND CONSIDERATIONS:

- Initial Investment: Implementing an IoT aided Vegetal Irrigation Routine requires an upfront investment in sensors, control systems, and infrastructure. The cost may vary depending on the scale of the operation.
- Technical Expertise: Proper installation, configuration, and maintenance of the system may require technical expertise. Training or outsourcing may be necessary for system setup and ongoing support.
- Data Security: As sensor data and control systems are connected to the internet, ensuring data security and protection against potential cyber threats becomes crucial.
- System Scalability: The system should be scalable to accommodate growing or changing irrigation needs. Expansion considerations should be made to cater to future requirements.

- Integration and Compatibility: Compatibility with existing irrigation infrastructure and integration with other farming or landscaping systems should be evaluated.

4.5 FACTORS TO CONSIDER BEFORE DEVELOPMENT OF IOT AIDED VEGETAL IRRIGATION ROUTINE

- Soil properties; soil type, drainage, water holding capacity.
- Water quality; availability, quality, quantity, crop water requirements
- Crop properties; yield potential, frost resistance, row space, harvest practices, rooting depth
- Climate requirements; humidity, temperature, precipitation
- Irrigation system properties: operating cost, and the ability to deliver and apply the amount of water needed to meet the crop's water requirement.

4.6 CURRENT IRRIGATION SYSTEMS

- Flood or furrow irrigation; entire soil surface is covered with water; it moves over the field by gravity flow.
- Sprinkler irrigation: crops are irrigated with high-pressure sprinklers set in the field; it can be solid, or hand moved.
- Drip irrigation: water is placed directly into the crop root zone from the low flow emitters, this usually also involves drip irrigation systems.
- Center Pivot irrigation: single central irrigation pipeline rotates around the pivot point. As it rotates, water sprinklers along the central pipe and irrigates crops.

CHAPTER 5 INTERNET OF THINGS

5.1 INTRODUCTION

The Internet of Things (IoT) describes the network of physical objects— “things”—that are embedded with sensors, software, and other technologies for the purpose of connecting and exchanging data with other devices and systems over the internet. These devices range from ordinary household objects to sophisticated industrial tools. With more than 7 billion connected IoT devices today, experts are expecting this number to grow to 10 billion by 2020 and 22 billion by 2025. Oracle has a network of device partners.

5.2 ARCHITECTURE

In essence, an IoT architecture is a system of numerous elements that range from sensors, protocols, actuators, to cloud services, and layers. Besides, devices and sensors the Internet of Things (IoT) architecture layers are distinguished to track the consistency of a system through protocols and gateways.

5.2.1 PERCEPTION/SENSING LAYER

The first layer of any IoT system involves “things” or endpoint devices that serve as a conduit between the physical and the digital worlds. Perception refers to the physical layer, which includes sensors and actuators that can collect, accept, and process data over the network. Sensors and actuators can be connected either wirelessly or via wired connections. The architecture does not limit the scope of its components nor their location.

5.2.2 NETWORK LAYER

Network layers provide an overview of how data is moved throughout the application. This layer contains Data Acquiring Systems (DAS) and Internet/Network gateways. A DAS performs data aggregation and conversion functions (collecting and aggregating data from sensors, then converting analog data to digital data, etc.). It is necessary to transmit and process the data collected by the sensor devices. That’s what the network layer does.

5.2.3 PROCESSING LAYER

The processing layer is the brain of the IoT ecosystem., data is analyzed, pre-processed, and stored here before being sent to the data center, where it is accessed by software applications that both monitor and manage the data as well as prepare further actions.

5.2.4 APPLICATION LAYER

User interaction takes place at the application layer, which delivers application-specific services to the user. An example might be a smart home application where users can turn on a coffee maker by tapping a button in an app or a dashboard that shows the status of the devices in a system.

5.3 APPLICATIONS

- The Internet of Things can be used in many different aspects of life, in both the private and public sectors. Thanks to IoT, people can track things like lost pets, their house's security systems, or appliance maintenance schedule.

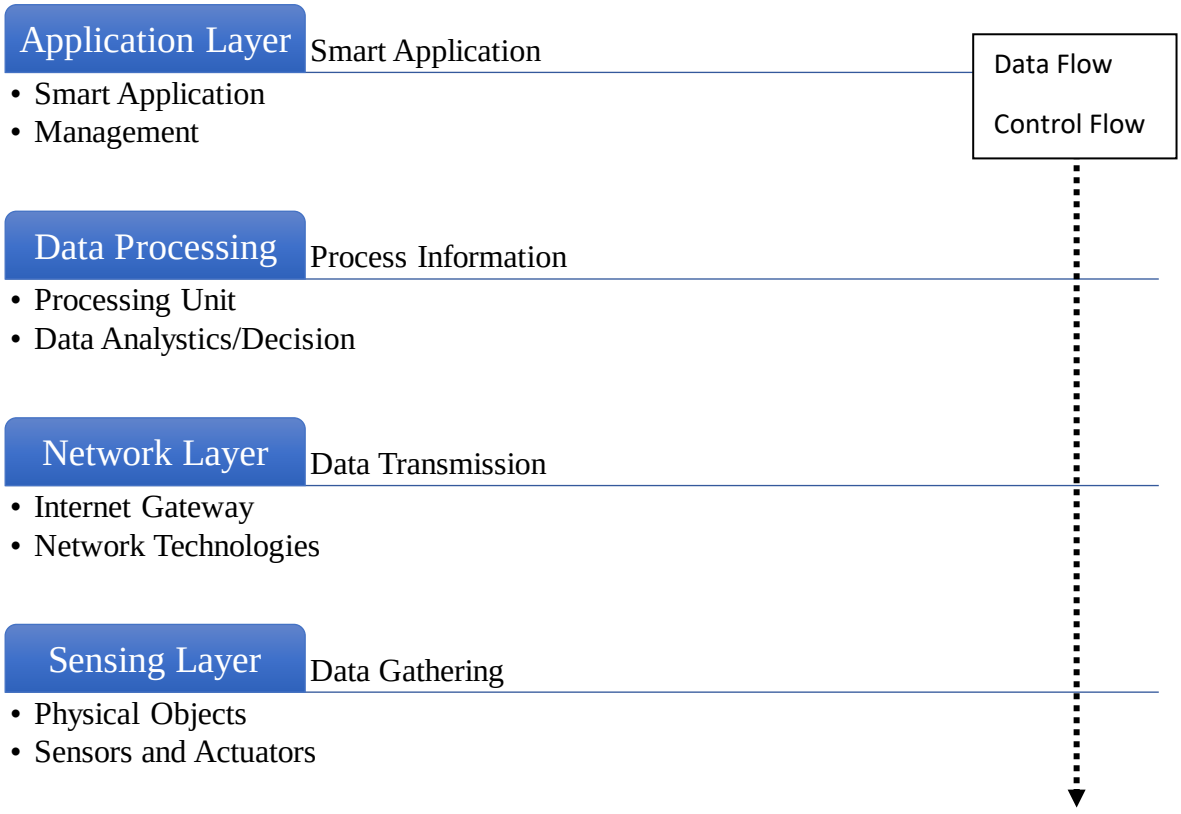

Figure 7 Staged IoT Structure

- Consumers can use the IoT to help them make restaurant reservations, monitor their exercise progress and overall health, and receive coupons for a store only by virtue of walking by the business in question.
- Businesses can use IoT to monitor supply chains, track customers' spending habits as well collect their feedback, monitor, and maintain inventory levels, and engage in predictive maintenance of their machines and devices.
- The IoT also proves helpful in ITIL, which is a set of IT service management, an important detail, since IT departments are called on to do more and more in a world that's getting increasingly digital, with more reliance on wireless networks.
- Blockchain, which is being increasingly used as a more efficient and secure method of transaction and data processing, is a natural beneficiary of IoT technology. We can expect to see IoT and Blockchain coming together more often in the future.

5.4 ENABLING TECHNOLOGIES

5.4.1 SHORT-RANGE WIRELESS

- Bluetooth mesh networking – Specification providing a mesh networking variant to Bluetooth low energy (BLE) with an increased number of nodes and standardized application layer (Models).
- Light-Fidelity (Li-Fi) – Wireless communication technology similar to the Wi-Fi standard but using visible light communication for increased bandwidth.
- Near-field communication (NFC) – Communication protocols enabling two electronic devices to communicate within a 4 cm range.
- Radio-frequency identification (RFID) – Technology using electromagnetic fields to read data stored in tags embedded in other items.
- Wi-Fi – Technology for local area networking based on the IEEE 802.11 standard, where devices may communicate through a shared access point or directly between individual devices.
- Zigbee – Communication protocols for personal area networking based on the IEEE 802.15.4 standard, providing low power consumption, low data rate, low cost, and high throughput.

- Z-Wave – Wireless communications protocol used primarily for home automation and security applications.

5.4.2 MEDIUM-RANGE WIRELESS

- LTE-Advanced – High-speed communication specification for mobile networks. Provides enhancements to the LTE standard with extended coverage, higher throughput, and lower latency.

5.4.3 LONG-RANGE WIRELESS

- Low-power wide-area networking (LPWAN) – Wireless networks are designed to allow long-range communication at a low data rate, reducing power and cost for transmission. Available LPWAN technologies and protocols: LoRaWan, Sigfox, NB-IoT, Weightless, RPMA, MIoTy.
- Very small aperture terminal (VSAT) – Satellite communication technology using small dish antennas for narrowband and broadband data.

5.5 DIFFERENCE BETWEEN IOT AND M2M

The Internet of things (IoT) is the network of physical devices, vehicles, home appliances, and others embedded with electronics, software, sensors, actuators, and connectivity which enables these things to connect and exchange data, creating opportunities for more immediate integration between the physical world and computer-based systems. Regarding M2M (machine to machine) communication, there are several differences. The M2M communications are used to connect machines to machines within the same cellular network. The IoT is used to connect machines to machines across different cellular networks.

CHAPTER 6 AUTOMATION

6.1 INTRODUCTION

Automation is the creation and application of technologies to produce and deliver goods and services with minimal human intervention. The implementation of automation technologies, techniques and processes improve the efficiency, reliability, and/or speed of many tasks that were previously performed by humans. Automation is being used in several areas such as manufacturing, transport, utilities, defense, facilities, operations and lately, information technology.

Usually, automation is employed to minimize labor or to substitute humans in the most menial or repetitive tasks. Automation is present in virtually all verticals and niches, although it's more prevalent in manufacturing, utilities, transportation, and security.

For example, most manufacturing plants make use of some automated processes in the form of robotic assembly lines. Human input is required only to define the processes and supervise them, while the assembling of the various components is left to the machines, which automatically convert raw materials into finished goods.

In the technology domain, the impact of automation is increasing rapidly, both in the software/hardware and machine layer. The implementation of new artificial intelligence (AI) and machine learning (ML) technologies is currently skyrocketing the evolution of this field.

6.2 EXAMPLES OF AUTOMATION

In the information technology domain, a software script can test a software product and produce a report. There are also various software tools available in the market which can generate code for an application. The users only need to configure the tool and define the process.

Advanced business intelligence in applications is another new form of high-quality automation. In other industries, automation has greatly improved productivity in the last decades, saving time and cutting costs.

From the simplest to the most complex application, automation is present in many forms in our everyday life. Common examples include household thermostats controlling boilers, the earliest automatic telephone switchboards, electronic navigation systems, or the most advanced algorithms behind self-driving cars.

- Home automation – uses a combination of hardware and software technologies that enable control and management over appliances and devices within a home.
- Network automation – the process of automating the configuration, management, and operations of a computer network.
- Office automation – involves using computers and software to digitize, store, process and communicate most routine tasks and processes in a standard office.
- Automated website testing – streamlines and standardizes website testing parameters for configuration changes that occur during the development phase.
- Data center automation – enables the bulk of the data center operations to be performed by software programs. Includes automated system operations, also known as lights-out operations.
- Test automation – software code goes through quality assurance (QA) testing automatically by scripts and other automation tools.

The most advanced level of automation is intelligent automation. It combines automation with artificial intelligence (AI) and machine learning (ML) capabilities. This means that machines that automation can continuously "learn" and enable better decision making and actions based on data from past situations they have encountered and analyzed. For example, in customer service, virtual assistants powered by AI/ML can reduce costs while empowering both customers and human agents, creating an optimal customer service experience. AIOps and digital workers are examples of intelligent automation.

CHAPTER 7 AUTOMATED INTERNET OF THINGS

7.1 INTORDUCTION

The internet of things, or IoT, is a system of interconnected devices and sensors that collect and share data. One of the many advantages of IoT is that it can help with automation. Automation can increase efficiency, productivity, and accuracy while reducing costs and up-time.

For example, let's say you own a manufacturing company that produces widgets. Traditionally, you would have needed workers to assemble widgets by hand. With automation, robots can do the work for you. This not only reduces the amount of time it takes to produce these widgets, it also cuts down on labor costs and allows your manufacturing process to run 24/7.

IoT can also be used to automate tasks in your personal life. For example, sensors can track when you leave work at the end of the day, automatically turning your lights and heating on ensuring that your smart home is the correct temperature for your arrival.

There are a wide range of opportunities for businesses to benefit from the implementation of IoT within their business, from improving supply chains to simply enabling communication between two or more remote devices. IoT technology can be as simple to implement as your standard cloud computing solutions.

7.2 BENEFITS

7.2.1 IOT AND ACCURACY

For many businesses, reducing errors is a main priority, human error is something that is often overlooked or discounted when refining processes and controlling costs. By implementing industrial automation capabilities to processing or manufacturing machines, and allowing them to control themselves automatically, fewer members of staff are needed to complete certain tasks, significantly reducing the opportunity for human error.

7.2.2 INCREASE UP-TIME WITH THE IOT.

Another advantage of automating your business processes using IoT is an increase in up-time. As we all know, your human workers cannot work around the clock, they come into work at a specific time and leave at another. This means that your business is only open for a certain number of hours per day (unless your staff's shifts overlap). Machines on the other hand, aren't restricted by time. Automating your business processes via the Internet of Things will allow them to run 24/7.

7.2.3 OPTIMIZE MAINTENANCE.

Preventative maintenance is key to running your machines successfully in the long term. IoT-enabled machines will be able to complete preventative maintenance or order servicing/parts on their own before they are even required, rather than a business having to wait for an issue to occur and then having a 4-6 week wait on a part leaving them out of action.

7.2.4 IOT ANALYSIS AND MONITORING

The beauty of IoT smart devices isn't necessarily the improved performance of business processes. In modern-day business, analytics and logistics are becoming more and more important as technology develops. Understanding why a problem has occurred or identifying areas of improvement is getting much easier because of this.

CHAPTER 8 HARDWARE ARCHITECTURE

The hardware architecture of the system can vary depending on the specific implementation and requirements. However, here is a general outline of the components that typically make up the hardware architecture:

8.1 SENSORS:

- Soil Moisture Sensors: These sensors measure the moisture content in the soil.
- Weather Sensors: Sensors that collect data on temperature, humidity, wind speed, and rainfall.
- Light Sensors: Optional sensors to measure the amount of sunlight received by plants.

8.2 CONTROLLER:

- Microcontroller or Single-Board Computer (SBC): This component processes the sensor data, performs calculations, and controls the overall irrigation system.
- Input/Output (I/O) Interface: Provides connectivity for the sensors and actuators.

8.3 ACTUATORS:

- Valves: These control the flow of water from the water source to the irrigation system.
- Pumps: If needed, pumps can be used to pressurize water for efficient irrigation.

8.4 CONNECTIVITY:

- Wireless Communication Module: Enables communication between the irrigation system and external devices or networks. Common options include Wi-Fi, Zigbee, or LoRaWAN.
- Internet Connection: Allows remote monitoring and control of the system through a web or mobile application.

8.5 POWER SUPPLY:

- Power Source: Typically includes an electrical connection to the grid or battery power for remote areas.
- Power Management: Circuitry to regulate and distribute power to various components of the system.

8.6 USER INTERFACE:

- Web/Mobile Application: A user-friendly interface for configuring and monitoring the irrigation system remotely.
- Display: In some cases, a local display may be included for on-site monitoring and configuration.

It is important to note that the hardware architecture can be tailored based on the specific needs of the application, scale of the system, and available resources. Additionally, advancements in technology and integration with other smart devices or systems can further enhance the capabilities of the IoT Aided Vegetal Irrigation System.

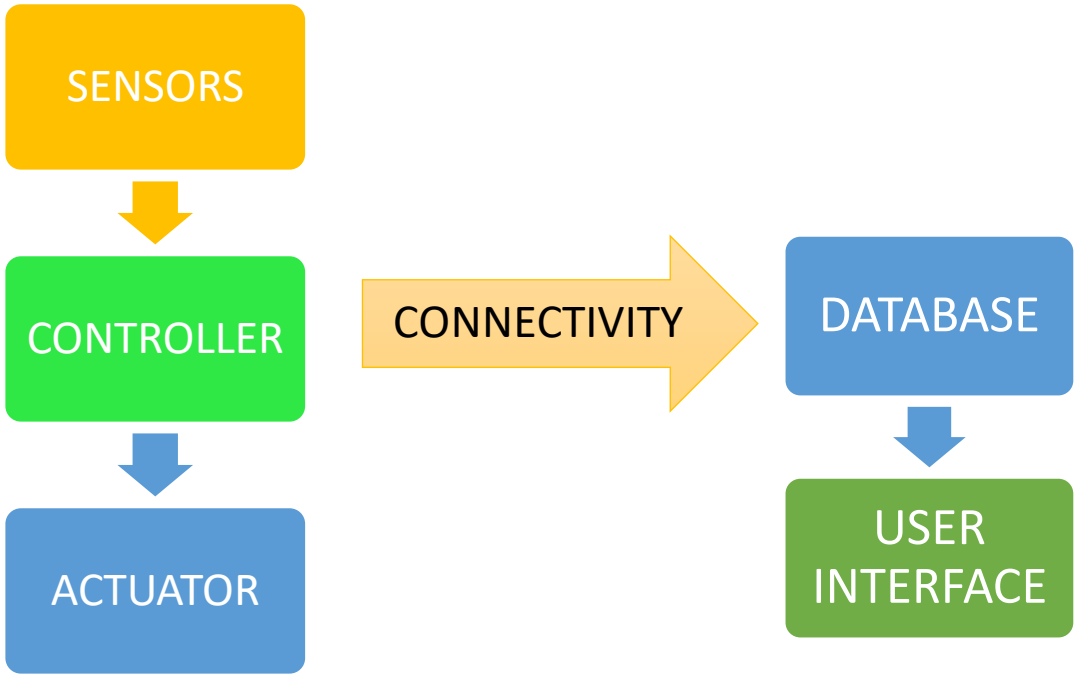

Figure 8 Hardware Architecture

CHAPTER 9 SOFTWARE AND HARDWARE REQUIREMENTS

To implement the model, you will need a combination of hardware and software components. In terms of software requirements, here are some key components typically involved in the model:

9.1 SOFTWARE REQUIREMENT

9.1.1 Microcontroller Programming

You will need to write code to control and manage the irrigation system using a microcontroller, such as Arduino, ESP32, or Raspberry Pi. Depending on the microcontroller you choose, you may need to use specific programming languages like Arduino IDE (C++), Python, or others.

9.1.2 Sensor Data Processing

The models often utilize various sensors, such as soil moisture sensors, temperature sensors, humidity sensors, or rain sensors. You will need to process the data received from these sensors to make decisions regarding irrigation scheduling and water flow control.

9.1.3 Communication Protocols

If you plan to integrate the model with other devices or platforms, you may need to implement communication protocols. This can include protocols like Wi-Fi, Bluetooth, MQTT, or HTTP to connect with other devices, cloud platforms, or mobile applications.

9.1.4 User Interface

You might want to create a user interface to monitor and control the irrigation system. This can be in the form of a web-based interface, a mobile application, or a dedicated display connected to the system. The software for the user interface will depend on the platform you choose, such as HTML/CSS/JavaScript for web interfaces or mobile app development frameworks like React Native or Flutter.

9.1.5 Decision Making and Automation

The models often include algorithms or logic to make decisions based on sensor data and predefined rules. These algorithms can determine when and how much water to supply based on factors like soil moisture levels, weather forecasts, or plant requirements. You will need to implement this decision-making logic within your software.

Data Storage and Analysis: If you want to analyze historical data or monitor the performance of your irrigation system, you may need to store data in a database. You can use database technologies like MySQL, MongoDB, or cloud-based storage solutions for this purpose. Additionally, you may employ data analysis techniques or visualization tools to gain insights from the collected data.

It's important to note that the specific software requirements can vary based on the complexity and features of the model. The above list provides a general overview of the common software components involved.

9.2 SENSORS

9.2.1 SOIL MOISTURE SENSOR –

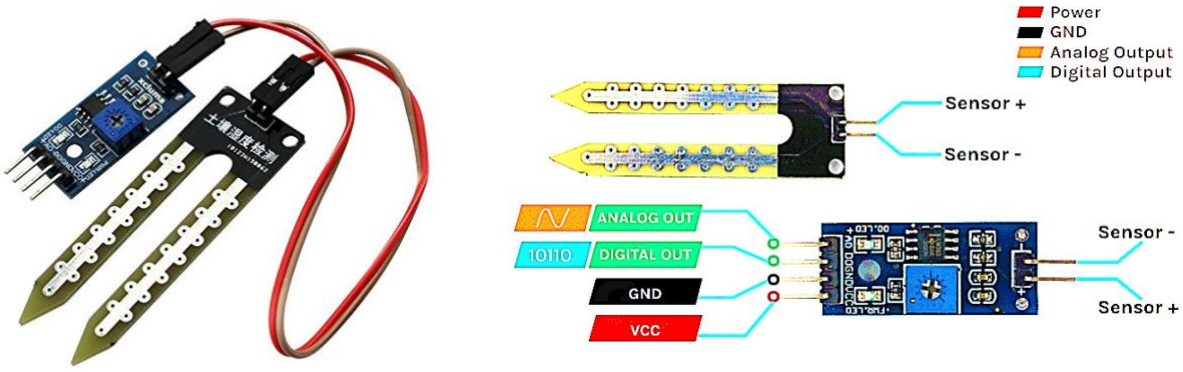

Figure 9 Soil Moisture Sensor and Pinout

A resistive soil moisture sensor works by using the relationship between electrical resistance and water content to gauge the moisture levels of the soil. You'll observe these sensors to possess two exposed probes that are inserted directly into the soil sample.

- An electrical current is sent from one probe to the other, which allows the sensor to measure the resistance of the soil between them.
- When the water content in the soil is high, it has a higher electrical conductivity (water is a good conductor of electricity!). Hence, a lower resistance reading is obtained which indicates high soil moisture.
- When the water content in the soil is low, it has poorer electrical conductivity. Hence, a higher resistance reading is obtained, which indicates low soil moisture.

Why are we using resistive soil moisture sensors?

A common gripe of resistive soil moisture sensors is that they are susceptible to corrosion over time, since the electrical current sent between its probes can cause electrolysis. In contrast, the capacitive soil moisture sensor does not have exposed electrodes and is comparatively corrosion-free. However, they are in general more costly.

- Infrequent Soil Moisture Readings – The corrosion will not be as severe.
- Cost is a Concern.

9.2.2 TEMPERATURE AND HUMIDITY SENSOR –

- DHT11 is a low-cost digital sensor for sensing temperature and humidity. This sensor can be easily interfaced with any micro-controller such as Arduino, Raspberry Pi etc.... to measure humidity and temperature instantaneously.

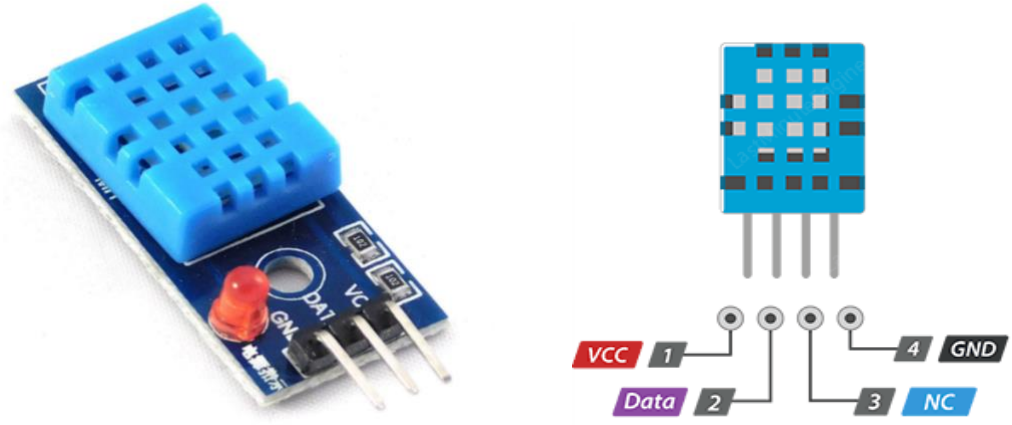

Figure 10 DHT 11 Sensor and Pinout

- DHT11 humidity and temperature sensor is available as a sensor and as a module. The difference between this sensor and module is the pull-up resistor and a power-on LED. DHT11 is a relative humidity sensor. To measure the surrounding air this sensor uses a thermistor and a capacitive humidity sensor.

Working Principle of DHT11 Sensor

DHT11 sensor consists of a capacitive humidity sensing element and a thermistor for sensing temperature. The humidity sensing capacitor has two electrodes with a moisture holding substrate as a dielectric between them. Change in the capacitance value occurs with the change in humidity levels. The IC measure, process this changed resistance values and change them into digital form.

For measuring temperature this sensor uses a Negative Temperature coefficient thermistor, which causes a decrease in its resistance value with increase in temperature. To get larger resistance value even for the smallest change in temperature, this sensor is usually made up of semiconductor ceramics or polymers.

The temperature range of DHT11 is from 0 to 50 degree Celsius with a 2-degree accuracy. The humidity range of this sensor is from 20 to 80% with 5% accuracy. The sampling rate of this sensor is 1Hz.i.e., it gives one reading for every second. DHT11 is small with an operating voltage from 3 to 5 volts. The maximum current used while measuring is 2.5mA.

9.3 MICRO CONTROLLER

The ESP32 is a popular microcontroller and system-on-chip (SoC) module widely used in various IoT (Internet of Things) applications. It is developed by Espressif Systems and is a successor to the ESP8266 module. The ESP32 offers a range of features and capabilities, making it suitable for a wide range of projects. Here's a description of the ESP32 module:

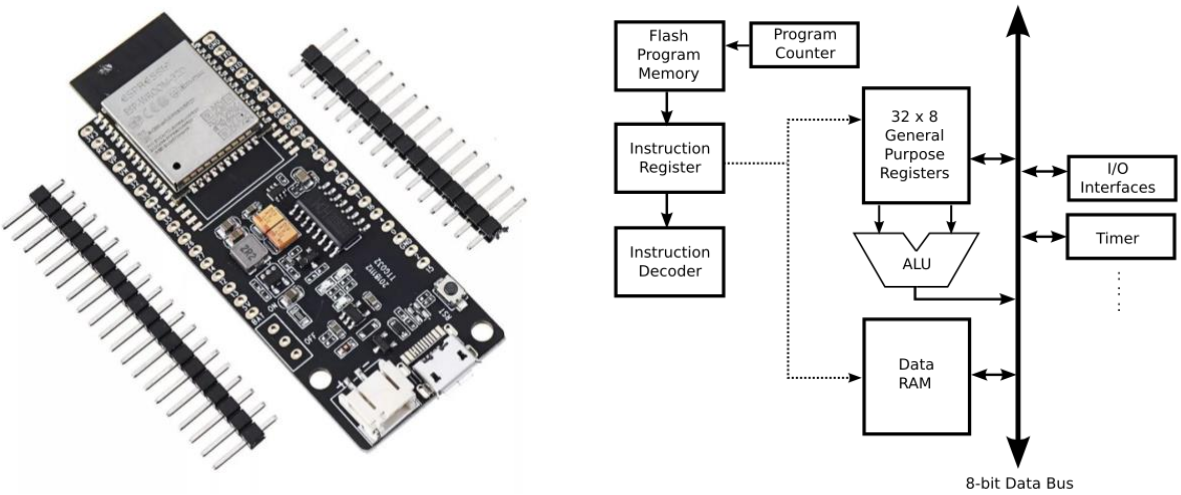

Figure 11 Microcontroller and its Architecture

9.3.1 Microcontroller Core:

- The ESP32 module is powered by a dual-core Tensilica LX6 microcontroller, running at up to 240 MHz clock speed.
- It supports both 32-bit and 16-bit instructions, providing high performance for various applications.

9.3.2 Wi-Fi and Bluetooth Connectivity:

- One of the key features of the ESP32 is its built-in Wi-Fi and Bluetooth capabilities.
- It supports Wi-Fi 802.11 b/g/n standards, providing reliable wireless connectivity for IoT applications.
- The module also includes Bluetooth 4.2 and BLE (Bluetooth Low Energy) support, enabling communication with other devices and peripherals.

9.3.3 Memory and Storage:

- The ESP32 module typically comes with a generous number of memory and storage options.
- It includes up to 520 KB of SRAM for program execution and data storage.
- Additionally, it can be equipped with up to 4 MB of flash memory for firmware storage.

9.3.4 GPIO and Peripheral Interfaces:

- The module provides a wide range of general-purpose input/output (GPIO) pins, allowing for easy interfacing with external sensors, actuators, and other devices.
- It supports various peripheral interfaces, including UART, SPI, I2C, I2S, and PWM, facilitating communication with a wide range of external devices.

9.3.5 Development Environment and Programming:

- The ESP32 module is well-supported by an extensive development ecosystem.
- It can be programmed using popular programming languages and frameworks like Arduino IDE, MicroPython, or Espressif's native ESP-IDF (Espressif IoT Development Framework).
- Various libraries and resources are available to simplify the development process and enable rapid prototyping.

The ESP32 module's combination of processing power, wireless connectivity, and rich feature set makes it a versatile and popular choice for IoT applications, including home automation, industrial monitoring, wearable devices, robotics, and more. Its widespread adoption and active community support contribute to its popularity among developers and enthusiasts.

9.4 ACTUATORS

A solenoid valve is an electrically controlled valve. The valve features a solenoid, which is an electric coil with a movable ferromagnetic core (plunger) in its center. In the rest position, the plunger closes off a small orifice. An electric current through the coil creates a magnetic field. The magnetic field exerts an upwards force on the plunger opening the orifice. This is the basic principle that is used to open and close solenoid valves. The features are as follows-

Figure 12 Solenoid Valve and Its Mechanism

- ❖ Clean liquids/gases only: Solenoid valves are designed to be used with clean liquids and gases.
- ❖ Precise flow control: Accurate fluid/gas regulation, ideal for sensitive processes in medical devices and manufacturing.
- ❖ Fast response time: Rapid opening/closing actions, vital for safety applications and swift reaction to hazards.
- ❖ Long service life: Durable, reliable performance reduces maintenance needs and withstands demanding usage.

9.5 USER INTERFACE

0/9” OLED Display – This is a general OLED display Module, 0.91inch diagonal, 128x32 pixels, with embedded controller, communicating via I2C interface.

- ❖ 0.91inch small form factor

- 128x32 high resolution
- The I2C interface requires only two signal pins.
- Comes with development resources and manual (examples for Raspberry Pi/Jetson Nano/Arduino/STM32)

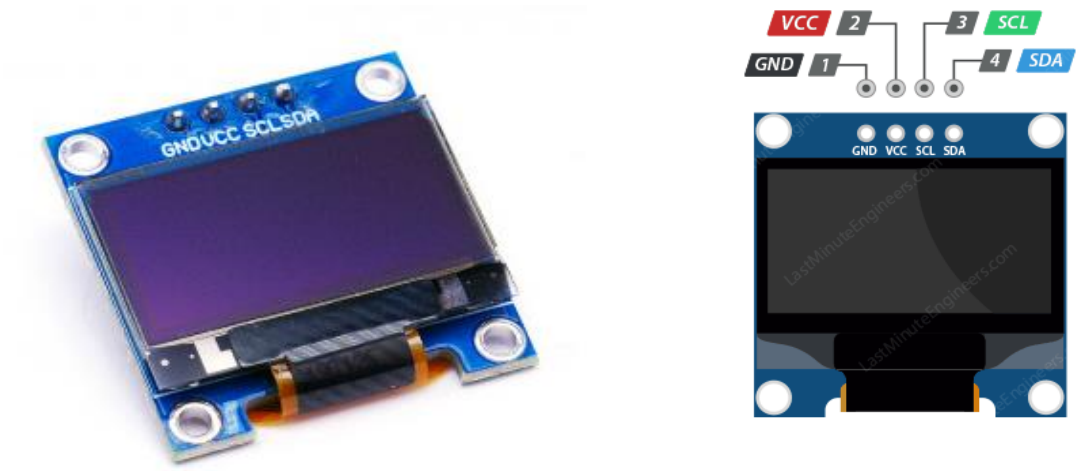

Figure 13 0.9" OLED Screen and Pinout

9.6 OTHER COMPONENTS

9.6.1 CAPACITORS –

Figure 14 Capacitors 100uF, 330nF, 100nF

It is a device for storing electrical energy, consisting of two conductors in proximity and insulated from each other. A simple example of such a storage device is the parallel-plate capacitor. If positive charges with total charge +Q are deposited on one of the conductors and an equal amount of negative charge −Q is deposited on the second conductor, the capacitor is said to have a charge Q.

The capacitors used in the system are of 100uF, 100nF and 330nF.

9.6.2 RESISTORS –

A resistor is an electrical component that limits or regulates the flow of electrical current in an electronic circuit. Resistors can also be used to provide a specific voltage for an active device such as a transistor.

Figure 15 Resistance

All other factors being equal, in a direct-current (DC) circuit, the current through a resistor is inversely proportional to its resistance, and directly proportional to the voltage across it. This is the well-known Ohm's Law. In alternating-current (AC) circuits, this rule also applies if the resistor does not contain inductance or capacitance.

There are 6 resistors used in the system of $1 * 10^2 \pm 0\%$ Ω.

9.6.3 TRANSISTOR –

A transistor is a miniature semiconductor that regulates or controls current or voltage flow in addition amplifying and generating these electrical signals and acting as a switch/gate for

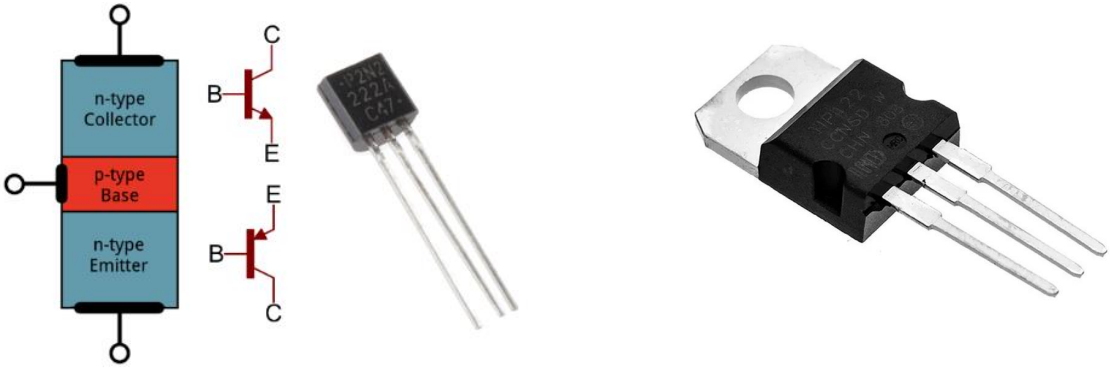

Figure 16 nPn Transistors - BCS 547 and TIP122

them. Typically, transistors consist of three layers, or terminals, of a semiconductor material, each of which can carry a current. The transistors used in the system are BC547 and TIP122.

9.6.4 VOLTAGE REGULATOR –

The function of a voltage regulator is to maintain a constant DC voltage at the output irrespective of voltage fluctuations at the input and (or) variations in the load current. In other

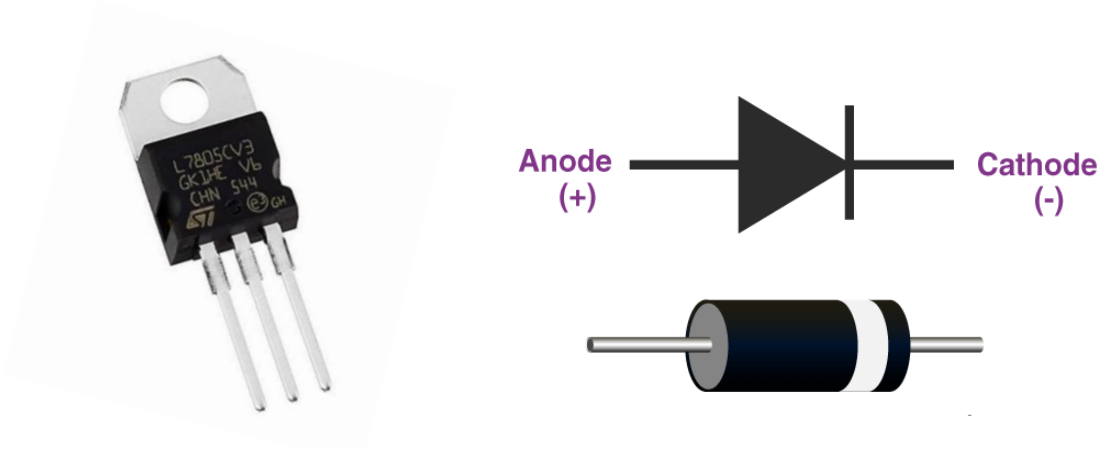

Figure 17 Voltage Regulator (7805) and Diode (1N007)

words, voltage regulator produces a regulated DC output voltage. Voltage regulators are also available in Integrated Circuits (IC) forms. These are called voltage regulator ICs.

9.6.5 DIODE –

A diode is a two-terminal electronic component that conducts current primarily in one direction (asymmetric conductance). It has low (ideally zero) resistance in one direction, and high (ideally infinite) resistance in the other. Diodes can be used as rectifiers, signal limiters, voltage regulators, switches, signal modulators, signal mixers, signal demodulators, and oscillators. The fundamental property of a diode is its tendency to conduct electric current in only one direction.

The diode used in the system is 1N007.

9.6.6 PUSH BUTTON –

A push button switch is a mechanical device used to control an electrical circuit in which the operator manually presses a button to actuate an internal switching mechanism.

Figure 18 Push Button, LED Light, General PCB

9.6.7 LED BULB –

Super bright 10mm LEDs are extremely bright with a wide beam angle. They're good for use in your projects, illuminations, head lamps, spotlights, car lighting, models or anywhere where you need low power, high intensity reliable lighting or indication.

9.6.8 PCB PLATE –

The PCB plate can be described as a unique metal plate that is fastened over the keyboard's PCB. This plate provides additional support. Additionally, a keyboard's keys are not installed directly on the Circuit board itself but rather on a PCB plate. Additionally, the keyboard's PCB plate stops it from flexing. The keyboard may be made stronger and more stable thanks to this plate. Larger keyboards typically use PCB plates. This can be attributed to the fact that it provides additional help. Additionally, the additional size may cause the keyboard to flex more. However, the PCB plate is absent from some keyboards. Nowadays, keyboards come with a plate.

9.6.9 DC POWER SOURCE –

A DC power supply is a type of power supply that gives direct current (DC) voltage to power a device. Because DC power supply is commonly used on an engineer ‘s or technician ‘s bench for a ton of power tests, they are also often called a "bench power supply." A DC voltage with typical values, 5V, 12V, 24V, or 48V, can also be accepted by a DC power supply as an input. The output voltage can also be generated; this ranges from less than a volt to >1000 volts DC.

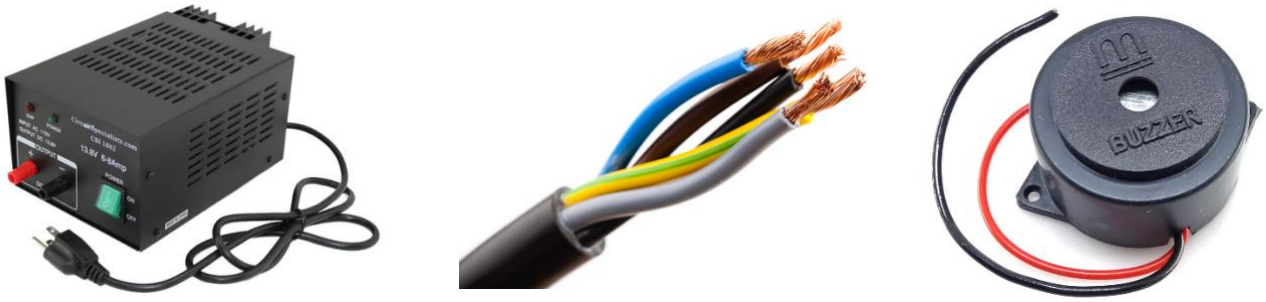

Figure 19 DC Power Supply, Multicore wire, Buzzer

9.6.10 CONNECTING WIRES –

A connecting wire allows travels the electric current from one point to another point without resistivity. Resistance of connecting wire should always be near zero. Copper wires have low resistance and are therefore suitable for low resistance.

9.6.11 BUZZER –

A buzzer or beeper is an audio signaling device, which may be mechanical, electromechanical, or piezoelectric (piezo for short). Typical uses of buzzers and beepers include alarm devices, timers, and confirmation of user input such as a mouse click or keystroke.

CHAPTER 10 DETAILED OVERVIEW OF ESP32

10.1 INTRODUCTION TO ESP 32

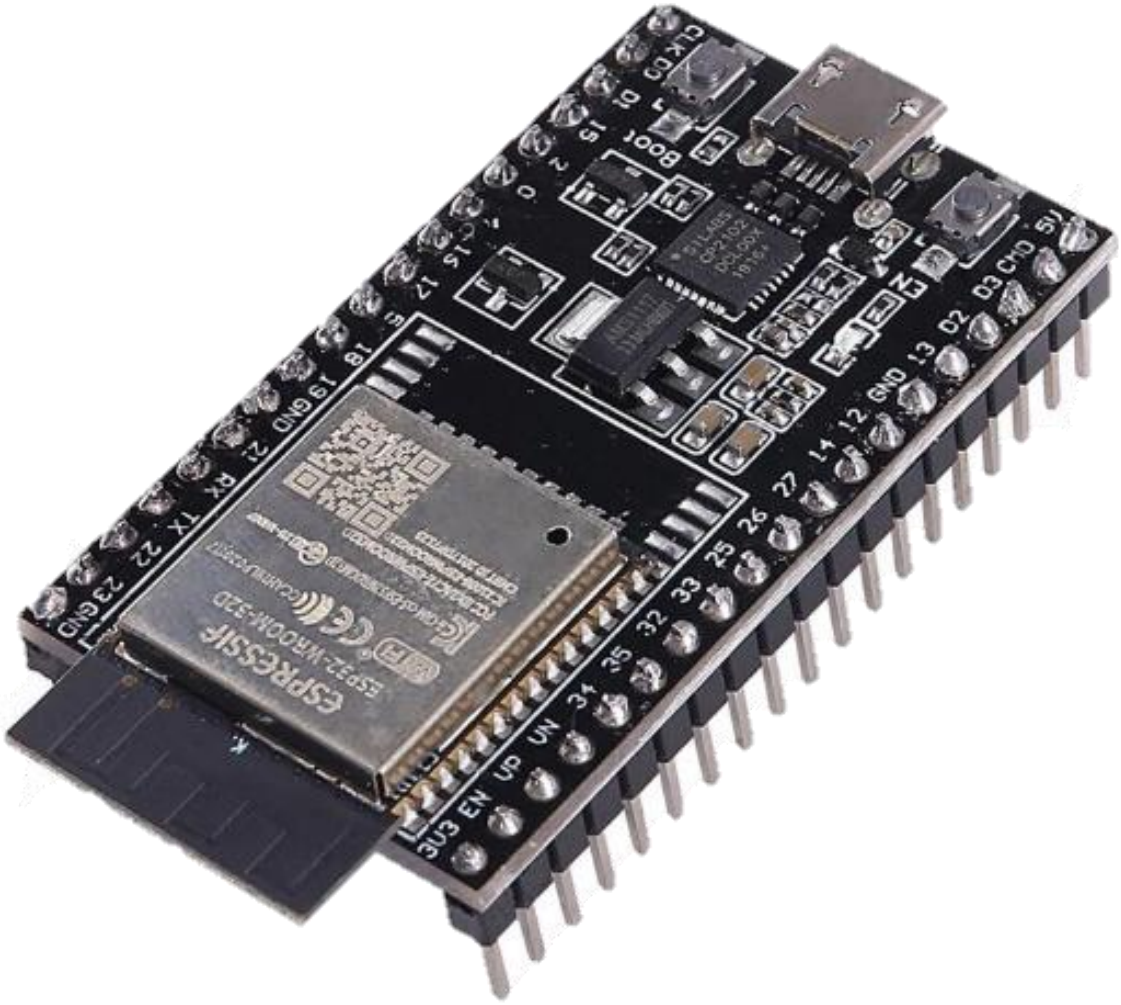

Figure 20 ESP 32 WROOM 32

ESP32 is a single 2.4 GHz Wi-Fi-and-Bluetooth combo chip designed with the TSMC low-power 40 nm technology. It is designed to achieve the best power and RF performance, showing robustness, versatility, and reliability in a wide variety of applications and power scenarios.

The ESP32 series of chips includes ESP32-D0WD-V3, ESP32-D0WDR2-V3, ESP32-U4WDH, ESP32-S0WD (NRND), ESP32-D0WDQ6-V3 (NRND), ESP32-D0WD (NRND), and ESP32-D0WDQ6 (NRND), among which,

- ESP32-S0WD (NRND), ESP32-D0WD (NRND), and ESP32-D0WDQ6 (NRND) are based on chip revisionv1 or chip revision v1.1.
- ESP32-D0WD-V3, ESP32-D0WDR2-V3, ESP32-U4WDH, and ESP32-D0WDQ6-V3 (NRND) are based on chip revision v3.0 or chip revision v3.1.

For details on part numbers and ordering information, please refer to Section 7. For details on chip revisions, please refer to ESP32 Chip Revision v3.0 User Guide and ESP32 Series SoC Errata.

10.2 FEATURED SOLUTIONS

Ultra-low Power Solution - ESP32 is designed for mobile, wearable electronics, and Internet-of-Things (IoT) applications. It features all the state-of-the-art characteristics of low-power chips, including fine-grained clock gating, multiple power modes, and dynamic power scaling. For instance, in a low-power IoT sensor hub application scenario, ESP32 is woken up periodically only when a specified condition is detected. Low-duty cycle is used to minimize the amount of energy that the chip expends. The output of the power amplifier is also adjustable, thus contributing to an optimal trade-off between communication range, data rate and power consumption.

Complete Integration Solution - ESP32 is a highly integrated solution for Wi-Fi-and-Bluetooth IoT applications, with around 20 external components. ESP32 integrates an antenna switch, RF balun, power amplifier, low noise receives amplifier, filters, and power management modules. As such, the entire solution occupies the minimal Printed Circuit Board (PCB) area.

10.3 WI-FI KEY FEATURES

- 802.11b/g/n
- 802.11n (2.4 GHz), 150 Mbps
- TX/RX A-MPDU, RX A-MSDU
- Immediate Block ACK
- Defragmentation
- Automatic Beacon monitoring
- 4 × virtual Wi-Fi interfaces
- Antenna diversity

10.4 MCU AND ADVANCED FEATURES

- Xtensa® single-/dual-core 32-bit LX6 microprocessor(s)
- 1 core at 240 MHz: 504.85 CoreMark; 2.10 CoreMark/MHz
- 2 cores at 240 MHz: 994.26 CoreMark; 4.14 CoreMark/MHz
- 448 KB ROM
- 520 KB SRAM
- 16 KB SRAM in RTC

10.5 ADVANCED PERIPHERALS

- 12-bit SAR ADC up to 18 channels
- 2 × 8-bit DAC
- 10 × touch sensors
- 4 × SPI
- 2 × I2S
- 2 × I2C
- 3 × UART
- 1 host (SD/eMMC/SDIO)
- 1 slave (SDIO/SPI)
- 34 × programmable GPIOs

10.6 SECURITY

- Secure boot
- Flash encryption
- 1024-bit OTP, up to 768-bit for customers
- Cryptographic hardware acceleration:

10.7 APPLICATIONS (A NON-EXHAUSTIVE LIST)

- Generic Low-power IoT Sensor Hub
- Generic Low-power IoT Data Loggers
- Cameras for Video Streaming
- Speech Recognition

10.8 BLOCK DIAGRAM

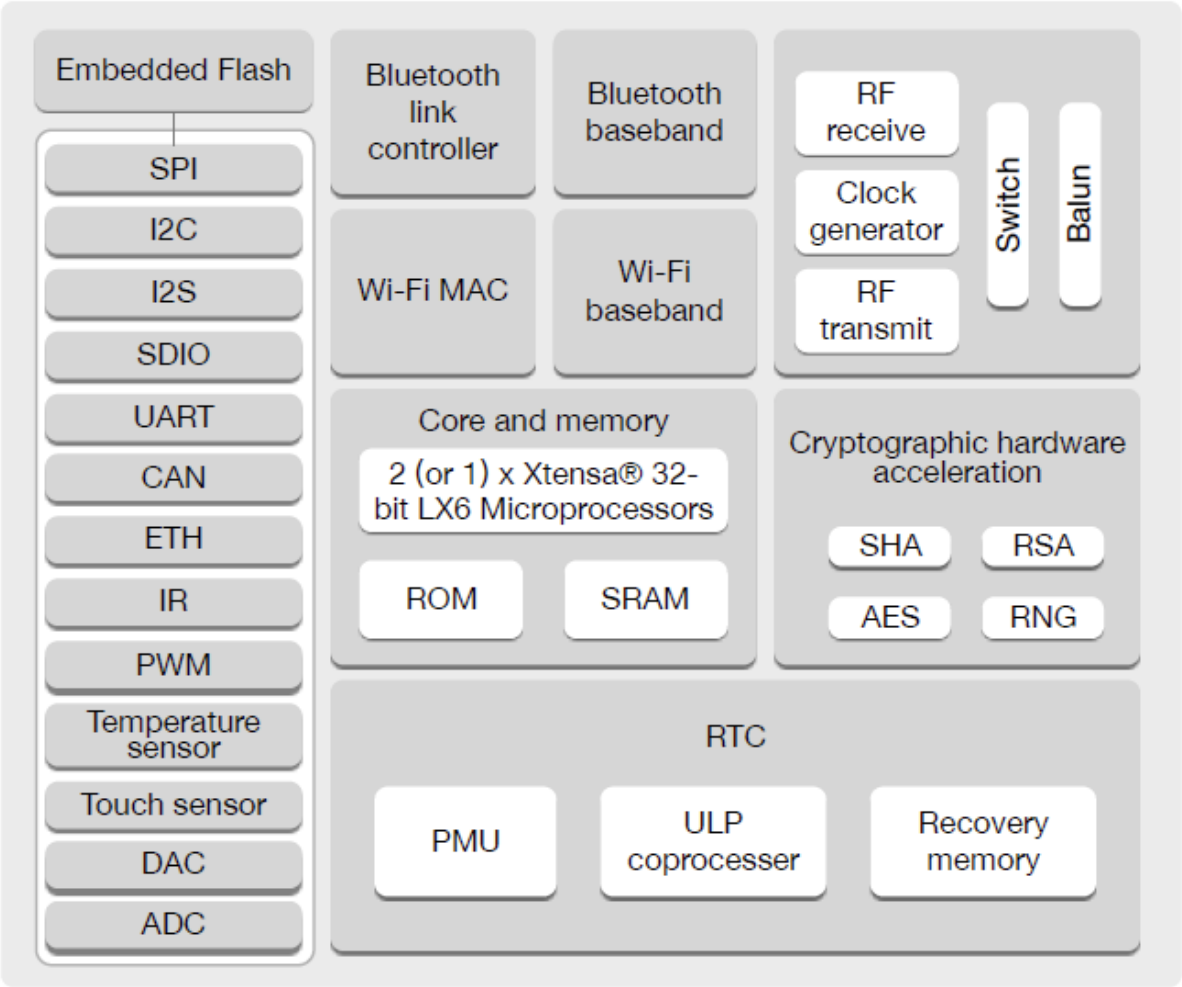

Figure 22 Block Diagram of ESP32

10.9 POWER SCHEME

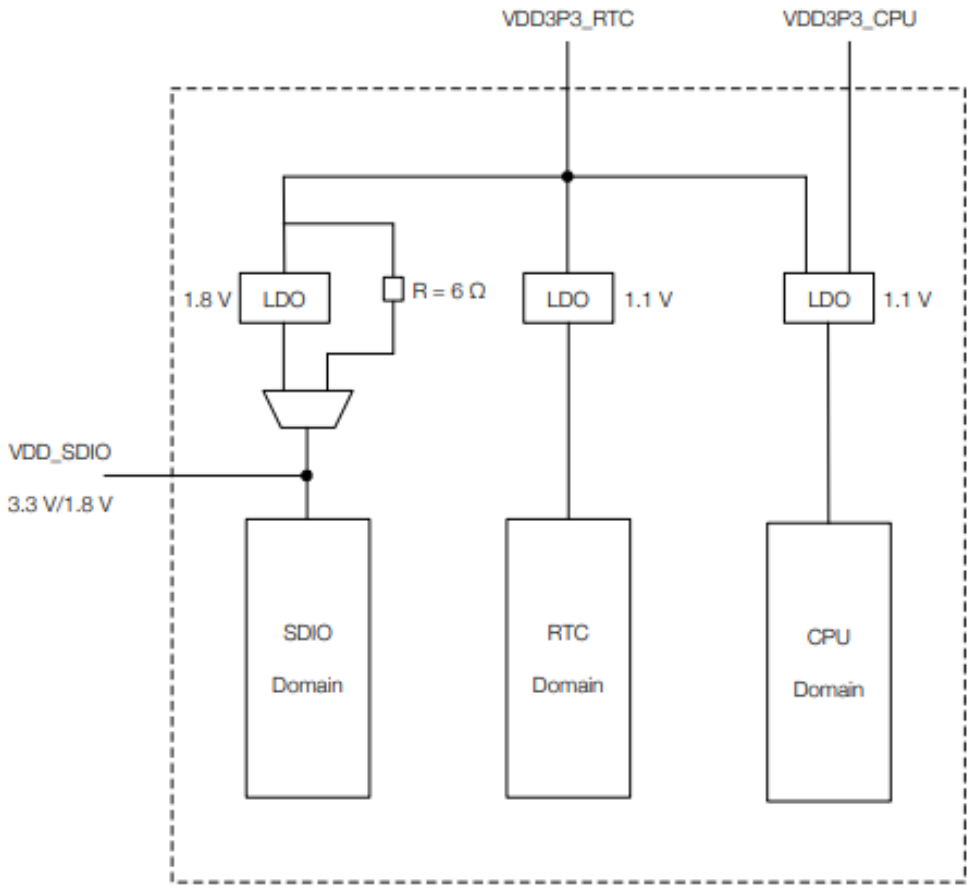

Figure 23 Power Schematic Diagram

- VDD3P3_RTC
- VDD3P3_CPU
- VDD_SDIO
- VDD3P3_RTC is also the input power supply for RTC and CPU.
- VDD3P3_CPU is also the input power supply for CPU.
- VDD_SDIO connects to the output of an internal LDO whose input is VDD3P3_RTC.
- When VDD_SDIO is connected to the same PCB net together with VDD3P3_RTC, the internal LDO is disabled automatically. The power scheme diagram is shown.

10.10 PIN LAYOUT

- ❖ CPU - ESP32 contains one or two low-power Xtensa® 32-bit LX6 microprocessor(s) with the following features:
- ❖ 7-stage pipeline to support the clock frequency of up to 240 MHz (160 MHz for ESP32-S0WD (NRND))
- ❖ 16/24-bit Instruction Set provides high code-density.
- ❖ Support for Floating Point Unit
- ❖ Internal Memory - ESP32's internal memory includes:
- ❖ 448 KB of ROM for booting and core functions
- ❖ 520 KB of on-chip SRAM for data and instructions
- ❖ 8 KB of SRAM in RTC, which is RTC FAST Memory and can be used for data storage; it's accessed by main CPU during RTC Boot from the Deep-sleep mode.

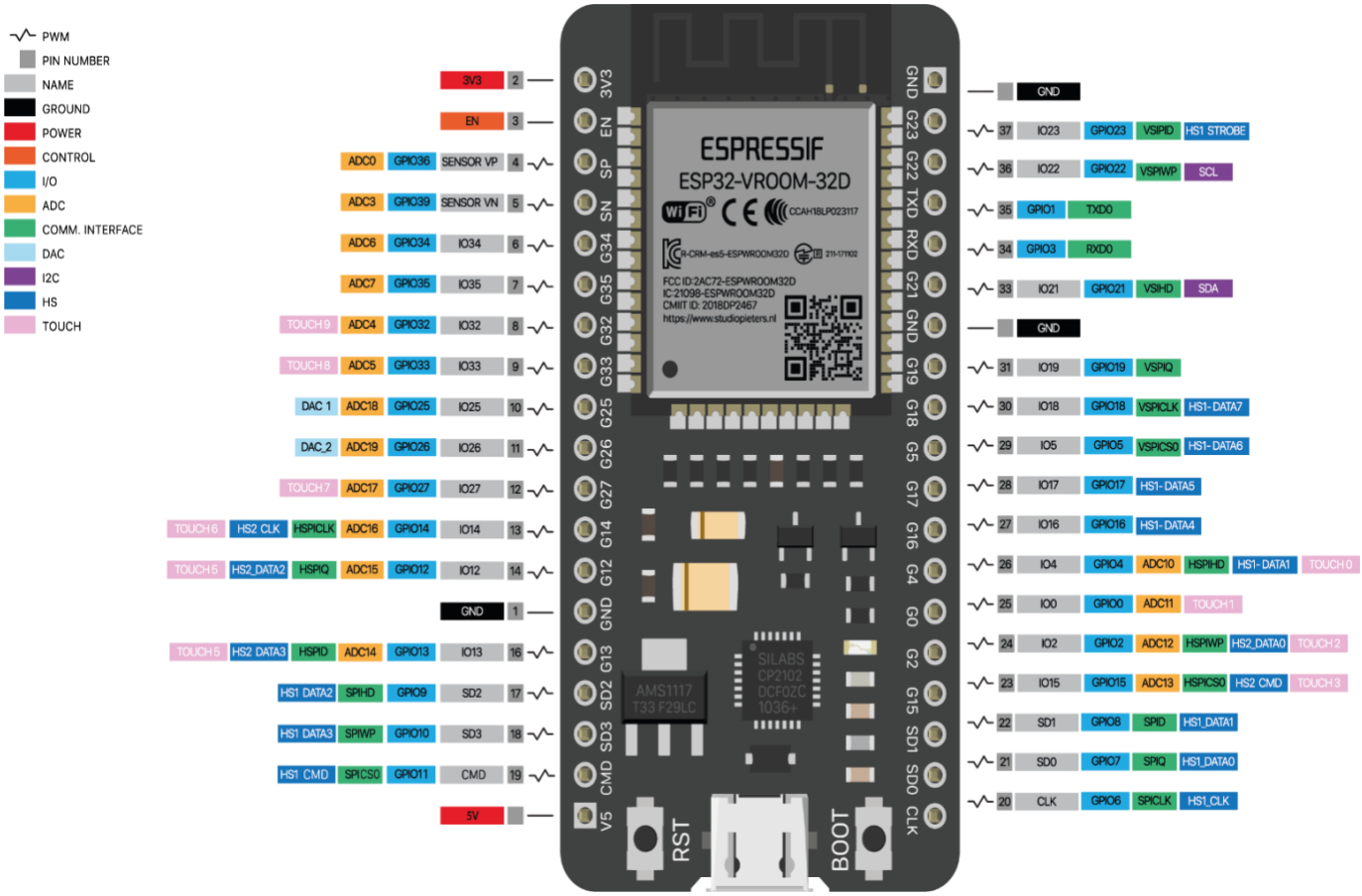

Figure 24 Pin Layout of ESP 32

CHAPTER 11 CIRCUIT DIAGRAM OF IOT AIDED VEGETAL IRRIGATION ROUTINE

11.1 INTRODUCTION

A circuit diagram (also known as an electrical diagram, elementary diagram, or electronic schematic) is a simplified conventional graphical representation of an electrical circuit.

The following circuit diagram depicts the main components of the system.

The circuit diagram was built and simulated on circuit-diagram.org. The above diagram clearly displays detailed yet easy to understand connections between all the components of the system.

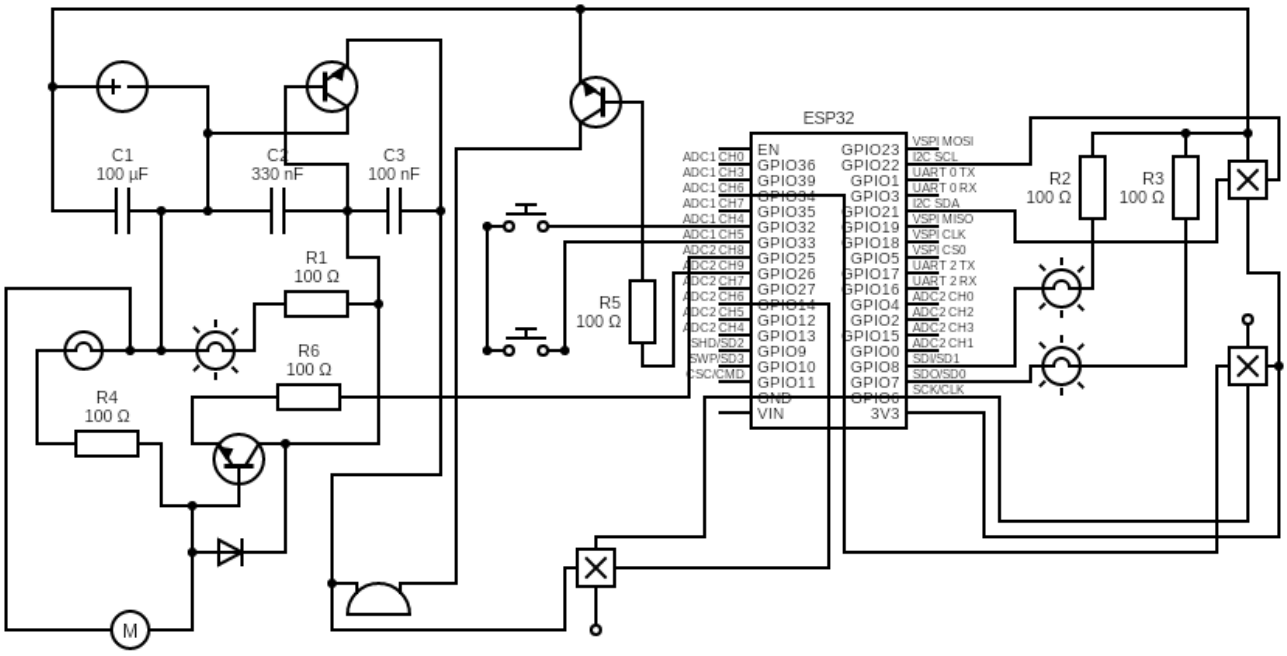

Figure 25 Circuit Diagram of the Model

11.2 FUNCTIONAL DESCRIPTION OF THE CIRCUIT –

1. The DC power source is connected to the voltage regulator and the capacitors (which are connected parallelly).
2. 100uF capacitor in series with a DC power supply helps ensure a cleaner and more stable power source for sensitive electronic components.
3. The Voltage Regulator reduces the 12 V DC to 5 V DC which is used to power ESP 32 and the rest of the components like Buzzer and the DHT 11 sensor.
4. The power source is directly connected to an LED + Resistor combination which acts as an indication whether the system is active or not.
5. The power is supplied to the TIP122 to establish further connectivity to the Solenoid Valve which requires an input of 12V DC to operate. The Valve is also connected to an LED + resistor combination to show its activity.
6. The Diode is placed between the Negative and Positive terminal of the NPN Transistor TIP122.
7. The diode is placed in series with the positive terminal of the transistor to provide reverse polarity protection.
8. The push buttons are connected to the ESP32's GPIO 32 and GPIO 33 respectively. GPIO 32 is being used to change mode of the working of system whereas the GPIO 33 is being used to manually operate the Solenoid Valve.
9. The DO of the DHT 11 is connected to ESP32's GPIO 14 to read the Temperature and Humidity Readings.
10. The Left side of the ESP32's is secured using a major ground line which terminates at the power supply itself.
11. At the right side of the ESP32, a Resistive soil moisture sensor is placed whose AO is connected to the GPIO 34 ADC1 CH6, so that the readings from the sensor could be easily read by the microcontroller.
12. The moisture sensor is powered by the 3.3 V DC output by the ESP32 which is also used for powering up the 0.9' OLED display.
13. The ground of ESP32, Soil Moisture Sensor and OLED screen are connected.
14. The OLED's SDA and SCL is connected to ESP32's GPIO 21 and GPIO 22 respectively.

15. GPIO 8 and GPIO 7 are connected to two LED + resistor combinations.
16. GPIO 8 is being used to show Wi-Fi connectivity.
17. GPIO 7 is being used to show whether the system is entirely active or not.
18. All the components are soldered on the general-purpose PCB to keep them secure and stable.

11.3 PCB LAYOUT –

A general-purpose PCB (Printed Circuit Board) is a versatile board that can be used for various electronic projects. Here are the steps to use a general-purpose PCB:

- Determine Your Project: Decide on the specific electronic project you want to build using the general-purpose PCB. It could be a simple circuit, a microcontroller-based project, or any other electronic application.
- Gather Components: Identify the components you'll need for your project, such as resistors, capacitors, integrated circuits, connectors, switches, LEDs, etc. Make sure you have the necessary components and their corresponding datasheets for reference.
- Plan Your Circuit: Sketch out the circuit diagram or create a schematic diagram using schematic design software or CAD tools. This diagram will guide you in placing and connecting the components on the PCB.
- Design the PCB Layout: Using PCB design software or CAD tools, create the PCB layout based on your circuit diagram. Place the components on the board and connect them using traces (copper pathways).
- Trace Routing: Route the traces on the PCB to establish electrical connections between the components. Pay attention to proper trace widths, clearance between traces, and signal integrity considerations. Ensure that the routing is optimized for signal flow and interference avoidance.
- Power and Ground Planes: Allocate appropriate areas for power and ground planes on the PCB layout. These planes help distribute power and provide a stable reference voltage for the components.

- Design Rule Check (DRC): Run a design rule check to identify any errors or design violations, such as incorrect clearances, unconnected pins, or other issues. Address any flagged problems to ensure a functional and reliable PCB.
- PCB Fabrication: Send the Gerber files to a PCB manufacturer or use a PCB prototyping service to fabricate the PCB based on your design. Choose a manufacturer that meets your specific requirements for quality, quantity, and turnaround time.
- PCB Assembly: Once you receive the fabricated PCB, you can start assembling the components onto the board. Use appropriate soldering techniques to attach the components to their designated pads on the PCB.

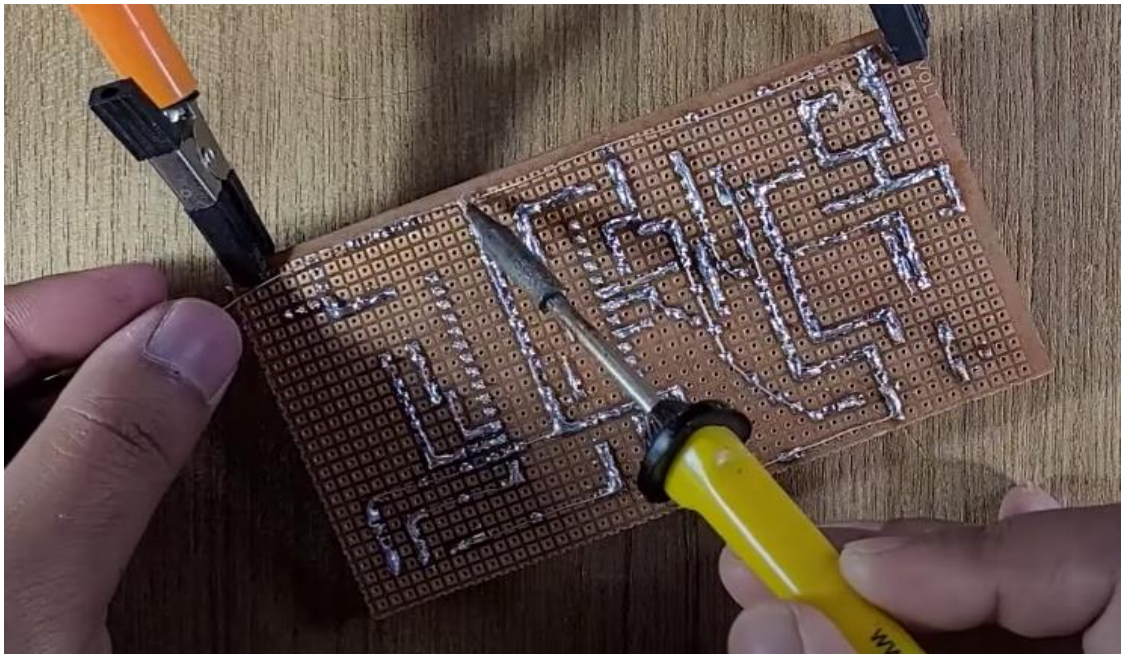

Figure 26 PCB Soldering Process

- Testing and Debugging: After assembly, thoroughly test the circuit to ensure it functions as expected. Check for any short circuits, open circuits, or other issues. Use multimeters, oscilloscopes, or other test equipment to validate the performance of your project.
- Enclosure and Integration: If required, design or obtain an enclosure for your project to provide protection and aesthetics. Mount the PCB and other components within the enclosure. Ensure proper wiring and connections between the PCB and external interfaces.

- ❖ Final Testing and Deployment: Perform final testing of your project to ensure it meets all functional and performance requirements. Once validated, deploy, or use the project in its intended application.

11.4 PCB TOP LAYOUT

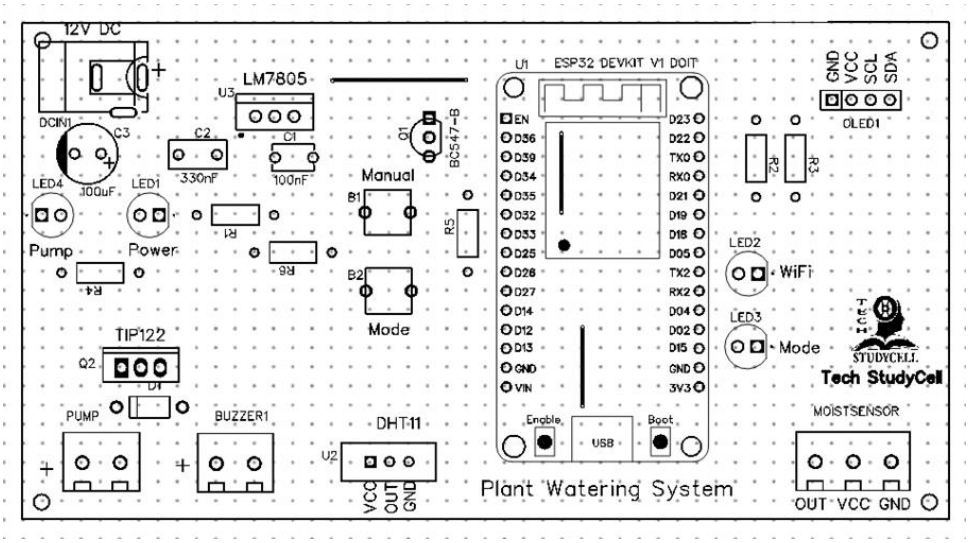

Figure 27 Components Arrangement on PCB

11.5 PCB BOTTOM LAYOUT

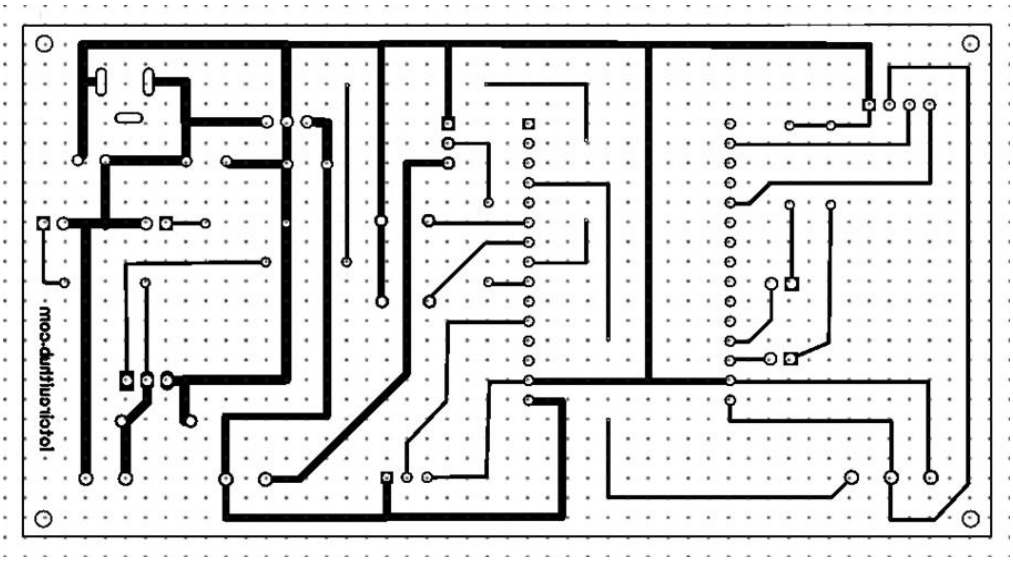

Figure 28 Connections of Components

11.6 ESP PIN CONNECTIONS.

ESP32 GPIOs Connected	Sensors/Components
D34	Resistive moisture sensor AOUT pin
D32	Push-Button to turn ON/OFF pump
D33	Push-Button for changing MODE.
D25	TIP122 base pin (controlling pump)
D26	BC547 base pin (controlling Buzzer)
D14	DHT11 sensor O/P pin
D15	LED indicator for MODE
D2	LED indicator for WiFi
D21	SDA of OLED
D22	SCL of OLED

CHAPTER 12 BLYNK IOT

12.1 INTRODUCTION

The first quarter of the 21st century has pushed the 4th industrial revolution with the increasingly widespread internet of things (IoT). One of the most popular applications on IoT is smart home. Smart home as a home which is smart enough to assist the inhabitants to live independently and comfortably with the help of technology is termed as smart home. In a smart home, all the mechanical and digital devices are interconnected to form a network, which can communicate with each other and with the user to create an interactive space.

Blynk is a popular IoT (Internet of Things) platform that allows you to easily build and control IoT projects using a smartphone or tablet. It provides a user-friendly interface to develop applications and connect hardware devices to the cloud. Blynk supports a wide range of hardware platforms, such as Arduino, Raspberry Pi, ESP8266, and many more.

12.2 WORKING

12.2.1 Sign up and Create a Blynk Account:

Start by signing up for a Blynk account on the Blynk website or through the Blynk app available on iOS and Android.

12.2.2 Create a New Project:

After logging in, create a new project in your Blynk account. Give your project a name and select the hardware platform you'll be using (Arduino, Raspberry Pi, etc.).

12.2.3 Obtain Blynk Libraries:

Depending on your hardware platform, download the appropriate Blynk libraries and install them in your development environment. These libraries provide the necessary functions and tools to connect your hardware to the Blynk platform.

12.2.4 Generate an Auth Token:

In your Blynk project, you'll find an Auth Token. This token is a unique identifier that allows your hardware device to communicate with the Blynk server. Keep this token safe as it's crucial for connecting your hardware to the Blynk platform.

12.2.5 Develop Your App Interface:

Use the Blynk app or web interface to design the user interface (UI) for your IoT project. You can add buttons, sliders, graphs, displays, and other widgets to control and monitor your connected devices.

12.2.6 Code Your Hardware:

Write the firmware code for your hardware device using the Blynk libraries. You'll need to configure the Wi-Fi or network settings and use the Auth Token to establish a connection between your device and the Blynk server. Use the Blynk library functions to read sensor data, control actuators, and update the UI widgets on the Blynk app.

12.2.7 Upload Firmware:

Compile and upload the firmware code to your hardware device using the appropriate development environment (Arduino IDE, Raspberry Pi tools, etc.).

12.2.8 Connect Your Hardware:

Make sure your hardware device is connected to the same Wi-Fi network or network as your smartphone or tablet running the Blynk app. The device should be powered on and ready to establish a connection with the Blynk server.

12.2.9 Test and Monitor:

Open the Blynk app on your smartphone or tablet and select your project. You should see the UI you designed earlier. Use the app to control your connected devices, monitor sensor data, and interact with your IoT project in real-time.

Blynk also offers additional features like data logging, push notifications, and integration with popular IoT services, allowing you to expand the functionality and capabilities of your projects.

12.3 BLYNK ARCHITECTURE

The Blynk platform includes the following components:

12.3.1 Blynk app builder:

Allows to you build apps for your projects using various widgets. It is available for Android and iOS platforms.

12.3.2 Blynk server:

Responsible for all the communications between your mobile device that's running the Blynk app and the hardware. You can use the Blynk Cloud or run your private Blynk server locally. It's open source, can easily handle thousands of devices, and can even be launched on a Raspberry Pi.

12.3.3 Blynk libraries:

Enables communication with the server and processes all the incoming and outcoming commands from your Blynk app and the hardware. They are available for all the popular hardware platforms.

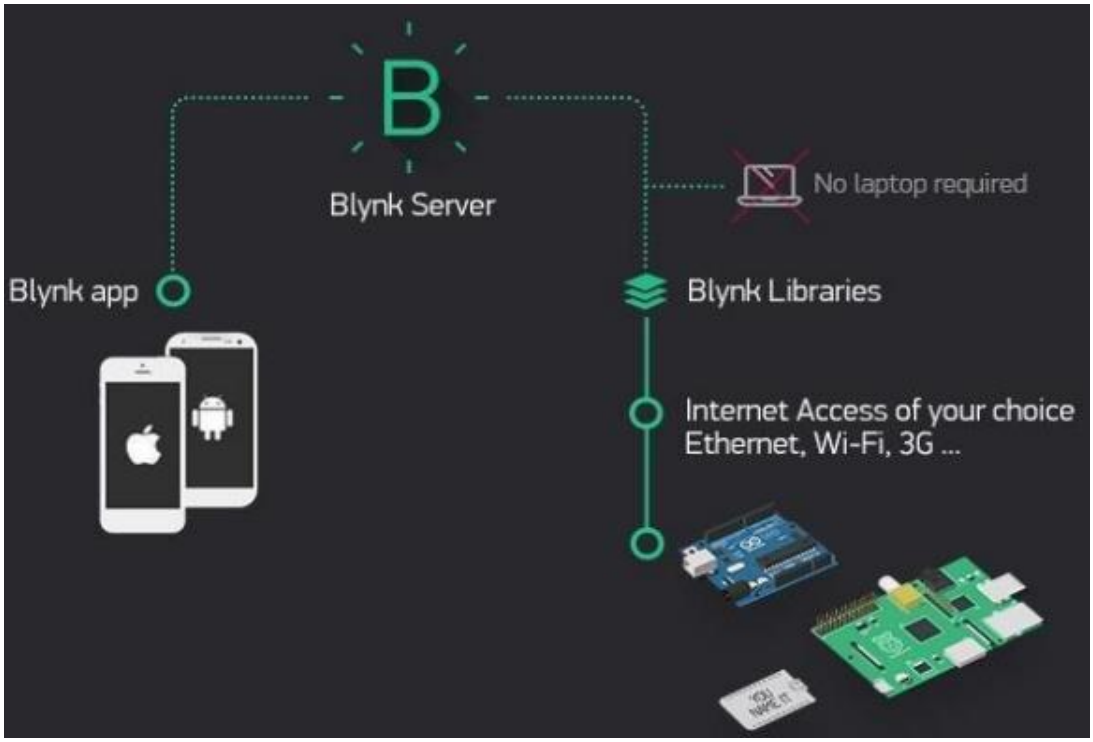

Figure 29 Blynk IoT Architecture in the Model

CHAPTER 13 FIREBASE

13.1 INTRODUCTION

Firebase is a comprehensive platform provided by Google that offers a range of backend services and tools to help developers build web and mobile applications. It provides various services for authentication, real-time database, cloud storage, hosting, machine learning, and more. Firebase simplifies the development process by handling the server infrastructure and offering easy integration with client-side applications.

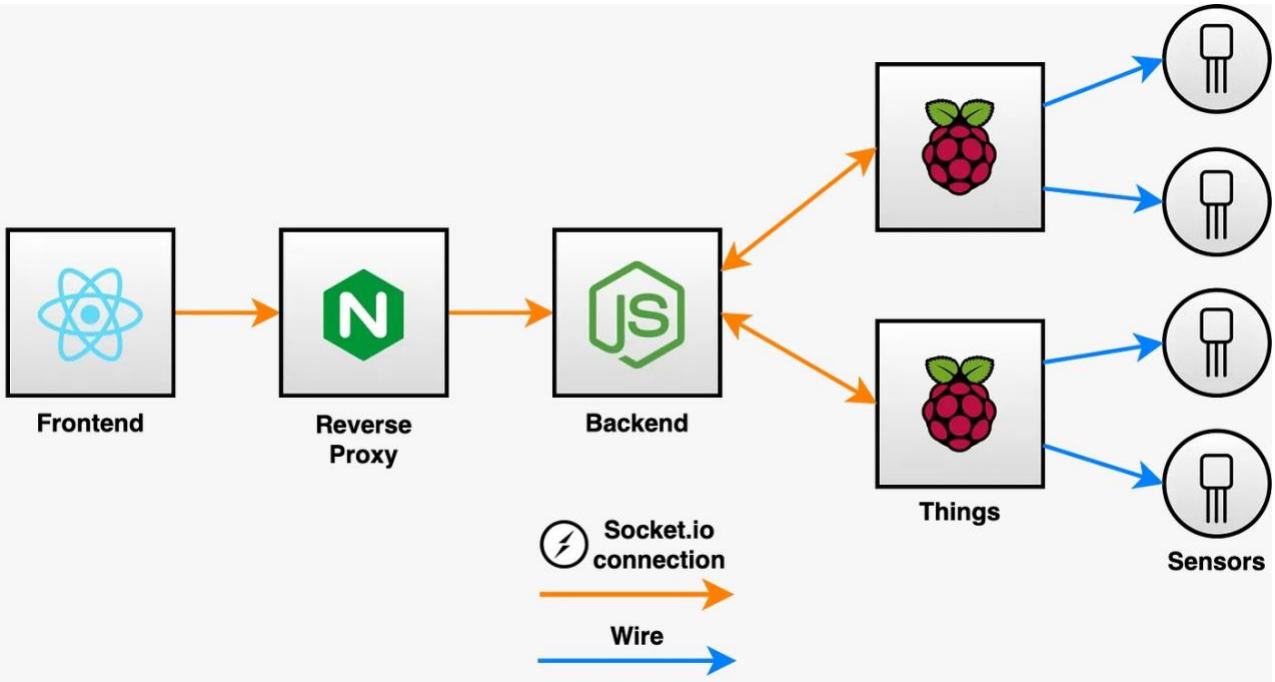

Figure 30 Process Flow of Firebase

13.2 FEATURES

Authentication: Firebase Authentication allows you to add user authentication to your application using different authentication methods like email/password, social media logins (Google, Facebook, Twitter), phone number verification, and more. It provides secure and easy-to-use authentication APIs and SDKs for different platforms.

Real-time Database: Firebase Realtime Database is a NoSQL cloud-hosted database that allows real-time synchronization of data across clients. It enables you to store and sync data

in real-time, making it suitable for collaborative applications, chat applications, and other scenarios where data needs to be updated instantly across multiple devices.

13.2.1 Cloud Firestore:

Cloud Firestore is a flexible, scalable, and fully managed NoSQL document database provided by Firebase. It offers powerful querying capabilities, offline data support, real-time updates, and automatic scaling. Firestore is designed to store, sync, and query structured data for web, mobile, and server applications.

13.2.2 Cloud Storage:

Firebase Cloud Storage provides secure cloud storage for your application's files, such as images, videos, and documents. It offers reliable file uploads and downloads, scalability, and integration with other Firebase services. You can easily manage user permissions, generate secure download URLs, and handle large file sizes.

13.2.3 Hosting:

Firebase Hosting allows you to deploy and host your web application with a simple and intuitive process. It provides a fast and secure content delivery network (CDN) that serves your static and dynamic content globally, ensuring quick loading times and high availability.

13.2.4 Cloud Functions:

Firebase Cloud Functions enables you to write serverless functions that automatically run in response to events triggered by Firebase services, such as database changes, authentication events, and HTTP requests. You can use Cloud Functions to extend your application's functionality and perform server-side logic without managing server infrastructure.

13.2.5 Analytics:

Firebase Analytics provides detailed insights into user behavior and app usage. It allows you to track key metrics, set up conversion events, and gain valuable insights into user engagement. The data collected can help you make data-driven decisions to improve your application.

13.2.6 Machine Learning:

Firebase offers ML Kit, a powerful mobile SDK that brings machine learning capabilities to your mobile applications. It provides ready-to-use APIs for text recognition, face detection, image labeling, barcode scanning, and more, simplifying the integration of machine learning features into your apps.

Firebase provides SDKs for various platforms, including iOS, Android, JavaScript, Unity, and more, making it easier to integrate Firebase services into your applications.

To get started with Firebase, you need to create a Firebase project in the Firebase console, configure the necessary services, and integrate the Firebase SDK into your application. Firebase documentation provides detailed guides, code samples, and tutorials to help you get up and running with each Firebase service. Note: As technology evolves, it's always recommended to refer to the official Firebase documentation for the most up-to-date and accurate information.

13.3 FIRE BASE REAL TIME DATABASE

Firebase Realtime Database is a cloud-hosted NoSQL database provided by Google as part of the Firebase platform. It is designed to store and synchronize data in real-time between connected clients. Firebase Realtime Database offers a simple and scalable solution for building real-time applications that require instant data synchronization across devices.

Key features of Firebase Realtime Database include:

- Real-time synchronization: Any changes made to the database are immediately synchronized and propagated to all connected clients. This means that updates made by one client are instantly visible to others.
- NoSQL data model: The database uses a flexible NoSQL JSON data model, allowing you to structure and organize data in a hierarchical manner.
- Offline capabilities: Firebase Realtime Database provides offline support, allowing clients to continue reading and writing data even when they are temporarily disconnected from the network. The data changes made during offline mode are synchronized when the connection is reestablished.

- Security and authentication: Firebase offer built-in authentication and security rules that allow you to control access to your database. You can enforce authentication and set up fine-grained rules to secure your data.
- SDKs and platform support: Firebase Realtime Database provides SDKs for various platforms, including web, iOS, Android, and server-side environments. This makes it easy to integrate the database into different applications and frameworks.
- To get started with Firebase Realtime Database, follow these general steps:
- Set up a Firebase project: Create a Firebase project on the Firebase Console (https://console.firebase.google.com/) and configure your project settings.
- Install the Firebase SDK: Add the Firebase SDK to your application by including the appropriate Firebase libraries for your platform. Firebase provides detailed documentation and guides for setting up the SDK in your specific development environment.
- Initialize the Firebase Realtime Database: Initialize the Firebase Realtime Database in your application code using the provided initialization methods from the Firebase SDK. This will establish the connection to your Firebase project.
- Read and write data: Use the Firebase SDK APIs to read and write data to the database. You can listen for real-time updates, perform queries, and update the database with the appropriate methods provided by the SDK.
- Set up security rules: Define security rules for your database to control who can read and write data. Firebase provides a rule-based language that allows you to specify access restrictions based on user authentication, data validation, and more.
- Firebase Realtime Database offers extensive documentation, tutorials, and sample code to help you understand the concepts and get started with building real-time applications. Refer to the Firebase documentation for the specific SDK and platform you are using to find detailed instructions and examples for working with Firebase Realtime Database.

13.4 FIREBASE ARCHITECTURE IN OUR SYSTEM –

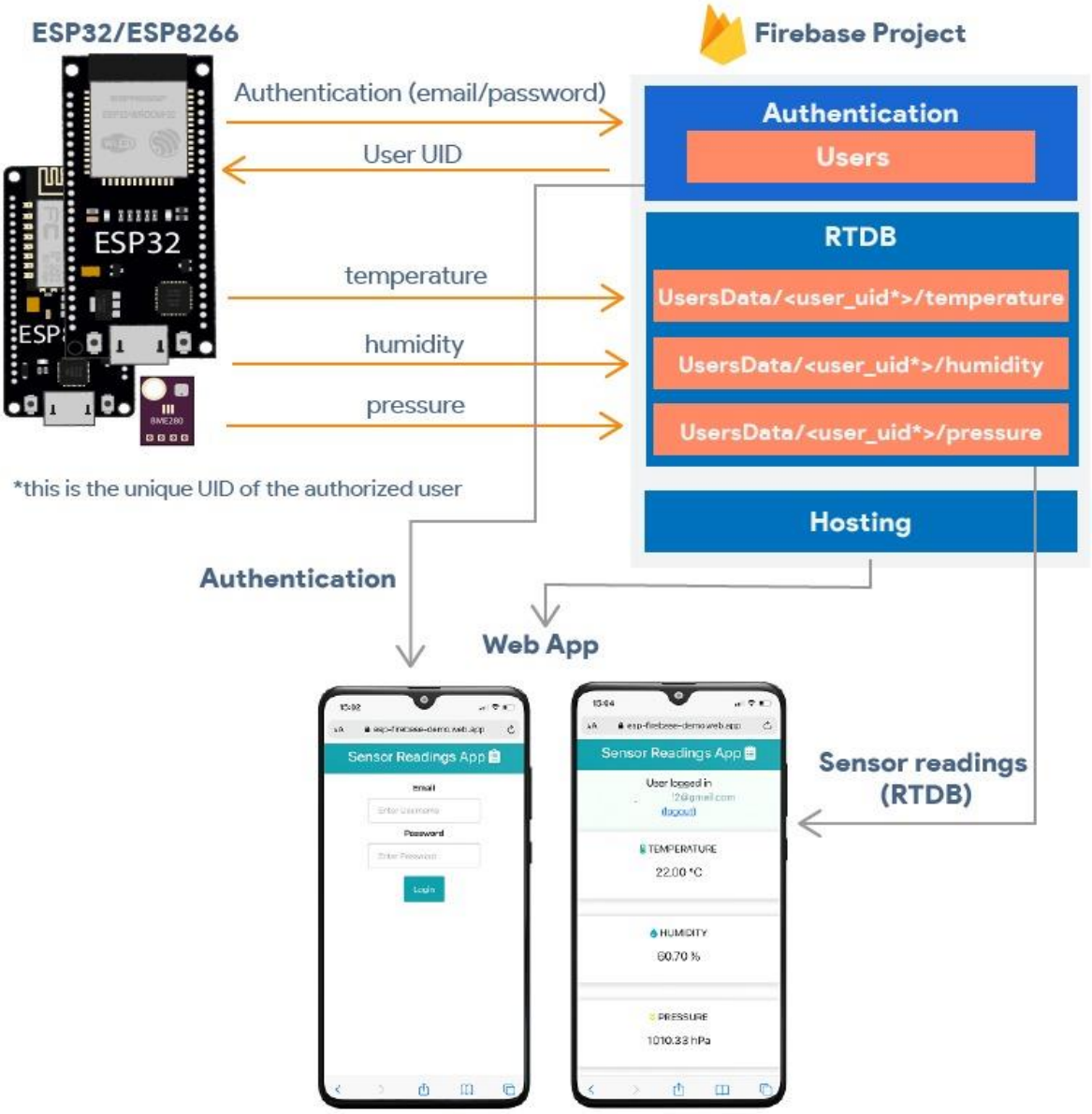

Figure 31 Firebase Connectivity in the Model

13.5 SCHEMA FOR THE REAL TIME DATABASE FOR THE MODEL

On Firebase Realtime Database, the schema for the model with soil moisture, temperature, and humidity fields can be designed as follows:

13.5.1 Fields:

- Each field will be represented as a node in the database with a unique identifier as the key.

- Each field node will have child properties such as field_name and field_location.

```arduino
- fields
  |- field_id_1
     |- field_name: "Field 1"
     |- field_location: "Location 1"
  |- field_id_2
     |- field_name: "Field 2"
     |- field_location: "Location 2"
  |- field_id_3
     |- field_name: "Field 3"
     |- field_location: "Location 3"
```

Figure 32 FIELD_ID in the Database

13.5.2 Soil Moisture Readings:

- Each soil moisture reading will be stored under the respective field node with a unique identifier as the key.

```
- soil_moisture_readings
  |- field_id_1
     |- moisture_reading_id_1
        |- moisture_level: 70
        |- reading_timestamp: "2023-05-28T10:00:00"
     |- moisture_reading_id_2
        |- moisture_level: 65
        |- reading_timestamp: "2023-05-28T11:00:00"
  |- field_id_2
     |- moisture_reading_id_1
        |- moisture_level: 80
        |- reading_timestamp: "2023-05-28T10:00:00"
     |- moisture_reading_id_2
        |- moisture_level: 75
        |- reading_timestamp: "2023-05-28T11:00:00"
```

Figure 33 Soil Moisture Readings in Database

- Each moisture reading node will have child properties such as moisture_level and reading_timestamp.

13.5.3 Temperature Readings:

- Each temperature reading will be stored under the respective field node with a unique identifier as the key.

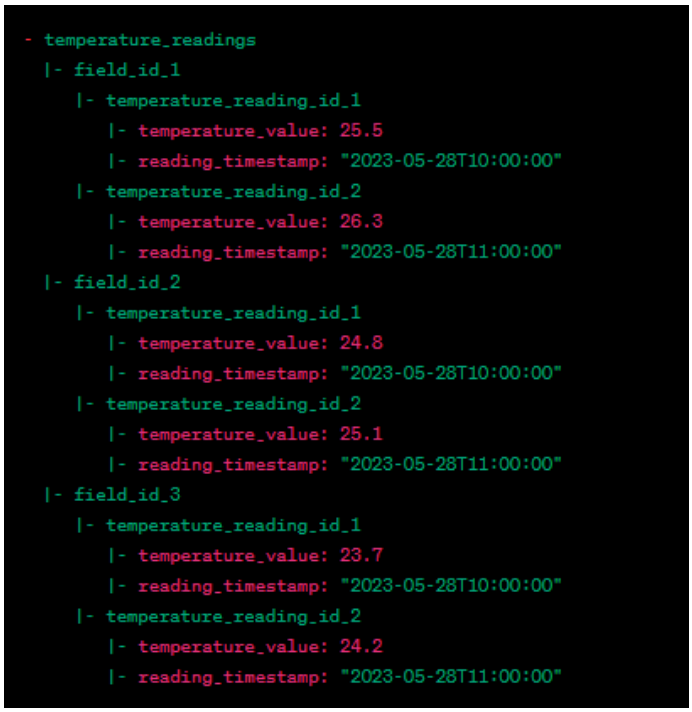

Figure 34 Temperature Readings in Database

- Each temperature reading node will have child properties such as temperature_value and reading_timestamp.

13.5.4 Humidity Readings:

- Each humidity reading will be stored under the respective field node with a unique identifier as the key.
- Each humidity reading node will have child properties such as humidity_value and reading_timestamp.

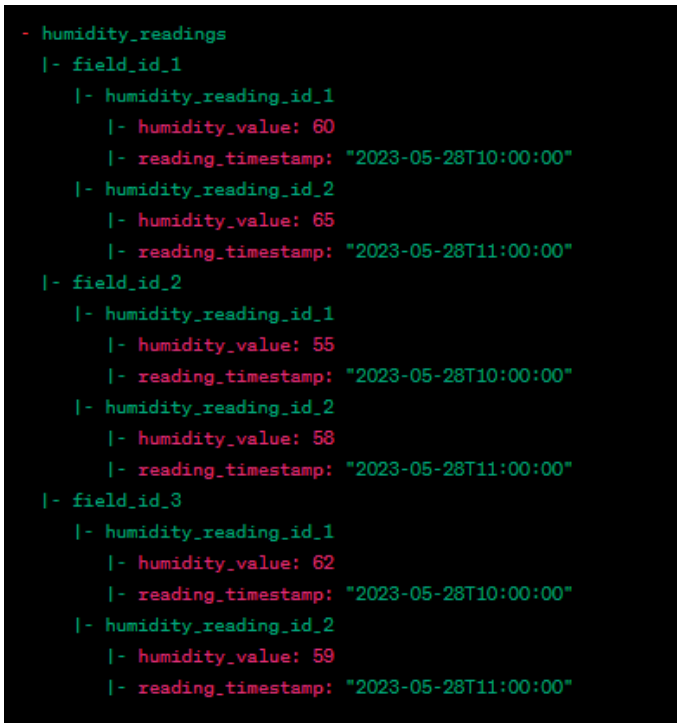

Figure 35 Humidity Readings in Database

This schema allows you to store information about the fields, their locations, soil moisture readings, temperature readings, and humidity readings. You can expand upon this schema by adding additional fields or tables as needed, such as weather data, sensor information, or irrigation schedules.

CHAPTER 14 SOFTWARE DESCRIPTION AND PROGRAMMING

This section of the documents mentions how the software side of the model is processed and programmed to make it operable.

14.1 LIBRARIES USED

14.1.1 ADAFRUIT.H

It is a collection of open-source software libraries developed by Adafruit Industries. These libraries are designed to provide easy-to-use interfaces and functions for controlling Adafruit's hardware modules and breakout boards.

The Adafruit library covers a wide range of devices, including sensors, displays, motor controllers, wireless modules, and more. It simplifies the process of working with these devices by providing higher-level functions and abstracting away low-level details.

To use the Adafruit library, follow these steps:

- Install the Arduino IDE: If you haven't already, download and install the Arduino IDE from the official Arduino website.
- Install the Adafruit library: Launch the Arduino IDE and go to "Sketch" -> "Include Library" -> "Manage Libraries". In the Library Manager window, search for the specific Adafruit module/library you want to use. For example, if you're using the Adafruit NeoPixel RGB LED modules, search for "Adafruit NeoPixel" and click the "Install" button to install the library.
- Include the library in your sketch: In your Arduino sketch, include the necessary Adafruit library by adding the following line at the beginning of #include <Adafruit_ModuleName.h>
- Replace "ModuleName" with the specific Adafruit module/library you are using.
- Utilize the library functions: Once the library is included, you can make use of its functions and classes to interact with the Adafruit hardware. Refer to the library's

documentation and example code to learn about the available functions and how to use them.

- Upload and test your sketch: Connect your Adafruit hardware module to your Arduino board as per the provided instructions. Compile and upload your sketch to the Arduino board and observe the behavior of the hardware module based on your code.
- It's worth noting that the specific steps and library names may vary depending on the Adafruit module you are using. Make sure to refer to the official Adafruit documentation, tutorials, and examples for detailed instructions on using their libraries with your specific hardware.

14.1.2 WIFI.H

In ESP32, the wifi.h header file is part of the WiFi library, which provides functions and definitions for interacting with Wi-Fi networks using the ESP32 Wi-Fi module. This library allows you to configure Wi-Fi settings, connect to networks, manage connections, and

```
#include <WiFi.h>

const char* ssid = "YourSSID";
const char* password = "YourPassword";

void setup() {
  Serial.begin(115200);
  delay(100);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("Wi-Fi connected");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
}
```

Figure 36 Code Snippet to implement WiFi.h

perform network operations. Here's a basic example that demonstrates the usage of wifi.h in ESP32:

In this example, we include the WiFi.h header file, set the Wi-Fi network name (SSID) and password, and then connect to the network using WiFi.begin(). We then wait for the connection to be established using WiFi.status(), and once connected, we print the local IP address to the serial monitor.

Note that this example assumes you have already installed the ESP32 core for Arduino in your Arduino environment. If you haven't, you can install it by going to "Tools" -> "Board" -> "Boards Manager" and searching for "esp32". Once installed, you can select the appropriate ESP32 board from the "Tools" -> "Board" menu.

14.1.3 WIFI_CLIENT.H

To establish a Wi-Fi client connection using the ESP32 module, you can utilize the WiFiClient class from the WiFi library. Here's an example that demonstrates how to create a Wi-Fi client and connect to a server:

In this example, we include the WiFi.h header file and define the Wi-Fi network name (SSID) and password. We also set the IP address and port of the server you want to connect to.

Inside the setup() function, we establish a Wi-Fi connection using WiFi.begin(). We wait for the connection to be established, and once connected, we print the local IP address to the serial monitor.

Next, we use the connect() method of the WiFiClient class to connect to the server specified by serverIP and serverPort. If the connection is successful, we print a message to indicate the successful connection and send a simple message to the server using client.println().

Remember to replace "YourSSID", "YourPassword", and "ServerIPAddress" with the appropriate values for your Wi-Fi network and server.

You can add your own code inside the loop() function to perform any necessary operations or handle data received from the server.

Please note that this is a basic example, and you may need to handle error conditions and implement additional logic depending on your specific application requirements.

```
#include <WiFi.h>

const char* ssid = "YourSSID";
const char* password = "YourPassword";
const char* serverIP = "ServerIPAddress";
const int serverPort = 80;

WiFiClient client;

void setup() {
  Serial.begin(115200);
  delay(100);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("Wi-Fi connected");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());

  if (client.connect(serverIP, serverPort)) {
    Serial.println("Connected to server");
    client.println("Hello from ESP32");
    client.println();
  }
```

Figure 37 Code Snippet to Implement WiFi_client.h

14.1.4 DHT.H

The dht.h header file is not a standard part of the ESP32 development environment. However, it is commonly used to interface with DHT (Digital Humidity and Temperature) sensors in Arduino-based projects, including those involving the ESP32.

To use the DHT sensor with the ESP32, you will need to install the DHT sensor library for Arduino. There are several libraries available for different versions of DHT sensors, such as DHT11, DHT22, and AM2302. One popular library is the "DHT sensor library" by Adafruit.

To install the library, follow these steps:

- Open the Arduino IDE.
- Go to "Sketch" -> "Include Library" -> "Manage Libraries".
- In the Library Manager, search for "DHT sensor library".
- Look for the library by Adafruit and click on it.
- Click the "Install" button to install the library.

After installing the library, you can include the DHT.h header file in your ESP32 project. Here's an example that demonstrates how to use the DHT sensor library with an ESP32:

```
#include <DHT.h>

#define DHTPIN 4        // Pin connected to the DHT sensor
#define DHTTYPE DHT11   // Type of DHT sensor (DHT11 or DHT22)

DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(115200);
  delay(100);

  dht.begin();
}

void loop() {
  delay(2000);

  float temperature = dht.readTemperature();
  float humidity = dht.readHumidity();

  Serial.print("Temperature: ");
  Serial.print(temperature);
  Serial.print(" °C, Humidity: ");
  Serial.print(humidity);
  Serial.println(" %");
}
```

Figure 38 Code Snippet to import DHT.h

In this example, we include the DHT.h header file and define the pin (DHTPIN) connected to the DHT sensor and the type of DHT sensor (DHTTYPE).

Inside the setup() function, we initialize the DHT sensor using dht.begin().

In the loop() function, we read the temperature and humidity values from the DHT sensor using dht.readTemperature() and dht.readHumidity(). We then print the values to the serial monitor.

14.1.5 ACEBUTTON.H

The AceButton library is a lightweight Arduino library that provides an easy way to create and manage buttons in your projects. It simplifies the process of handling button presses, releases, and various button states.

To use the AceButton library, you need to install it in your Arduino environment. Here's how you can install it:

- Open the Arduino IDE.
- Go to "Sketch" -> "Include Library" -> "Manage Libraries".
- In the Library Manager, search for "AceButton".
- Look for the library called "AceButton" by Brian T. Park and click on it.
- Click the "Install" button to install the library.

```
#include <AceButton.h>

const int BUTTON_PIN = 2;

AceButton button(BUTTON_PIN);

void handleEvent(AceButton*, uint8_t, uint8_t);

void setup() {
  Serial.begin(115200);
  button.setEventHandler(handleEvent);
}

void loop() {
  button.check();
}

void handleEvent(AceButton* button, uint8_t eventType, uint8_t buttonState)
  if (eventType == AceButton::kEventReleased) {
    Serial.println("Button released");
  }
```

Figure 39 Code Snippet to import Acebutton.h

Once you have installed the AceButton library, you can use it in your Arduino projects. Here's a simple example that demonstrates the basic usage of AceButton:

In this example, we include the AceButton.h header file and define the pin (BUTTON_PIN) to which the button is connected.

Inside the setup() function, we initialize the AceButton with the specified pin using button.setEventHandler() to set the event handler function.

In the loop() function, we call button.check() to check the button state.

The handleEvent() function is the event handler that will be called when the button state changes. In this example, we print a message to the serial monitor when the button is released.

14.2 CODE FOR THE FUNCTIONING OF ESP32

14.2.1 CODE OF THE OPERATION OF ESP32 WITH COMPONENTS

```
/* Blynk Template ID */
#define testing ""
#define ESP32 MODEL ""
#define BLYNK_AUTH_TOKEN ""
// Your WiFi credentials.
char ssid[] = "IOTTEST"; //WiFi Name
char_password[] = “”; // WiFi Password
//Setting the maximum wet and maximum dry value measured by the sensor
int wetSoilVal = 930;  //min value when soil is wet
int drySoilVal = 3000;  //max value when soil is dry
//Set ideal moisture range percentage (%) in soil
int moistPerLow =   20;  //min moisture %
int moistPerHigh =   80;  //max moisture %
//Libraries
#include <Adafruit_SSD1306.h>
#include <WiFi.h>
#include <WiFiClient.h>
#include <BlynkSimpleEsp32.h>
#include <DHT.h>
#include <AceButton.h>
using namespace ace_button;
// Define connections to sensor
```

```
#define SensorPin       34  //D34
#define DHTPin          14 //D14
#define RelayPin        25 //D25
#define wifiLed         2   //D2
#define RelayButtonPin 32 //D32
#define ModeSwitchPin   33 //D33
#define BuzzerPin       26 //D26
#define ModeLed         15 //D15
#define DHTTYPE DHT11     // DHT 11
#define VPIN_MoistPer    V1
#define VPIN_TEMPERATURE V2
#define VPIN_HUMIDITY    V3
#define VPIN_MODE_SWITCH V4
#define VPIN_RELAY       V5
#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 32 // OLED display height, in pixels
// Declaration for an SSD1306 display connected to I2C (SDA, SCL pins)
#define OLED_RESET     -1 // Reset pin # (or -1 if sharing Arduino reset pin)
Adafruit_SSD1306 display (SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
int     sensorVal;
int     moisturePercentage;
bool    toggleRelay = LOW;
bool    prevMode = true;
int     temperature1 = 0;
```

```
int     humidity1   = 0;
string  currMode  = "A";
char auth[] = BLYNK_AUTH_TOKEN;
ButtonConfig config1;
AceButton button1(&config1);
ButtonConfig config2;
AceButton button2(&config2);
void handleEvent1(AceButton*, uint8_t, uint8_t);
void handleEvent2(AceButton*, uint8_t, uint8_t);
BlynkTimer timer;
DHT dht(DHTPin, DHTTYPE);
void checkBlynkStatus() { // called every 3 seconds by SimpleTimer
  bool isconnected = Blynk.connected();
  if (isconnected == false) {
    Serial.print("Blynk Not Connected ");
    digitalWrite(wifiLed, LOW);
  }
  if (isconnected == true) {
    digitalWrite(wifiLed, HIGH);
    Serial.println("Blynk Connected");
  }}
BLYNK_CONNECTED () {
  Blynk.syncVirtual(VPIN_MoistPer);
  Blynk.syncVirtual(VPIN_RELAY);
```

```
  Blynk.syncVirtual(VPIN_TEMPERATURE);
  Blynk.syncVirtual(VPIN_HUMIDITY);
  //Blynk.syncVirtual(VPIN_MODE_SWITCH);
  Blynk.virtualWrite(VPIN_MODE_SWITCH, prevMode);
}
BLYNK_WRITE(VPIN_RELAY) {
  if(!prevMode){
    toggleRelay = param.asInt();
    digitalWrite(RelayPin, toggleRelay);
  }
  else{
    Blynk.virtualWrite(VPIN_RELAY, toggleRelay);
  }}
BLYNK_WRITE(VPIN_MODE_SWITCH) {
  if(prevMode !=  param.asInt()){
    prevMode =  param.asInt();
    currMode = prevMode ? "A" : "M";
    digitalWrite(ModeLed, prevMode);
    controlBuzzer(500);
    if(!prevMode && toggleRelay == HIGH){
      digitalWrite(RelayPin, LOW);
      toggleRelay = LOW;
      Blynk.virtualWrite(VPIN_RELAY, toggleRelay);
    }  }  }
```

```
void controlBuzzer(int duration){
  digitalWrite(BuzzerPin, HIGH);
  delay(duration);
  digitalWrite(BuzzerPin, LOW);
}
void displayData(String line1 , String line2){
  display.clearDisplay();
  display.setTextSize(2);
  display.setCursor(30,2);
  display.print(line1);
  display.setTextSize(1);
  display.setCursor(1,25);
  display.print(line2);
  display.display();
}
void getMoisture(){
  sensorVal = analogRead(SensorPin);
  if (sensorVal > (wetSoilVal - 100) && sensorVal < (drySoilVal + 100) ){
    moisturePercentage = map(sensorVal ,drySoilVal, wetSoilVal, 0, 100);
    // Print result to serial monitor
    Serial.print("Moisture Percentage: ");
    Serial.print(moisturePercentage);
    Serial.println(" %");
  }
```

```
  else{
    Serial.println(sensorVal);
  }
  delay(100);
}
void getWeather(){
  float h = dht.readHumidity();
  float t = dht.readTemperature(); // or dht.readTemperature(true) for Fahrenheit
  if (isnan(h) || isnan(t)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }
  else {
    humidity1 = int(h);
    temperature1 = int(t);
   Serial.println(temperature1);
   Serial.println(humidity1);
  } }
void sendSensor()
{
  getMoisture(); // get Moisture reading
  getWeather(); // get reading from DHT11
  displayData(String(moisturePercentage) + " %", "T:" + String(temperature1) + " C,  H:" +
String(humidity1) + " %  " + currMode);
  Blynk.virtualWrite(VPIN_MoistPer, moisturePercentage);
```

```
  Blynk.virtualWrite(VPIN_TEMPERATURE, temperature1);
  Blynk.virtualWrite(VPIN_HUMIDITY, humidity1);
}
void controlMoist(){
  if(prevMode){
    if (moisturePercentage < (moistPerLow)){
      if (toggleRelay == LOW){
        controlBuzzer(500);
        digitalWrite(RelayPin, HIGH);
        toggleRelay = HIGH;
        Blynk.virtualWrite(VPIN_RELAY, toggleRelay);
        delay(1000);
      }         }
    if (moisturePercentage > (moistPerHigh)){
      if (toggleRelay == HIGH){
        controlBuzzer(500);
        digitalWrite(RelayPin, LOW);
        toggleRelay = LOW;
        Blynk.virtualWrite(VPIN_RELAY, toggleRelay);
        delay(1000);
      }    }   }
  else{
    button1.check();
  }}
```

```
void setup() {
 // Set up serial monitor
 Serial.begin(115200);
 // Set pinmodes for GPIOs
 pinMode(RelayPin, OUTPUT);
 pinMode(wifiLed, OUTPUT);
 pinMode(ModeLed, OUTPUT);
 pinMode(BuzzerPin, OUTPUT);
 pinMode(RelayButtonPin, INPUT_PULLUP);
 pinMode(ModeSwitchPin, INPUT_PULLUP);
 digitalWrite(wifiLed, LOW);
 digitalWrite(ModeLed, LOW);
 digitalWrite(BuzzerPin, LOW);
 dht.begin();    // Enabling DHT sensor
 config1.setEventHandler(button1Handler);
 config2.setEventHandler(button2Handler);
 button1.init(RelayButtonPin);
 button2.init(ModeSwitchPin);
 if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
   Serial.println(F("SSD1306 allocation failed"));
   for(;;);
 }
 delay(1000);
 display.setTextSize(1);
```

```
  display.setTextColor(WHITE);
  display.clearDisplay();
  WiFi.begin(ssid, pass);
  timer.setInterval(2000L, checkBlynkStatus); // check if Blynk server is connected every 2 seconds
  timer.setInterval(3000L, sendSensor); // display and send sensor reading every 3 seconds
  Blynk.config(auth);
  //delay(1000);
  controlBuzzer(1000);
  digitalWrite(ModeLed, prevMode);
}
 void loop() {
  Blynk.run();
  timer.run(); // Initiates SimpleTimer
  button2.check();
  controlMoist();
}
void button1Handler(AceButton* button, uint8_t eventType, uint8_t buttonState) {
  Serial.println("EVENT1");
  switch (eventType) {
    case AceButton::kEventReleased:
      //Serial.println("kEventReleased");
      digitalWrite(RelayPin, !digitalRead(RelayPin));
      toggleRelay = digitalRead(RelayPin);
      Blynk.virtualWrite(VPIN_RELAY, toggleRelay);
```

```
      break;
  }}
void button2Handler(AceButton* button, uint8_t eventType, uint8_t buttonState) {
  Serial.println("EVENT2");
  switch (eventType) {
    case AceButton::kEventReleased:
      //Serial.println("kEventReleased");
      if(prevMode && toggleRelay == HIGH){
        digitalWrite(RelayPin, LOW);
        toggleRelay = LOW;
        Blynk.virtualWrite(VPIN_RELAY, toggleRelay);
      }
      prevMode = !prevMode;
      currMode = prevMode ? "A" : "M";
      digitalWrite(ModeLed, prevMode);
      Blynk.virtualWrite(VPIN_MODE_SWITCH, prevMode);
      controlBuzzer(500);
      break;
  }}
```

14.2.2 CODE FOR OPERATION OF SOIL MOISTURE SENSOR

```
//Get Resistive Soil Moisture Sensor V2.0 Reading (MAX & MIN)
//Provide 3.3V across VCC & GND, and connect AOUT pin with D34 of ESP32
#define SensorPin      34 //D34
int    sensorVal;
```

```
void setup () {
  // Set up serial monitor
  Serial.begin(115200);
}
void loop () {
  sensorVal = analogRead(SensorPin);
  Serial.print("Moisture Value: ");
  Serial.println(sensorVal);
  delay (1000);
}
```

14.2.3 CODE OF THE IMPLEMENTATION OF FIREBASE DATABASE

```
#if defined (ESP32)
#include <WiFi.h>
#include <FirebaseESP32.h>
#elif defined (ESP8266)
#include <ESP8266WiFi.h>
#include <FirebaseESP8266.h>
#endif
#include "DHT.h"
//Provide the token generation process info.
#include <addons/TokenHelper.h>

//Provide the RTDB payload printing info and other helper functions.
```

```
#include <addons/RTDBHelper.h>

#define DHTPIN 2 // Digital pin connected to the DHT sensor

#define DHTTYPE DHT11 // DHT 11

/* 1. Define the WiFi credentials */

#define WIFI_SSID "iot_net_source"

#define WIFI_PASSWORD ""

/* 2. Define the API Key */

#define API_KEY "AIzaSyDONmOXVgqkyp8QjBU5NW-cJB-xlOS7Tmc"

/* 3. Define the RTDB URL */

#define DATABASE_URL " espiot-a54cb-default-rtdb.firebaseio.com" ;

/* 4. Define the user Email and password that alreadey registerd or added in your project
*/

#define USER_EMAIL "amrittyagigzb@gmail.com"

#define USER_PASSWORD "omshanti123"

//Define Firebase Data object

FirebaseData fbdo;

FirebaseAuth auth;

FirebaseConfig config;


unsigned long sendDataPrevMillis = 0;

unsigned long count = 0;

DHT dht(DHTPIN, DHTTYPE);

void setup()

{

  Serial.begin(115200);
```

```
dht.begin();
WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
Serial.print("Connecting to Wi-Fi");
while (WiFi.status() != WL_CONNECTED)
{
  Serial.print(".");
  delay(300);
}
Serial.println();
Serial.print("Connected with IP: ");
Serial.println(WiFi.localIP());
Serial.println();
Serial.printf("Firebase Client v%s\n\n", FIREBASE_CLIENT_VERSION);
/* Assign the api key (required) */
config.api_key = API_KEY;
/* Assign the user sign in credentials */
auth.user.email = USER_EMAIL;
auth.user.password = USER_PASSWORD;
/* Assign the RTDB URL (required) */
config.database_url = DATABASE_URL;
/* Assign the callback function for the long running token generation task */
config.token_status_callback = tokenStatusCallback; //see addons/TokenHelper.h
//Or use legacy authenticate method
//config.database_url = DATABASE_URL;
```

```
  //config.signer.tokens.legacy_token = "<database secret>";

  Firebase.begin(&config, &auth);

  Firebase.reconnectWiFi(true);

  Firebase.setDoubleDigits(5);

  /** Timeout options.

  //WiFi reconnected timeout (interval) in ms (10 sec - 5 min) when WiFi disconnected.

  config.timeout.wifiReconnect = 10 * 1000;

  //Socket connection and SSL handshake timeout in ms (1 sec - 1 min).

  config.timeout.socketConnection = 10 * 1000;

  //Server response read timeout in ms (1 sec - 1 min).

  config.timeout.serverResponse = 10 * 1000;

  //RTDB Stream keep-alive timeout in ms (20 sec - 2 min) when no server's keep-alive
event data received.

  config.timeout.rtdbKeepAlive = 45 * 1000;

  //RTDB Stream reconnect timeout (interval) in ms (1 sec - 1 min) when RTDB Stream
closed and want to resume.

  config.timeout.rtdbStreamReconnect = 1 * 1000;

  //RTDB Stream error notification timeout (interval) in ms (3 sec - 30 sec). It determines
how often the readStream will return false (error) when it called repeatedly in loop.

  config.timeout.rtdbStreamError = 3 * 1000;

  Note:

 The function that starting the new TCP session i.e. first time server connection or previous
session was closed, the function won't exit until the time of
config.timeout.socketConnection.

  You can also set the TCP data sending retry with

  config.tcp_data_sending_retry = 1;

  */
```

```
}

void loop()

{

  float h = dht.readHumidity();

  float t = dht.readTemperature();

  if (Firebase.ready() && (millis() - sendDataPrevMillis > 15000 || sendDataPrevMillis == 0))

  {

    sendDataPrevMillis = millis();

Serial.printf("Set Temperature... %s\n", Firebase.setFloat(fbdo, F("/test/temperature"), t) ? "ok" : fbdo.errorReason().c_str());

    Serial.printf("Get Temperature... %s\n", Firebase.getFloat(fbdo, F("/test/temperature")) ? String(fbdo.to<float>()).c_str() : fbdo.errorReason().c_str());

    Serial.printf("Set Humidity... %s\n", Firebase.setDouble(fbdo, F("/test/humidity"), h) ? "ok" : fbdo.errorReason().c_str());

    Serial.printf("Get Humidity... %s\n", Firebase.getDouble(fbdo, F("/test/humidity")) ? String(fbdo.to<double>()).c_str() : fbdo.errorReason().c_str());

       //For the usage of FirebaseJson, see examples/FirebaseJson/BasicUsage/Create_Parse_Edit.ino

    FirebaseJson json;

    if (count == 0)

    {

      json.set("value/round/" + String(count), F("cool!"));

      json.set(F("vaue/ts/.sv"), F("timestamp"));

      Serial.printf("Set json... %s\n", Firebase.set(fbdo, F("/test/json"), json) ? "ok" : fbdo.errorReason().c_str());

    }

    else
```

```
  {
    json.add(String(count), "smart!");
    Serial.printf("Update node... %s\n", Firebase.updateNode(fbdo,
F("/test/json/value/round"), json) ? "ok" : fbdo.errorReason().c_str());
  }
  Serial.println();
 count++;
 }}
```

14.2.4 CODE OF THE WEBAPP

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>ESP Firebase App</title>

  <!-- The core Firebase JS SDK is always required and must be listed first -->
  <script src="https://www.gstatic.com/firebasejs/8.10.0/firebase-app.js"></script>

  <!-- TODO: Add SDKs for Firebase products that you want to use
    https://firebase.google.com/docs/web/setup#available-libraries -->
  <script src="https://www.gstatic.com/firebasejs/8.8.1/firebase-database.js"></script>

  <script>
  // REPLACE WITH YOUR web app's Firebase configuration
  var firebaseConfig = {
    apiKey: "REPLACE_WITH_YOUR_Firebase_CONFIGURATION",
```

```
    authDomain: "REPLACE_WITH_YOUR_Firebase_CONFIGURATION",
    databaseURL: "REPLACE_WITH_YOUR_Firebase_CONFIGURATION",
    projectId: "REPLACE_WITH_YOUR_Firebase_CONFIGURATION",
    storageBucket: "REPLACE_WITH_YOUR_Firebase_CONFIGURATION",
    messagingSenderId: "REPLACE_WITH_YOUR_Firebase_CONFIGURATION",
    appId: "REPLACE_WITH_YOUR_Firebase_CONFIGURATION"
  };
  // Initialize Firebase
  firebase.initializeApp(firebaseConfig);
  var database = firebase.database();
</script>
<script src="app.js" defer></script>

</head>
<body>
  <h1>ESP32 Firebase Web App Example</h1>
  <p>Reading int: <span id="reading-int"></span></p>
  <p>Reading float: <span id="reading-float"></span></p>
</body>
</html>
```

STYLE.CSS

```
html {
  font-family: Verdana, Geneva, Tahoma, sans-serif;
  display: inline-block;
  text-align: center;
}

body {
  margin: 0;
```

```
    width: 100%;
}

.topnav {
    overflow: hidden;
    background-color: #049faa;
    color: white;
    font-size: 1rem;
    padding: 5px;
}

#authentication-bar{
    background-color:mintcream;
    padding-top: 10px;
    padding-bottom: 10px;
}

#user-details{
    color: cadetblue;
}

.content {
    padding: 20px;
}

.card {
    background-color: white;
    box-shadow: 2px 2px 12px 1px rgba(140,140,140,.5);
    padding: 5%;
```

```
}

.cards {
    max-width: 800px;
    margin: 0 auto;
    margin-bottom: 10px;
    display: grid;
    grid-gap: 2rem;
    grid-template-columns: repeat(auto-fit, minmax(200px, 2fr));
}

.reading {
    color: #193036;
}

.date-time{
    font-size: 0.8rem;
    color: #1282A2;
}

button {
    background-color: #049faa;
    color: white;
    padding: 14px 20px;
    margin: 8px 0;
    border: none;
    cursor: pointer;
    border-radius: 4px;
}
```

```
button:hover {
  opacity: 0.8;
}
.deletebtn{
   background-color: #c52c2c;
}

.form-elements-container{
   padding: 16px;
   width: 250px;
   margin: 0 auto;
}

input[type=text], input[type=password] {
   width: 100%;
   padding: 12px 20px;
   margin: 8px 0;
   display: inline-block;
   border: 1px solid #ccc;
   box-sizing: border-box;
}

table {
   width: 100%;
   text-align: center;
   font-size: 0.8rem;
}
tr, td {
   padding: 0.25rem;
```

```
}
tr:nth-child(even) {
    background-color: #f2f2f2
}
tr:hover {
    background-color: #ddd;
}
th {
    position: sticky;
    top: 0;
    background-color: #50b8b4;
    color: white;
}

/* The Modal (background) */
.modal {
    display: none; /* Hidden by default */
    position: fixed; /* Stay in place */
    z-index: 1; /* Sit on top */
    left: 0;
    top: 0;
    width: 100%; /* Full width */
    height: 100%; /* Full height */
    overflow: auto; /* Enable scroll if needed */
    background-color: #474e5d;
    padding-top: 50px;
}

/* Modal Content/Box */
```

```
.modal-content {
    background-color: #fefefe;
    margin: 5% auto 15% auto; /* 5% from the top, 15% from the bottom and centered */
    border: 1px solid #888;
    width: 80%; /* Could be more or less, depending on screen size */
}

/* Style the horizontal ruler */
hr {
    border: 1px solid #f1f1f1;
    margin-bottom: 25px;
}

/* The Modal Close Button (x) */
.close {
    position: absolute;
    right: 35px;
    top: 15px;
    font-size: 40px;
    font-weight: bold;
    color: #f1f1f1;
}

.close:hover,
.close:focus {
    color: #f44336;
    cursor: pointer;
}
```

```
/* Clear floats */
.clearfix::after {
  content: "";
  clear: both;
  display: table;
}

/* Change styles for cancel button and delete button on extra small screens */
@media screen and (max-width: 300px) {
  .cancelbtn, .deletebtn {
    width: 100%;
  }
}
```

AUTH.JS

```
document.addEventListener("DOMContentLoaded", function(){
  // listen for auth status changes
  auth.onAuthStateChanged(user => {
    if (user) {
      console.log("user logged in");
      console.log(user);
      setupUI(user);
      var uid = user.uid;
      console.log(uid);
    } else {
      console.log("user logged out");
      setupUI();
    }
  });
```

```
  // login
  const loginForm = document.querySelector('#login-form');
  loginForm.addEventListener('submit', (e) => {
    e.preventDefault();
    // get user info
    const email = loginForm['input-email'].value;
    const password = loginForm['input-password'].value;
    // log the user in
    auth.signInWithEmailAndPassword(email, password).then((cred) => {
      // close the login modal & reset form
      loginForm.reset();
      console.log(email);
    })
    .catch((error) =>{
      const errorCode = error.code;
      const errorMessage = error.message;
      document.getElementById("error-message").innerHTML = errorMessage;
      console.log(errorMessage);
    });
  });

  // logout
  const logout = document.querySelector('#logout-link');
  logout.addEventListener('click', (e) => {
    e.preventDefault();
    auth.signOut();
  });
});
```

INDEX.JS

```
// convert epochtime to JavaScripte Date object
function epochToJsDate(epochTime){
  return new Date(epochTime*1000);
}

// convert time to human-readable format YYYY/MM/DD HH:MM:SS
function epochToDateTime(epochTime){
  var epochDate = new Date(epochToJsDate(epochTime));
  var dateTime = epochDate.getFullYear() + "/" +
    ("00" + (epochDate.getMonth() + 1)).slice(-2) + "/" +
    ("00" + epochDate.getDate()).slice(-2) + " " +
    ("00" + epochDate.getHours()).slice(-2) + ":" +
    ("00" + epochDate.getMinutes()).slice(-2) + ":" +
    ("00" + epochDate.getSeconds()).slice(-2);

  return dateTime;
}

// function to plot values on charts
function plotValues(chart, timestamp, value){
  var x = epochToJsDate(timestamp).getTime();
  var y = Number (value);
  if(chart.series[0].data.length > 40) {
    chart.series[0].addPoint([x, y], true, true, true);
  } else {
    chart.series[0].addPoint([x, y], true, false, true);
  }
}
```

```
// DOM elements
const loginElement = document.querySelector('#login-form');
const contentElement = document.querySelector("#content-sign-in");
const userDetailsElement = document.querySelector('#user-details');
const authBarElement = document.querySelector('#authentication-bar');
const deleteButtonElement = document.getElementById('delete-button');
const deleteModalElement = document.getElementById('delete-modal');
const deleteDataFormElement = document.querySelector('#delete-data-form');
const viewDataButtonElement = document.getElementById('view-data-button');
const hideDataButtonElement = document.getElementById('hide-data-button');
const tableContainerElement = document.querySelector('#table-container');
const chartsRangeInputElement = document.getElementById('charts-range');
const loadDataButtonElement = document.getElementById('load-data');
const cardsCheckboxElement = document.querySelector('input[name=cards-checkbox]');
const gaugesCheckboxElement = document.querySelector('input[name=gauges-
checkbox]');
const chartsCheckboxElement = document.querySelector('input[name=charts-checkbox]');

// DOM elements for sensor readings
const cardsReadingsElement = document.querySelector("#cards-div");
const gaugesReadingsElement = document.querySelector("#gauges-div");
const chartsDivElement = document.querySelector('#charts-div');
const tempElement = document.getElementById("temp");
const humElement = document.getElementById("hum");
const presElement = document.getElementById("pres");
const updateElement = document.getElementById("lastUpdate")

// MANAGE LOGIN/LOGOUT UI
```

```
const setupUI = (user) => {
  if (user) {
    //toggle UI elements
    loginElement.style.display = 'none';
    contentElement.style.display = 'block';
    authBarElement.style.display ='block';
    userDetailsElement.style.display ='block';
    userDetailsElement.innerHTML = user.email;

    // get user UID to get data from database
    var uid = user.uid;
    console.log(uid);

    // Database paths (with user UID)
    var dbPath = 'UsersData/' + uid.toString() + '/readings';
    var chartPath = 'UsersData/' + uid.toString() + '/charts/range';

    // Database references
    var dbRef = firebase.database().ref(dbPath);
    var chartRef = firebase.database().ref(chartPath);

    // CHARTS
    // Number of readings to plot on charts
    var chartRange = 0;
    // Get number of readings to plot saved on database (runs when the page first loads and
whenever there's a change in the database)
    chartRef.on('value', snapshot =>{
      chartRange = Number(snapshot.val());
      console.log(chartRange);
```

```
    // Delete all data from charts to update with new values when a new range is selected
    chartT.destroy();
    chartH.destroy();
    chartP.destroy();
    // Render new charts to display new range of data
    chartT = createTemperatureChart();
    chartH = createHumidityChart();
    chartP = createPressureChart();
    // Update the charts with the new range
    // Get the latest readings and plot them on charts (the number of plotted readings corresponds to the chartRange value)
    dbRef.orderByKey().limitToLast(chartRange).on('child_added', snapshot =>{
      var jsonData = snapshot.toJSON(); // example: {temperature: 25.02, humidity: 50.20, pressure: 1008.48, timestamp:1641317355}
      // Save values on variables
      var temperature = jsonData.temperature;
      var humidity = jsonData.humidity;
      var pressure = jsonData.pressure;
      var timestamp = jsonData.timestamp;
      // Plot the values on the charts
      plotValues(chartT, timestamp, temperature);
      plotValues(chartH, timestamp, humidity);
      plotValues(chartP, timestamp, pressure);
    });
  });

  // Update database with new range (input field)
  chartsRangeInputElement.onchange = () =>{
JJ    chartRef.set(chartsRangeInputElement.value);
```

```
};

//CHECKBOXES
// Checbox (cards for sensor readings)
cardsCheckboxElement.addEventListener('change', (e) =>{
  if (cardsCheckboxElement.checked) {
    cardsReadingsElement.style.display = 'block';
  }
  else{
    cardsReadingsElement.style.display = 'none';
  }
});
// Checbox (gauges for sensor readings)
gaugesCheckboxElement.addEventListener('change', (e) =>{
  if (gaugesCheckboxElement.checked) {
    gaugesReadingsElement.style.display = 'block';
  }
  else{
    gaugesReadingsElement.style.display = 'none';
  }
});
// Checbox (charta for sensor readings)
chartsCheckboxElement.addEventListener('change', (e) =>{
  if (chartsCheckboxElement.checked) {
    chartsDivElement.style.display = 'block';
  }
  else{
    chartsDivElement.style.display = 'none';
  }
```

```
});

// CARDS
// Get the latest readings and display on cards
dbRef.orderByKey().limitToLast(1).on('child_added', snapshot =>{
  var jsonData = snapshot.toJSON(); // example: {temperature: 25.02, humidity: 50.20, pressure: 1008.48, timestamp:1641317355}
  var temperature = jsonData.temperature;
  var humidity = jsonData.humidity;
  var pressure = jsonData.pressure;
  var timestamp = jsonData.timestamp;
  // Update DOM elements
  tempElement.innerHTML = temperature;
  humElement.innerHTML = humidity;
  presElement.innerHTML = pressure;
  updateElement.innerHTML = epochToDateTime(timestamp);
});

// GAUGES
// Get the latest readings and display on gauges
dbRef.orderByKey().limitToLast(1).on('child_added', snapshot =>{
  var jsonData = snapshot.toJSON(); // example: {temperature: 25.02, humidity: 50.20, pressure: 1008.48, timestamp:1641317355}
  var temperature = jsonData.temperature;
  var humidity = jsonData.humidity;
  var pressure = jsonData.pressure;
  var timestamp = jsonData.timestamp;
  // Update DOM elements
  var gaugeT = createTemperatureGauge();
```

```
  var gaugeH = createHumidityGauge();
  gaugeT.draw();
  gaugeH.draw();
  gaugeT.value = temperature;
  gaugeH.value = humidity;
  updateElement.innerHTML = epochToDateTime(timestamp);
});

// DELETE DATA
// Add event listener to open modal when click on "Delete Data" button
deleteButtonElement.addEventListener('click', e =>{
  console.log("Remove data");
  e.preventDefault;
  deleteModalElement.style.display="block";
});

// Add event listener when delete form is submited
deleteDataFormElement.addEventListener('submit', (e) => {
  // delete data (readings)
  dbRef.remove();
});

// TABLE
var lastReadingTimestamp; //saves last timestamp displayed on the table
// Function that creates the table with the first 100 readings
function createTable(){
  // append all data to the table
  var firstRun = true;
  dbRef.orderByKey().limitToLast(100).on('child_added', function(snapshot) {
```

```
      if (snapshot.exists()) {
        var jsonData = snapshot.toJSON();
        console.log(jsonData);
        var temperature = jsonData.temperature;
        var humidity = jsonData.humidity;
        var pressure = jsonData.pressure;
        var timestamp = jsonData.timestamp;
        var content = '';
        content += '<tr>';
        content += '<td>' + epochToDateTime(timestamp) + '</td>';
        content += '<td>' + temperature + '</td>';
        content += '<td>' + humidity + '</td>';
        content += '<td>' + pressure + '</td>';
        content += '</tr>';
        $('#tbody').prepend(content);
        // Save lastReadingTimestamp --> corresponds to the first timestamp on the returned
snapshot data
        if (firstRun){
          lastReadingTimestamp = timestamp;
          firstRun=false;
          console.log(lastReadingTimestamp);
        }
      }
    });
  };


  // append readings to table (after pressing More results... button)
  function appendToTable(){
    var dataList = []; // saves list of readings returned by the snapshot (oldest-->newest)
```

```
    var reversedList = []; // the same as previous, but reversed (newest--> oldest)
    console.log("APEND");
    dbRef.orderByKey().limitToLast(100).endAt(lastReadingTimestamp).once('value', function(snapshot) {
      // convert the snapshot to JSON
      if (snapshot.exists()) {
        snapshot.forEach(element => {
          var jsonData = element.toJSON();
          dataList.push(jsonData); // create a list with all data
        });
        lastReadingTimestamp = dataList[0].timestamp; //oldest timestamp corresponds to the first on the list (oldest --> newest)
        reversedList = dataList.reverse(); // reverse the order of the list (newest data --> oldest data)

        var firstTime = true;
        // loop through all elements of the list and append to table (newest elements first)
        reversedList.forEach(element =>{
          if (firstTime){ // ignore first reading (it's already on the table from the previous query)
            firstTime = false;
          }
          else{
            var temperature = element.temperature;
            var humidity = element.humidity;
            var pressure = element.pressure;
            var timestamp = element.timestamp;
            var content = '';
            content += '<tr>';
            content += '<td>' + epochToDateTime(timestamp) + '</td>';
```

```
        content += '<td>' + temperature + '</td>';
        content += '<td>' + humidity + '</td>';
        content += '<td>' + pressure + '</td>';
        content += '</tr>';
        $('#tbody').append(content);
       }
      });
    }
  });
}

viewDataButtonElement.addEventListener('click', (e) =>{
  // Toggle DOM elements
  tableContainerElement.style.display = 'block';
  viewDataButtonElement.style.display ='none';
  hideDataButtonElement.style.display ='inline-block';
  loadDataButtonElement.style.display = 'inline-block'
  createTable();
});

loadDataButtonElement.addEventListener('click', (e) => {
  appendToTable();
});

hideDataButtonElement.addEventListener('click', (e) => {
  tableContainerElement.style.display = 'none';
  viewDataButtonElement.style.display = 'inline-block';
  hideDataButtonElement.style.display = 'none';
});
```

```
  // IF USER IS LOGGED OUT
  } else{
    // toggle UI elements
    loginElement.style.display = 'block';
    authBarElement.style.display ='none';
    userDetailsElement.style.display ='none';
    contentElement.style.display = 'none';
  }
}
```

CHART-DEFINITION.JS

```
// Create the charts when the web page loads
window.addEventListener('load', onload);

function onload(event){
  chartT = createTemperatureChart();
  chartH = createHumidityChart();
  chartP = createPressureChart();
}

// Create Temperature Chart
function createTemperatureChart() {
  var chart = new Highcharts.Chart({
    chart:{
      renderTo:'chart-temperature',
      type: 'spline'
    },
    series: [
      {
```

```
      name: 'BME280'
    }
  ],
  title: {
    text: undefined
  },
  plotOptions: {
    line: {
      animation: false,
      dataLabels: {
        enabled: true
      }
    }
  },
  xAxis: {
    type: 'datetime',
    dateTimeLabelFormats: { second: '%H:%M:%S' }
  },
  yAxis: {
    title: {
      text: 'Temperature Celsius Degrees'
    }
  },
  credits: {
    enabled: false
  }
 });
 return chart;
}
```

```
// Create Humidity Chart
function createHumidityChart(){
  var chart = new Highcharts.Chart({
    chart:{
      renderTo:'chart-humidity',
      type: 'spline'
    },
    series: [{
      name: 'BME280'
    }],
    title: {
      text: undefined
    },
    plotOptions: {
      line: {
        animation: false,
        dataLabels: {
          enabled: true
        }
      },
      series: {
        color: '#50b8b4'
      }
    },
    xAxis: {
      type: 'datetime',
      dateTimeLabelFormats: { second: '%H:%M:%S' }
    },
```

```
    yAxis: {
      title: {
        text: 'Humidity (%)'
      }
    },
    credits: {
      enabled: false
    }
  });
  return chart;
}

// Create Pressure Chart
function createPressureChart() {
  var chart = new Highcharts.Chart({
    chart:{
      renderTo:'chart-pressure',
      type: 'spline'
    },
    series: [{
      name: 'BME280'
    }],
    title: {
      text: undefined
    },
    plotOptions: {
      line: {
        animation: false,
        dataLabels: {
```

```
        enabled: true
      }
    },
    series: {
      color: '#A62639'
    }
  },
  xAxis: {
    type: 'datetime',
    dateTimeLabelFormats: { second: '%H:%M:%S' }
  },
  yAxis: {
    title: {
      text: 'Pressure (hPa)'
    }
  },
  credits: {
    enabled: false
  }
 });
 return chart;
}
```

GUAGE-DEFINITION.JS

```
// Create Temperature Gauge
function createTemperatureGauge() {
  var gauge = new LinearGauge({
    renderTo: 'gauge-temperature',
    width: 120,
    height: 400,
```

```
units: "Temperature C",
minValue: 0,
startAngle: 90,
ticksAngle: 180,
maxValue: 40,
colorValueBoxRect: "#049faa",
colorValueBoxRectEnd: "#049faa",
colorValueBoxBackground: "#f1fbfc",
valueDec: 2,
valueInt: 2,
majorTicks: [
    "0",
    "5",
    "10",
    "15",
    "20",
    "25",
    "30",
    "35",
    "40"
],
minorTicks: 4,
strokeTicks: true,
highlights: [
    {
        "from": 30,
        "to": 40,
        "color": "rgba(200, 50, 50, .75)"
    }
```

```
        ],
        colorPlate: "#fff",
        colorBarProgress: "#CC2936",
        colorBarProgressEnd: "#049faa",
        borderShadowWidth: 0,
        borders: false,
        needleType: "arrow",
        needleWidth: 2,
        needleCircleSize: 7,
        needleCircleOuter: true,
        needleCircleInner: false,
        animationDuration: 1500,
        animationRule: "linear",
        barWidth: 10,
    });
    return gauge;
}
// Create Humidity Gauge
function createHumidityGauge(){
    var gauge = new RadialGauge({
        renderTo: 'gauge-humidity',
        width: 300,
        height: 300,
        units: "Humidity (%)",
        minValue: 0,
        maxValue: 100,
        colorValueBoxRect: "#049faa",
        colorValueBoxRectEnd: "#049faa",
        colorValueBoxBackground: "#f1fbfc",
```

```
valueInt: 2,
majorTicks: [
    "0",
    "20",
    "40",
    "60",
    "80",
    "100"

],
minorTicks: 4,
strokeTicks: true,
highlights: [
    {
        "from": 80,
        "to": 100,
        "color": "#03C0C1"
    }
],
colorPlate: "#fff",
borderShadowWidth: 0,
borders: false,
needleType: "line",
colorNeedle: "#007F80",
colorNeedleEnd: "#007F80",
needleWidth: 2,
needleCircleSize: 3,
colorNeedleCircleOuter: "#007F80",
needleCircleOuter: true,
```

```
        needleCircleInner: false,
        animationDuration: 1500,
        animationRule: "linear"
    });
    return gauge;
}
```

SNAPSHOTS

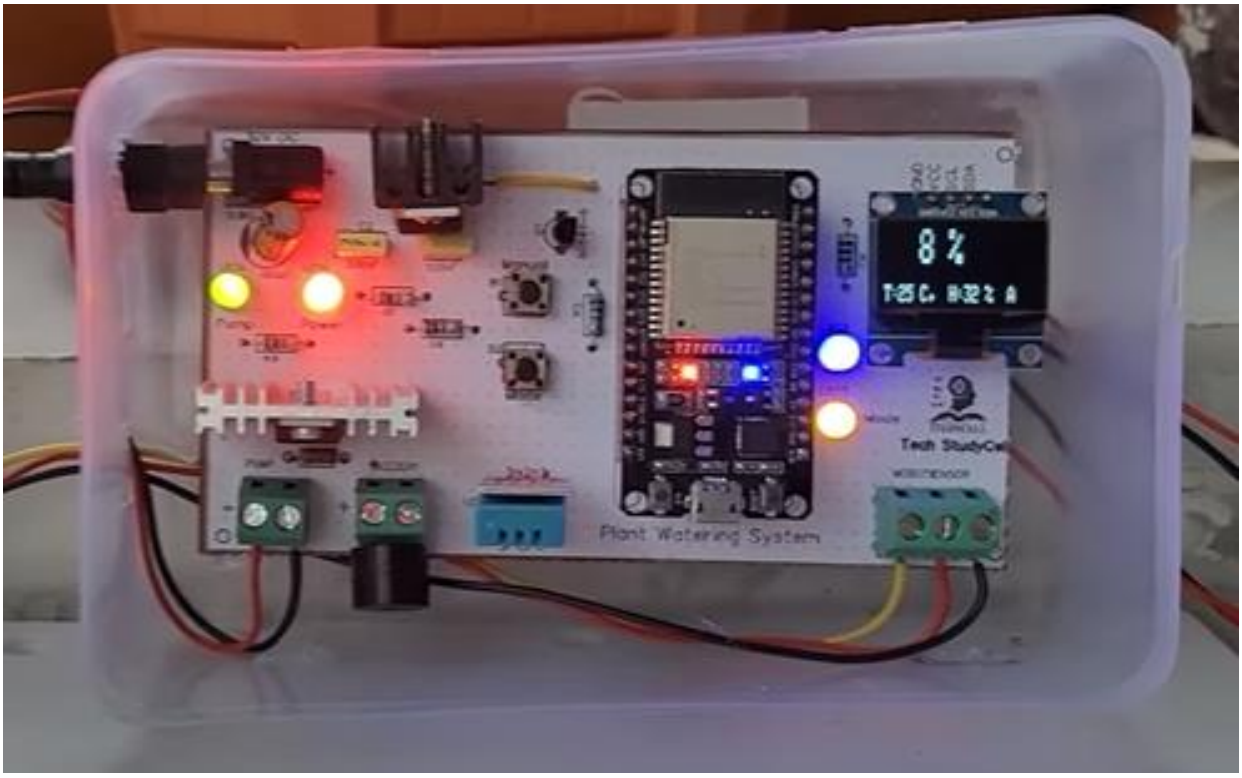

Figure 40 SNAPSHOT OF THE HARDWARE

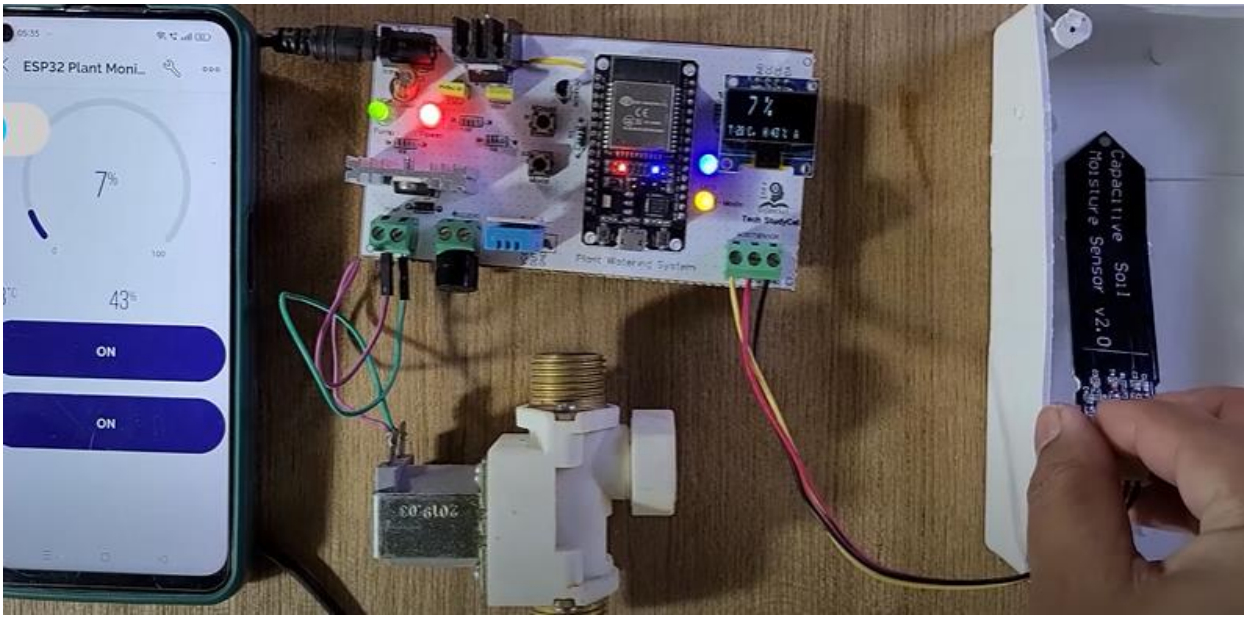

Figure 41COMPLETE MODEL

SUMMARY

The IoT-aided irrigation system project aimed to develop a smart and efficient irrigation system for agricultural purposes by leveraging the capabilities of the Internet of Things (IoT). The project focused on optimizing water usage, improving crop yield, and promoting sustainable agricultural practices.

The project involved the integration of various components, including sensors, microcontrollers, communication modules, and a centralized control system. Soil moisture sensors were deployed in the field to collect real-time data about soil moisture levels. These sensors were connected to microcontrollers, such as Arduino or ESP32, which transmitted the data to the central control system using wireless communication protocols like Wi-Fi or LoRa.

The central control system, implemented using a Raspberry Pi or a cloud-based platform, received and processed the sensor data. It applied intelligent algorithms and decision-making models to determine the optimal irrigation schedule and water requirements for different crops based on factors like soil moisture, weather conditions, and crop type. The system also incorporated weather forecasting data to adapt irrigation schedules accordingly.

The project demonstrated significant improvements in water efficiency, ensuring that crops received adequate irrigation while avoiding overwatering. By automating the irrigation process and optimizing water usage, the system helped farmers reduce water wastage, conserve resources, and lower operational costs.

Furthermore, the project focused on user-friendly interfaces for monitoring and controlling the irrigation system. A mobile application or web-based dashboard provided real-time updates on soil moisture levels, irrigation schedules, and alerts for abnormal conditions. This enabled farmers to remotely monitor the system, make adjustments as needed, and receive notifications about critical events.

The project's implementation and testing phase involved deploying the IoT-aided irrigation system in real-world agricultural settings, monitoring its performance, and collecting

feedback from farmers. The results showed improved crop health, increased yield, and reduced water consumption compared to traditional irrigation methods.

In conclusion, the IoT-aided irrigation system project successfully developed a smart irrigation solution that integrated IoT technologies, data analytics, and intelligent decision-making algorithms. The system demonstrated its potential to enhance water efficiency, optimize crop irrigation, and promote sustainable agricultural practices. The project's findings and outcomes provide valuable insights for future research and implementation of similar IoT-based irrigation systems in the agricultural sector.

PART II

CHAPTER 15 OVERVIEW

As the global population continues to grow, there is an ever-increasing demand for food production to keep pace with the needs of society. Unfortunately, plant diseases pose a significant threat to agricultural productivity by affecting the yields and quality of crops. These diseases can cause significant losses for farmers and food producers, resulting in increased prices or even food shortages.

Traditionally, the identification and control of plant diseases have been time-consuming and labor-intensive processes. Trained professionals have had to rely on years of experience and extensive knowledge to diagnose and manage these diseases. This has often resulted in delays in identifying and treating problems, which can have a significant impact on crop yields and quality.

The damage and costs of plant diseases go beyond the economic loss to farmers and food producers. Environmental impact, human health consequences, and social and economic disruptions can also arise. With the emergence of modern technologies, it is time to focus on innovative solutions to detect and control plant diseases.

Our project aims to showcase one such solution, which utilizes advanced machine learning techniques to classify plant diseases with high accuracy while aiding diagnose, treat, and manage them effectively. In addition to classifying plant diseases, our application has the capability to analyze a vast amount of data and provide users with useful insights in real-time.

To address these challenges, our team developed an innovative solution that utilizes machine learning techniques and artificial intelligence to classify 38 different plant diseases. Our application provides farmers, agronomists, and researchers with a powerful tool that can quickly and accurately identify plant diseases and provide appropriate recommendations for management and control. By leveraging these advanced technologies, we believe that we can significantly improve global food production and help address some of the biggest challenges facing society today.

To overcome these challenges, we developed an application that utilizes machine learning techniques and artificial intelligence to classify plant diseases. The application can accurately identify 38 different plant diseases and provide appropriate measures to control and manage them.

The disease classification model was developed using Artificial Neural Network (ANN) and achieved an accuracy rate of over 90%. In addition, the application features an Open AI-powered Language Model (LLM) that aids disease diagnosis, treatment, and management.

Our project's aim was to develop a solution that could help farmers, agronomists, and researchers to quickly identify and control plant diseases, leading to higher crop yields and improved food production.

This project has the potential to revolutionize the agriculture industry by offering an easy-to-use and affordable tool that can be utilized by growers globally to help achieve sustainable food production. Moreover, the project's use of open-source tools ensures that our work is replicable and scalable, enabling a more significant impact across broader geographic areas.

We firmly believe that the application we have developed has the potential to deliver real benefits to the agriculture industry and overcome the challenges posed by plant diseases. By improving crop health and productivity, we hope that our solution can play a small role in contributing positively to global food security.

CHAPTER 16 LITERARATURE SURVEY II

Plant disease detection using computer vision and machine learning techniques has gained considerable attention in recent years due to its potential to improve crop yields and food security. A wide range of approaches have been proposed, relying on traditional image processing techniques as well as machine learning algorithms. While earlier works focused on detecting plant diseases based on leaf color, shape, and texture, recent studies have shifted towards capturing symptoms at the pixel level using deep learning models. Meanwhile, conversational agents or chatbots have emerged as a promising solution to improve the accessibility of information related to plant diseases, providing users with personalized and timely advice. In this section, we summarize existing works related to plant disease detection and conversational agents, highlighting their strengths and weaknesses, and providing context for our own contributions.

There were when many such research and applications which were already built regarding the subject but all of them generally covered the machine learning model development part which was trained on a huge set of images containing infected plants and a few of those resources that we came across while developing our application or given below.

- One such application was created by Ameya Upalanchi, who creatively utilized the power of TensorFlow framework to develop a model with the help of a convolutional neural network. The model was then embedded on an open-source application created using a Python library called Streamlit. This was my first inspiration on how to portray my idea and provide a better user interface, as well as how to combine and embed machine learning models and a user interface so that users could access the model's abilities.
- While exploring resources, I came across a similar resource based on the same idea, but the way that it utilized the application framework and integrated the machine learning model was completely different. This was an innovative way for me to gain insight into creating my own architecture.
- Another powerful resource I came across was a real project deployed for a similar use case, which was built using TensorFlow and a React application. This project was

deployed for user interaction for production purposes, and the developer creatively utilized both the machine learning model and web development framework.

While referencing all these resources, I was able to build my own application that embodies these concepts.

In addition to the various application-based resources, I also looked through some research papers to gain insights into creating my own machine learning model. These papers provided valuable information and data sets for creating machine learning models, including training data. The resources referred to by these papers were extremely valuable and helped in creating a robust machine learning model.

My application is divided into two significant parts - identifying plant diseases and providing a user interface for humans to interact with the application. Once the disease is identified, the user interface helps in designing an appropriate treatment plan for the plants, ensuring they thrive.

Incorporating large language models into the application became necessary to provide an intelligent and interactive experience for users. For this purpose, I integrated OpenAI's large language model with the help of an API into the application. This integration allowed me to create a more human-like assistant that could provide one-on-one interaction with the user, helping them get more insights on how to treat the plant after identifying the disease. With this integration, the assistant can recommend various effective and suitable treatment options that can help the plant recover quickly from the identified disease.

Integrating a large language model into my application was no easy feat. I had to go through several applications or sample applications that have been built since the time ChatGPT was first released. Through my research, I analyzed and built more sample applications that incorporate large language models. This research allowed me to create an application that uses OpenAI's large language model, with all the necessary knowledge incorporated into it.

I will provide links to these resources later, which can be useful for anyone looking to gain insights on integrating a large language model into their application.

All the research and development eventually culminated in the final output - the application that I aimed to create. My main objective was to provide humans or users with one-on-one interaction with someone who has information regarding the disease and use this interaction for the benefit of reducing errors in the farming process and increase yield throughout the growth of the crop.

There were countless resources and research papers that I had to go through to build this application. These resources provided me with valuable insights and gave me the necessary knowledge to create an application that not only identifies the plant disease but also provides suitable treatment options to help the plant recover. With all these in mind, I created the application with the sole purpose of helping farmers around the world to reap the benefits of modern agriculture.

CHAPTER 17 PROBLEM STATEMENT

Plant diseases have always posed a significant threat to agricultural productivity, affecting crop yields and quality, posing significant environmental threats, and subsequent social and economic disruptions. Identification and control of plant diseases have been a longstanding challenge for farmers, researchers, and agronomists. Traditional methods of plant disease identification are time-consuming, labor-intensive, and require trained professionals. These challenges result in farmers suffering from reduced yields and increased losses, food producers having to face financial loss, and consumers having to endure increased prices, and even food scarcities.

The traditional methods of plant disease identification have several limitations that include the difficulty of accurately identifying a specific disease, the lack of objective criteria in assessing disease presence, the expense and time required to hire trained experts, and the manual effort required to monitor the disease continuously. Furthermore, traditional disease identification/management methods present issues related to scale and disparity, leaving scores of farmers with insufficient or no access to knowledge concerning diseases, resulting in significant losses in the food supply chain.

To address these challenges efficiently, our application employs cutting-edge machine learning methods and artificial intelligence to solve the challenges posed by plant diseases, aiding diagnoses and managing them accurately and effectively. By utilizing our innovative solution, users can prevent crop damage and boost yields, leading to improved food production and security.

CHAPTER 18 PROPOSED METHODOLOGY

Our machine learning-powered plant disease classification application follows a three-stage approach: data acquisition, data preprocessing, and training the classification models.

- **Data acquisition**

Our dataset consisted of a collection of diseased plant images acquired from multiple sources, including open-source repositories and partnerships with plant pathology research labs. The dataset included images of 38 unique plant diseases.

- **Data preprocessing**

The data was thoroughly preprocessed through various transformations, including normalization, resizing, and augmentation. We introduced various types of noise in the data, simulated environmental variance and applied different styles of image processing, and mixing. Before inputting the images for the classifier, we performed color space conversion from Red Green Blue (RGB) to Hue Saturation Value (HSV).

- **Training the classification models**

The input images were classified as being diseased or healthy using a binary classification model, followed by predicting the disease present in the image using a multi-classifier model. We trained and tested the model on an GPU on cloud and used the categorical cross-entropy loss function to calculate the loss during the training process.

Evaluation of the classification model was conducted on a separate test dataset, which is distinct from the training dataset. Performance metrics of the classifier were assessed on metrics such as Precision, Recall, and F1-score to evaluate the model's capabilities better.

Further, we integrated our classification model with the GPT Language model (Open AI); this integration allowed our application to provide relevant information about the identified plant diseases. Using the Open AI API, the application provides users with adequate information to help diagnose, treat and manage plant diseases. We integrated our classification model with ChromaDB, an open-source document-based database system, to store and manage relevant information about plant diseases that can be useful for further analysis by the Language model. This information includes scientific research papers, field

notes, recommended treatments, and other relevant data. We deployed our application interface using Streamlit, an open-source Python framework for building web applications.

Our approach encompasses data acquisition, data preprocessing, training and evaluation of a Classification model's performance, deploying it using Streamlit and integrating it with ChromaDB to provide contextualized data for easy visualizations, insights, and recommendations while using the Language Model to assists the users on plant disease diagnosis, treatment, and management.

So, on a high level my whole application is divided into two functionalities:

- One is where my application is identifying the disease which is present in the plant specimen.
- The other is a large language model which provides human-like interaction with the user so that user can look for assistance and treatment regarding his requirements.

In the next few chapters, we will be explaining you these two parts in depth.

PROPOSED ARCHITECTURE

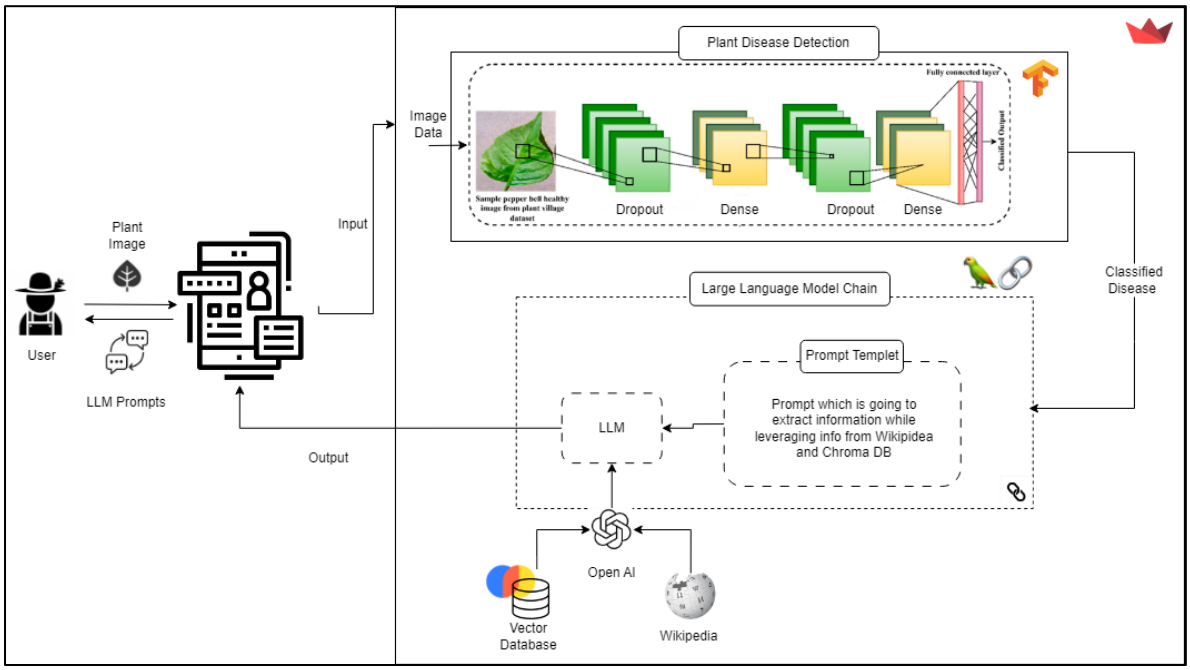

Figure 42 Schematic Diagram of the Proposed Architecture.

CHAPTER 19 SOFTWARE AND HARDWARE REQUIREMENTS

In this section, we will outline the software and hardware requirements which were used while building this project and are also required for running this project. Having a clear understanding of these requirements is essential to ensure the software operates efficiently and effectively. We will detail the minimum specifications for the hardware and software, as well as recommended specifications for optimum performance. This information will be useful for users who want to ensure they have adequate hardware and software before installing the software or application.

For more information on environment setup refer to Appendix I.

The following software and hardware requirements are recommended for running the machine learning project developed using TensorFlow, Python, LangChain, and Streamlit with GPU acceleration:

19.1 Software Requirements

- Python 3.6 or higher
- TensorFlow 2.0 or higher
- LangChain
- Streamlit
- Chromadb
- CUDA 10 or higher (if using NVIDIA GPU)
- cuDNN (if using NVIDIA GPU)

And many more are there, which will be present with the project repository.

19.2 Hardware Requirements

- NVIDIA GPU with a minimum of 4 GB of VRAM
- A CPU with at least 4 cores and 4 threads or higher
- At least 8 GB of RAM or higher

- At least 100 GB of available disk space

Note: The specific software and hardware requirements may vary depending on the size of the data, the complexity of the machine learning models, and the specific use case. It is recommended to refer to the documentation and guidelines provided by each library and to consult with the hardware and software vendor for more detailed requirements.

These requirements should be considered as a guideline for the recommended hardware and software required to run the machine learning project. Adjustments may be necessary depending on the specific use case and the size and complexity of the data being used. It is important to review the documentation and recommendations provided by the libraries used and the hardware and software vendors to ensure optimal performance of the machine learning project.

CHAPTER 20 PLANT DISEASE CLASSIFICATION MODEL

Image classification is a technique used in computer vision to classify an image into one or more predefined categories based on the image content. This is done by extracting relevant features from the image and training a machine learning model on a dataset of labeled images to recognize patterns and make accurate predictions.

20.1 USAGE OF IMAGE CLASSIFICATION

In the context of plant disease classification, image classification can be used to automatically detect and diagnose plant diseases based on visual symptoms captured in images of leaves, stems, or other plant parts. This can have several benefits, including:

- Early detection of plant diseases: By automatically analyzing images, plant diseases can be detected early before they develop into full-blown infestations.
- Quick and accurate diagnosis: Image classification models can make diagnoses on a large scale, providing accurate results in a matter of seconds and reducing the need for time-consuming manual inspections.
- Reduced pesticide use: With early detection and accurate diagnoses, farmers can reduce their use of pesticides, saving money and reducing the negative impact of agriculture on the environment.

20.2 PROCESS OF IMAGE CLASSIFICATION

- Collecting a dataset of images: This dataset should include images of healthy plants as well as plants affected by different diseases.
- Image pre-processing: The images need to be normalized and preprocessed to ensure consistent quality and clarity.
- Feature extraction: Features are extracted from images by identifying patterns, textures, and colors that can be used to distinguish healthy plants from those affected by diseases.

- Training a model: A machine learning model is trained using labeled images to learn how to accurately classify plants as healthy or diseased.
- Validation: The trained model is tested on a validation set of images to ensure its accuracy and improve it if necessary

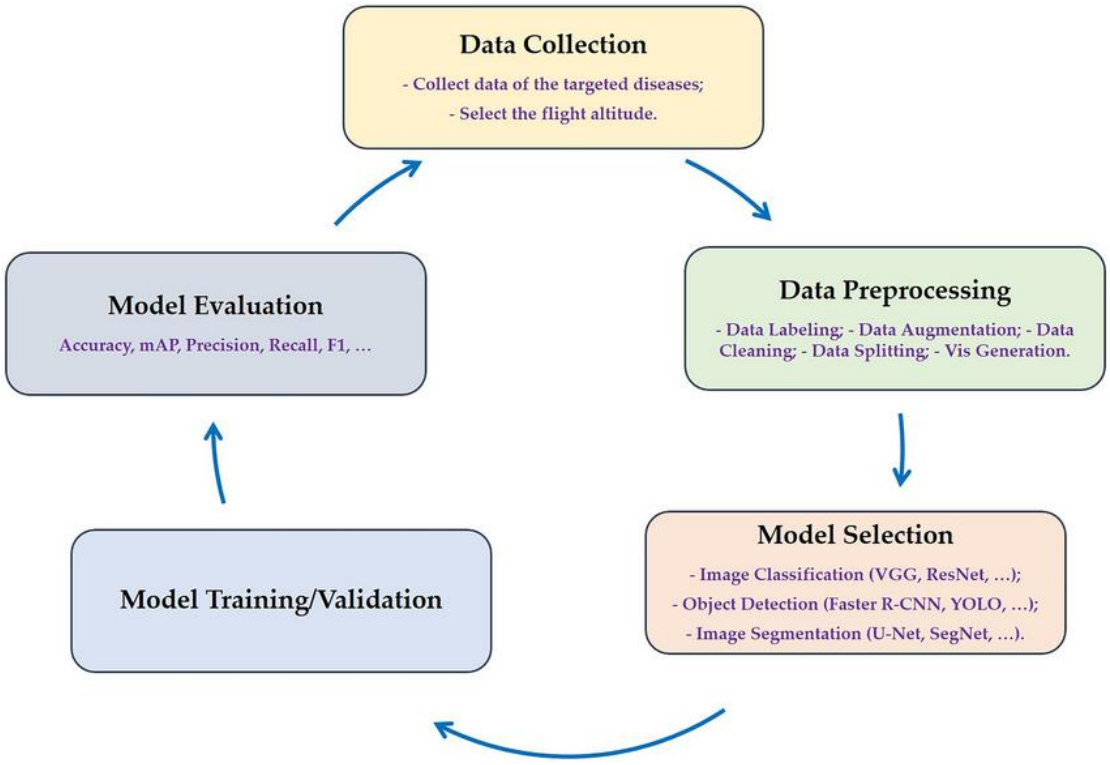

Figure 43 Machine Learning Lifecycle for plant disease detection

20.3 DATASET

The PlantVillage dataset is a collection of images of diseased and healthy plant leaves and the associated metadata such as crop type, disease type, and severity. It was created by researchers at Penn State University to aid in the development of machine learning algorithms to detect plant diseases.

The dataset contains over 54,000 images of plant leaves from 26 different crop species, including tomatoes, potatoes, apples, and grapes. These images are divided into three categories: healthy, diseased, and diseased with a severe symptom. The plant diseases represented in the dataset include bacterial spot, late blight, and powdery mildew, among others.

Each image in the dataset comes with detailed metadata to provide context and additional information for machine learning algorithms. This metadata includes information about the crop type, disease type, and severity. Additionally, each image has been labeled by expert plant pathologists to ensure accuracy and consistency across the dataset.

The PlantVillage dataset is a valuable resource for researchers and developers working on the detection and diagnosis of plant diseases using machine learning algorithms. It provides a comprehensive collection of high-quality images and metadata that can be used to train and test algorithms for accuracy and effectiveness.

Overall, image classification can be a powerful tool for plant disease classification as it allows farmers and researchers to identify and manage plant diseases, leading to healthier crops and better yields quickly and accurately.

Examples of the images and different phenotypes are given in Table 1.

	Fungi	**Bacteria**	**Mold**	**Virus**	**Mite**	**Healthy**
Apple (3172)	*Gymnosporangium juniperi-virginianae (276)* *Venturia insequalis (630)* *Botryospaeria obtuse (621)*					**(1645)**
Blueberry (1502)						**(1502)**
Cherry (1906)	*Podosphaera spp (1052)*					**(854)**
Corn (3852)	*Cercospora zeae-maydis (513)* *Puccinia sorghi (1192)* *Exserohilum turcicum (985)*					**(1162)**
Grape (4063)	*Guignardia bidwellii (1180)* *Phaeomoniella--spp. (1384)* *Pseudocercospor a vitis (1076)*					**(423)**
Orange (5507)	*Candidatus Liberibacter (5507)*					
Peach (2657		*Xanthomonas campestris (2291)*				**(360)**
Bell Pepper (2475)		*Xanthomonas campestris (997)*				**(1478)**
Potato (2152)	*Alternaria solani (1000)*		*Phytophthora Infestans (1000)*			**(152)**

Raspberry (371)						**(371)**
Soybean (5090)						**(5090)**
Squash (1835)	*Erysiphe cichoracearum / Sphaerotheca fuliginea (1835)*					
Strawberry (1565)	*Diplocarpon earlianum (1109)*					**(456)**
Tomato (18,162)	*Alternaria solani (1000)* *Septoria lycopersici (1771)* *Corynespora cassiicola (1404)* *Fulvia fulva (952)*	*Xanthomonas campestris pv. Vesicatoria (2127)*	*Phytophthora Infestans (1910)*	*Tomato Yello Leaf Curl Virus (5357)* *Tomato Mosaic Virus (373)*	*Tetranychus urticae (1676)*	**(1592)**

Table 1 Description of images present in the dataset

This whole data set contains various classes of diseases for a different types of plants and even in a separate plant there are different category of diseases that might affect the plant this is just a general data set which is for everyone for research purposes this is definitely not something that you can put to production or you can just provide it to someone so this is something which can help me or anyone to build a base application on the top of which other images can be inserted for training the model and making it available for a more different kind of use cases.

Some sample images are also present below.

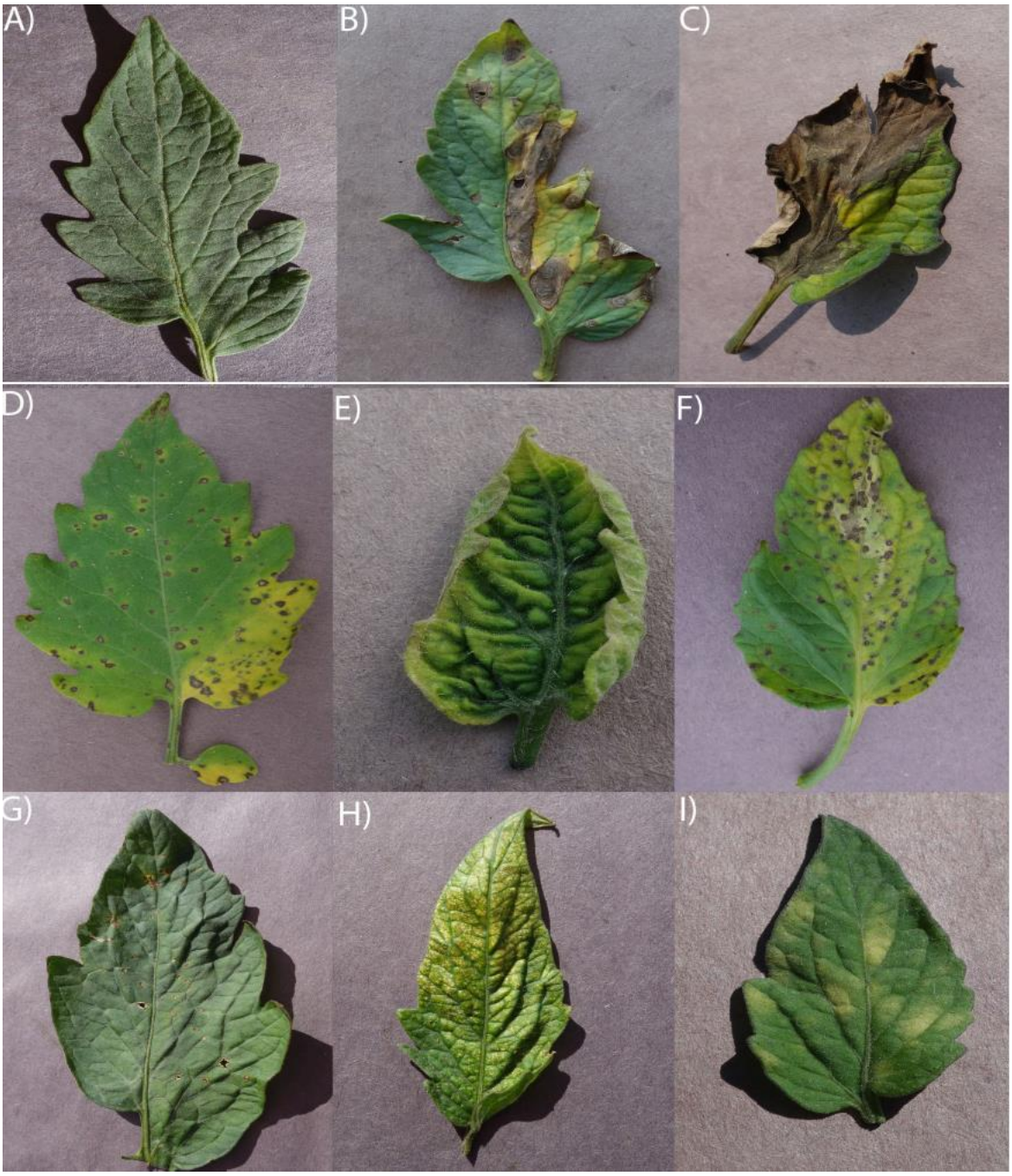

Figure 44 Examples of different phenotypes of tomato plants. A) Healthy leaf B) Early Blight C) Late Blight (Phytophthora Infestans) D) Septoria Leaf Spot (Septorialycopersici) E) Yellow Leaf Curl Virus (Family Geminiviridae genus Begomovirus) F) BacterialSpot (Xanthomonas campestris pv. vesicatoria) G) Target Spot (Corynespora cassiicola) H) Spider Mite (Tetranychus urticae) Damage

This data set provides us with very helpful information so now it's time to import the data set into our environment so that we can work with it perform different transformation on data according to our requirements.

Dataset can be found on this URL:

https://storage.googleapis.com/plantdata/PlantVillage.zip

Now what we're going to do we are going to load out data set into our environment so that we can use it for training a model testing a model and perform validation on it. But before doing so we need to import the required libraries to perform all the actions that we are going to perform throughout this process.

```
from __future__ import absolute_import, division, print_function, unicode_literals


import tensorflow as tf
#tf.logging.set_verbosity(tf.logging.ERROR)
#tf.enable_eager_execution()

import tensorflow_hub as hub
import os
from tensorflow.keras.layers import Dense, Flatten, Conv2D
from tensorflow.keras import Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import layers
#from keras import optimizers
```

Here we are loading the required libraries for the performing all kind of operations or a data so that we can train our model to achieve our desired output, one such library is tensor flow detail about this library and including other libraries will be provided in the appendix I please refer to it in case if you are interested in what these libraries are or how I have set up this whole environment. When's the importing of libraries is completed it's time to download the data set and perform some splitting or creating training set and validation set.

```
# Loading Data

zip_file = tf.keras.utils.get_file(origin='https://storage.googleapis.com/plantdata/PlantVillage.zip',
                                   fname='PlantVillage.zip', extract=True)
```

The above snippet of code is a going to download my PlantVillage data from are you URL which is mentioned in the code, and I am using uh Tensorflow Keras library for getting that file from the given URL.

When the data is downloaded now it is time to create validation training and test data set meaning what we are going to do we are going to divide our whole image data set which contains around 54,000 number of images into three segments the idea of dividing the whole data set into three segments is that when we need to train our whole model we need our data

set which can be used to to or tune the parameters of our machine learning model so for that we are going to use a training data set and next what we need we need to check if the training has been performed according to the requirement meaning if the model is fulfilling the requirements or not for that what we does is we create a small validation test which is used for just validating that the required accuracy of our model is there or not and finally what we do is we test on machine learning model on an unseen data set which is referred to as test data set so this is the ideology of dividing a whole data set into three separate parts generally what are the best practices are in training data set and test data set splitting we generally divided into the ratio of 80% of training data set and 20% of training data set then further we divide our training data set into two separate parts of which a small part will be given to validation set and the rest will be used for training of our data model.

```
# Prepare training and validation dataset

data_dir = os.path.join(os.path.dirname(zip_file), 'PlantVillage')
train_dir = os.path.join(data_dir, 'train')
validation_dir = os.path.join(data_dir, 'validation')
```

Here we are defining the two directories of our train and validation data set. Now to perform the splitting there are various ways to do that one way is to just call a function from the library and another that I'm using is just creating a function which can perform the splitting for me so here the snippet of the code which is performing that operation.

```
import time
import os
from os.path import exists

def count(dir, counter=0):
    "returns number of files in dir and subdirs"
    for pack in os.walk(dir):
        for f in pack[2]:
            counter += 1
    return dir + " : " + str(counter) + "files"
```

```
print('total images for training :', count(train_dir))
print('total images for validation :', count(validation_dir))
```

As we can see the function is going to take a directory as an argument and then perform the division or you can say splitting of the whole number of images which is present in a particular directory and then later snippet what I have done I have printed the number of images which are going to be used for training and validation while calling the function which

is defined above so the output of the print function is this which is depicting the number of images which are assigned for training and validation set.

```
total images for training : /root/.keras/datasets/PlantVillage/train : 43444files
total images for validation : /root/.keras/datasets/PlantVillage/validation : 10861files
```

As you can see there are around 43,000 files which are present for training our machine learning model now what we need next is the classes we have in our data set as the kind of learning we are using in this application is supervised learning. For more information on different kinds of learning and an overview of machine learning and deep learning please refer to the appendix we have provided at the end of this documentation.

Now uh as I mentioned before we are using supervised learning the idea of supervised learning is that we have some features and labels what are features? Features are nothing but the input variable that our model is going to take and labels are the output that our model is supposed to be given so basically in supervised learning we train our model to adapt to the relationship between the input variables that our model should receive and the output that it should give there is a mathematical model which is going to be created by performing various iterations over the data set by performing different kinds of shuffles so that the model can understand what is the relationship between input and output and can adapt to it can learn it and after learning it model will be able to perform those classifications without going through the same process again this kind of learning is referred to as a supervised learning so now what our aim is we have to provide different classes to which we want our model to pertain input variable of our application will be images and what we want our model is to perform a classification depending on the disease present in the image.

```
categories = list(train_data.class_indices.keys())
print(train_data.class_indices)
```

In the above snippet what we are doing is we are creating classes for the diseases that are present in our data set this is one way of doing it where we are just extracting the different classes we have present in our data set another way that I have those in my application development is just extracting it from a GitHub repo which already contain adjacent format category files containing all 38 categories.

```
import json

with open('Plant-Diseases-Detector-master/categories.json', 'r') as f:
    cat_to_name = json.load(f)
    classes = list(cat_to_name.values())

print (classes)
```

The above code is going to open the categories file sentence the different categories or the labels we want to train our model for. All the above code depends on which directory you are working on. if the name of the directory is same as mentioned in the code and you execute the same code in your systems you will get this result.

```
['Apple___Apple_scab', 'Apple___Black_rot', 'Apple___Cedar_apple_rust', 'Apple___healthy', 'Blueberry___healthy', 'Cherry_
(including_sour)___Powdery_mildew', 'Cherry_(including_sour)___healthy', 'Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spo
t', 'Corn_(maize)___Common_rust_', 'Corn_(maize)___Northern_Leaf_Blight', 'Corn_(maize)___healthy', 'Grape___Black_rot', 'G
rape___Esca_(Black_Measles)', 'Grape___Leaf_blight_(Isariopsis_Leaf_Spot)', 'Grape___healthy', 'Orange___Haunglongbing_(Cit
rus_greening)', 'Peach___Bacterial_spot', 'Peach___healthy', 'Pepper,_bell___Bacterial_spot', 'Pepper,_bell___healthy', 'Po
tato___Early_blight', 'Potato___Late_blight', 'Potato___healthy', 'Raspberry___healthy', 'Soybean___healthy', 'Squash___Pow
dery_mildew', 'Strawberry___Leaf_scorch', 'Strawberry___healthy', 'Tomato___Bacterial_spot', 'Tomato___Early_blight', 'Toma
to___Late_blight', 'Tomato___Leaf_Mold', 'Tomato___Septoria_leaf_spot', 'Tomato___Spider_mites Two-spotted_spider_mite', 'T
omato___Target_Spot', 'Tomato___Tomato_Yellow_Leaf_Curl_Virus', 'Tomato___Tomato_mosaic_virus', 'Tomato___healthy']
```

This is the list containing different classes of disease was Internet data set and against which we are going to change our model. One question which may arise as that why we have saved all these categories in Jason format, as json format is structured in a way of a dictionary data structure which is supported by python it makes it easy for us to work with this kind of key value pair data very easily.

```
print('Number of classes:',len(classes))
```

```
Number of classes: 38
```

On running the above snippet of code, I will be getting the number of classes which are present in the data set which I have extracted from the categories JSON file.

Now we have extracted the data from the data source and saved it into our own premise system or wherever you are working, and as we have also loaded the whole data set into our environment of Python so that we can access it and perform all kind of operations like model training data transformation and much more and now our next step these two transform this whole data set depending on different requirements so that we can make our model more efficient.

20.4 DATA PREPROCESSING

Data preprocessing is an essential step in image processing and computer vision. This process involves transforming raw image data into a format that can be utilized by machine learning algorithms. Data preprocessing for images typically includes tasks such as resizing, normalization, cropping, and data augmentation. Here's a detailed description of the common steps involved in image data preprocessing:

1. Image Resizing

The first step in image data preprocessing is to resize the images. This step is important because it helps to standardize the size of the images in the data set, which is essential for effective machine learning. Different algorithms have different requirements for image size, and resizing the images can be helpful in reducing the computational cost.

2. Normalization

Normalization is the process of scaling the pixel values in the images to a fixed range. This step is usually done to ensure that the pixel values lie between 0 and 1, which makes it easier for the machine learning algorithm to learn and identify patterns in the data.

3. Cropping

Cropping involves removing unwanted parts of the image that are not relevant to the task at hand. This step can help in reducing the computational cost and reduce the variability of the images in the data set.

4. Data Augmentation

Data augmentation is a technique used to artificially increase the size of the data set by creating new images from the original images. This step is particularly important when the data set is small or imbalanced. Common data augmentation techniques include flipping, rotation, zooming, and translation.

5. Quality Control

Finally, quality control is essential to ensure that the data set is clean and free from any errors or issues that may affect the performance of the machine learning algorithm. This step involves inspecting the images to ensure that they are labeled correctly and that there are no missing or corrupted images.

Overall, image data preprocessing is an essential step in preparing image data for machine learning. It is important to carefully consider the specific needs of the algorithm being used and adjust the preprocessing steps accordingly to ensure optimal performance.

As we may have different sizes of images in our data set or even in the case of user consider it this way if a user brings in an image which doesn't matter size or the number of input variables or the number of parameters a model can take it can cause a conflict so in the pipeline of our model processing we need to introduce a step where the image size is needed to be resized, the below snippet of code is a setting the standard size of image which model is going to take as input.

```
# Setup Image shape and batch size
IMAGE_SHAPE = (224, 224)

BATCH_SIZE = 64 #@param {type:"integer"}
```

As we can see the size of the image will be 224 * 224 which gives use 50,176 number of pixels, and this is going to be the size of the first layer of our neural network meaning this is several input variables are neural network is going to take as an input.

As now we have our two main datasets fun for training our model and another for providing the validation for the model, so we need to perform various data preprocessing tasks on both at first the below snippet of code is going to show you the different tasks that are performed on all the images of the training data set.

```
train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale = 1./255,
    rotation_range=40,
    horizontal_flip=True,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    fill_mode='nearest' )
```

Here we are using a library called Keras and there are various methods for performing preprocessing on images, and we are using ‘ImageDataGenerator()’ method for performing the different operations on the images for preprocessing.

Now let's dive into different operations which are being performed on the single image in the whole data set.

- Image rescaling is an important image pre-processing technique used to standardize image sizes before they are input into a machine learning algorithm for training or testing. This technique can be used to resize all images to a specific size, or to make them proportional to one another. There are several reasons why image rescaling is necessary in image pre-processing:
  - Standardization of Image Sizes: Different images in a dataset may have different sizes and resolutions, which can make it difficult for machine learning algorithms to effectively learn from the data. Rescaling images to a specific size ensures that all images in a dataset have the same dimensions, making it easier for the algorithm to learn from them.
  - Consistency: Large variations in image sizes can lead to inconsistent performance by the algorithm, making it difficult to identify patterns in the data. Rescaling images helps ensure consistency in the image sizes, thereby improving the accuracy of the algorithm.
  - Reducing Computational Cost: Large images take longer to process, thereby increasing the computational cost of running the algorithm. Rescaling images to a smaller size can reduce this computational cost, enabling the algorithm to run faster and more efficiently.
- Some common techniques used in rescaling images include:
  - Scaling by a fixed ratio: Images can be scaled up or down by a fixed percentage to achieve a standard size.
  - Scaling to a specific size: All images in a dataset can be rescaled to a specific size (such as 256x256 or 512x512 pixels) to ensure consistency.
  - Scaling to a proportional size: Images can be rescaled so that images of the same proportion have the same size. This technique maintains the aspect ratio of the image while standardizing the size.

  Rescaling images is an important image pre-processing task that helps standardize image sizes, ensures consistency, and reduces computational cost. It helps machine learning

algorithms to learn better features from the data and improve the overall accuracy of the models trained on the image datasets.

- Rotation is a common image pre-processing technique used to augment and expand image datasets for training machine learning models. By rotating images within a certain range, we can create new images with different angles that can help the model generalize better and improve its performance. In image pre-processing, the rotation range refers to the degree range that the images will be rotated. This range can be specified by a minimum and maximum degree or as a single value, depending on the application. Some common rotation ranges include:
  - Fixed Rotation: This involves rotating all images in a dataset by a fixed angle (e.g., 90 degrees, 180 degrees, etc.). This technique is useful when the images have a specific orientation, such as landscape or portrait.
  - Random Rotation: This involves rotating each image in the dataset by a randomly generated angle within a specified range, typically between 0 to 360 degrees. This technique creates new images with different angles, expanding the dataset and improving the performance of machine learning models.
  - Limited Rotation: This technique involves rotating images by a specified range, such as -15 to +15 degrees, to create new images that are similar to the original but with slight variations. This technique can improve the model's ability to detect objects at different angles.

Rotation is an important image pre-processing technique used to augment and expand image datasets for training machine learning models. By rotating images within a certain range, we can create new images with different angles that can help the model generalize better and improve its performance.

- Horizontal flip is a common image pre-processing technique that involves generating a mirrored image of an original image along its vertical axis. It is a type of data augmentation that can be used to generate new images for a dataset, which can help to improve the accuracy and robustness of machine learning models. In horizontal flip, the image is flipped along the horizontal axis, such that the left side of the original image

becomes the right side of the flipped image, and the right side of the original image becomes the left side of the flipped image. Some of the benefits of using horizontal flip as an image pre-processing task include:

- Creating New Images: By flipping images horizontally, it generates new images that are similar to the original image, but with a slightly different appearance. This technique can artificially increase the size of the dataset and help to improve the performance of the machine learning model.
- Enhancing the Robustness of the Model: By including flipped images in the dataset, the machine learning algorithm can be trained on different orientations of the object, and improve the model's ability to recognize the object from different perspectives. This is particularly helpful when images have objects with varying orientations such as text, traffic signs, and so on.
- Reducing Overfitting: Overfitting occurs when a machine learning model is trained too well on the training data and performs poorly on test data. Using techniques like horizontal flip can help the model generalize better by exposing it to more diverse images during training, thereby reducing overfitting.

Horizontal flip is typically a simple yet effective image pre-processing technique that is easy to implement for most machine learning applications. It may be used alone or in combination with other pre-processing techniques like rotation, normalization, and cropping.

❖ Width shift is a common image pre-processing technique used in data augmentation to generate new images from an existing dataset by shifting the image horizontally along the x-axis. The width shift range refers to the range within which the images can be shifted or translated horizontally. In width shift, the image can be shifted left or right by a specified distance or a randomly generated distance within the specified range. This technique simulates images of objects that are partially captured or objects that are not in the center of the image. Some of the benefits of using width shift range as an image pre-processing task include:

- Creating New Images: By shifting images horizontally, we can create new and diverse images that help to increase the size of the dataset and enable the model to learn more

robust patterns. Such images can help the machine learning algorithm better recognize objects in different positions.

- Improving Generalization: Training a machine learning model on a dataset with data translated within a range can result in the model learning features that are invariant to object position. Therefore, it can more efficiently detect and differentiate the object even if it's partially visible.
- Reducing Overfitting: Like other image pre-processing techniques, using the width shift range in data augmentation helps to reduce overfitting by artificially increasing the diversity of the dataset and preventing unnecessary memorization of the training data.

The width shift range is specified as a fraction of the image dimensions, and the actual translation performed is chosen randomly within this range. Typically, a value between 0.2 and 0.3 of the image dimensions is a good starting point, but the optimal value for this parameter may vary depending on the specific use case.

Overall, width shift range is a useful technique for image pre-processing and data augmentation as it can help to increase dataset diversity, prevent overfitting, and improve the accuracy of machine learning models.

❖ Height shift is a common image pre-processing technique used in data augmentation to generate new images from an existing dataset by shifting the image vertically along the y-axis. The height shift range refers to the range within which the images can be shifted or translated vertically.In height shift, the image can be shifted up or down by a specified distance or a randomly generated distance within the specified range. This technique simulates images of objects that are captured at different heights or heights that are distorted by perspective. Some of the benefits of using height shift range as an image pre-processing task include:

- Creating New Images: By shifting images vertically, we can create new and diverse images that help to increase the size of the dataset and enable the model to learn more robust patterns. Such images can help the machine learning algorithm better recognize objects in different positions.

- Improving Generalization: Training a machine learning model on a dataset with data translated within a range can result in the model learning features that are invariant to object height. Therefore, the model can more efficiently detect and differentiate the object even if it's captured from different heights.
- Reducing Overfitting: As described previously, using the height shift range in data augmentation helps to reduce overfitting by artificially increasing the diversity of the dataset and preventing the model from memorizing the training data.

The height shift range is specified as a fraction of the image dimensions, and the actual translation performed is chosen randomly within this range. Typically, a value between 0.2 and 0.3 of the image dimension is a good starting point, but the optimal value for this parameter may vary depending on the specific use case.

Overall, height shift range is a useful technique for image pre-processing and data augmentation as it can help to increase dataset diversity, prevent overfitting, and improve the robustness and accuracy of machine learning models.

❖ Shear range is a common image pre-processing technique used in data augmentation to generate new images from an existing dataset by shearing the image. Shear refers to the deformation or distortion of the shape of an object along an axis that is parallel to its face. In image processing, shearing refers to a transformation that slants the image along a specified axis.In shear range, the image is sheared at a random angle within a specified range along horizontal or vertical axes. This technique simulates the perspective distortion that is common in images captured from different angles.Some of the benefits of using shear range as an image pre-processing task include:

- Creating New Images: By shearing images along horizontal or vertical axes, we can create new and diverse images that help to increase the size of the dataset and enable the model to learn more robust patterns. Such images can help the machine learning algorithm better recognize objects from different perspectives.
- Improving Generalization: Training a machine learning model on a dataset with data sheared within a range can result in the model learning features that are invariant to perspective distortion.

- Reducing Overfitting: As with other image pre-processing techniques, using the shear range in data augmentation helps to reduce overfitting by artificially increasing the diversity of the dataset.

The shear range is specified as an angle range within which the image will be sheared. Typically, a range between 0 and 0.3 radians is a good starting point, but the optimal value for this parameter may vary depending on the specific use case.

Overall, shear range is a useful technique for image pre-processing and data augmentation as it can help to increase dataset diversity, prevent overfitting, and improve the robustness and accuracy of machine learning models.

❖ Zoom range is a common image pre-processing technique used in data augmentation to generate new images from an existing dataset by zooming the image. This technique can be used to simulate the images of objects captured at different distances and can help to improve the performance and generalization of machine learning models.In zoom range, the image is zoomed in or out by a specified percentage or a randomly generated percentage within a specified range. This technique can help to artificially increase the size of the dataset and enable the model to better recognize objects at various distances.Some of the benefits of using zoom range as an image pre-processing task include:

- Creating New Images: By zooming the images, we can create new and diverse images that help to increase the size of the dataset, augment the training data, and improve the performance of the machine learning models.
- Improving Generalization: Training a machine learning model on a dataset with data zoomed within a range can result in the model learning features that are invariant to object size or distance, thereby improving the model's ability to recognize objects at various zoom levels.
- Reducing Overfitting: Using the zoom range in data augmentation can help to reduce overfitting by artificially increasing the diversity of the dataset and preventing unnecessary memorization of the training data.

The zoom range is specified as a percentage range within which the image will be zoomed. Typically, a range between 0 and 0.3 is a good starting point, but the optimal value for this parameter may vary depending on the specific use case.

Overall, zoom range is a useful technique for image pre-processing and data augmentation as it can help to increase dataset diversity, prevent overfitting, and improve the accuracy and robustness of machine learning models.

- ❖ Fill mode is a common image pre-processing technique used to fill in the empty spaces that might appear in an image after pre-processing tasks like rotation or resizing. When an image is rotated or resized, the empty pixels that are created can cause noise, outliers, or unrealistic features that might hinder the learning capability of machine learning algorithms. Fill mode is used to fill these empty pixels in different ways, depending on the specific use case. There are several fill modes that are commonly used in image pre-processing, but the most common ones include crop, reflect, and constant methods.
    - Crop Method: This method involves simply cropping the image to remove the empty pixels. This technique is useful when the original image contains sufficient data for the machine learning task and the empty pixels can be ignored.
    - Reflect Method: This method involves mirroring the available pixels to fill in empty pixels. This technique is useful when the image borders contain important information, and the machine learning algorithm should be able to learn from it.
    - Constant Method: This method involves filling the empty pixels with a predefined constant value. This technique is useful when preserving the shape of the image is very important, and the best practice is to keep the image size and shape fixed across the dataset.

Using the fill mode in the image pre-processing task is essential to maintain the integrity and quality of the images while ensuring that the data is maximally informative for the machine learning algorithms. It helps prevent overfitting by augmenting the data without unnecessary information and helps machine learning algorithms learn important features and patterns from the images.

After finding the different transformation that we have to perform on the image, they are not exactly transformation more like augmentation which are being performed in image for

making the disease detection more difficult for our machine learning model so that it can adapt to different scenarios for example we may have some leaves which are being affected by some light exposure or the direction of an uh image out the orientation of the image so these kinds of previous processing tasks are needed to be introduced in the data pipeline or else I won't be able to perform in the best way possible, now it is time to generate the transformed data, below snippet of code is going to perform that task.

```
train_generator = train_datagen.flow_from_directory(
    train_dir,
    subset="training",
    shuffle=True,
    seed=42,
    color_mode="rgb",
    class_mode="categorical",
    target_size=IMAGE_SHAPE,
    batch_size=BATCH_SIZE)
```

The same operations are performed for validation it asset with few tunings in the settings as uh we don't really want to make our images more complex as validation tests are performed just to see how well our model has been trained so the transformation not that necessary when it comes to validating our model unlike in the case of training our model which requires covering as many aspects or scenarios as we can cover.

```
Found 10861 images belonging to 38 classes.
Found 43444 images belonging to 38 classes.
```

The above result shows us the number of images present in validation class and then training class of images respectively.

20.5 MODEL BUILDING

Model building is a critical step in the process of plant disease classification using machine learning. The goal of model building is to design a machine learning model that can accurately classify plant leaves images into diseased or healthy categories. Here are the essential steps involved in building a plant disease classification model:

- Data Collection: Collect a large dataset of images that are labeled as either healthy or diseased. The dataset should be representative of the plant species and diseases that the model is designed to identify.
- Data Pre-processing: Pre-process the data by resizing, normalization, cropping, and augmenting it to improve the quality and diversity of the dataset.
- Training-Validation-Testing Split: Split the pre-processed data into training, validation, and testing sets. The training set is used to build the model, the validation set is used for hyperparameter tuning, and the testing set is used for model evaluation.
- Model Selection: Select the appropriate machine learning algorithm for the classification task. Common algorithms used for plant disease classification include Convolutional Neural Networks (CNNs), Support Vector Machines (SVMs), Random Forests, and K-Nearest Neighbors (KNNs).
- Model Architecture: Design the architecture of the machine learning model based on the selected algorithm. This architecture should take into account the specifications of the dataset and the performance of the learning algorithm.
- Hyperparameter Tuning: Optimize the model by selecting the optimal hyperparameters to improve the model's performance on the validation set.
- Model Training: Train the model using the training set, and validate it using the validation set. Train the model using an iterative process such as Batch Gradient Descent or Mini-batch Gradient Descent, to improve model performance.
- Model Evaluation: Evaluate the trained model's performance on the testing set using metrics such as accuracy, precision, recall, and F1 score, and confusion matrix.
- Model Deployment: Deploy the model to an application or service that can leverage the model's prediction capabilities. The deployment can be on-premises or on the cloud dependently.

In conclusion, model building is a fundamental step in plant disease classification using machine learning. It involves selecting the appropriate algorithm, designing the model architecture, optimizing the hyperparameters, training the model and evaluating the performance on the testing set, and deploying the trained model. This process requires a careful selection of the specific tasks outlined above to achieve optimal performance.

```
# Build the model
model = tf.keras.Sequential([
  hub.KerasLayer("https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature_vector/4",
                 output_shape=[1280],
                 trainable=False),
  tf.keras.layers.Dropout(0.4),
  tf.keras.layers.Dense(512, activation='relu'),
  tf.keras.layers.Dropout(rate=0.2),
  tf.keras.layers.Dense(train_generator.num_classes, activation='softmax')
])
```

The error model consists of two parts first part is the pre trained model provided by trans of row called mobile net V2 whose architecture is given below.

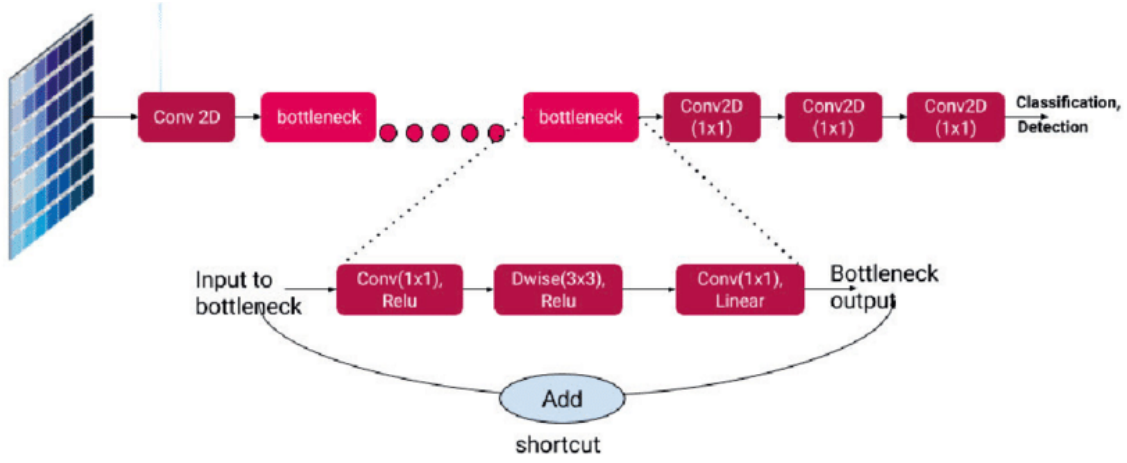

Figure 45 Architecture of MobileNet V2

And then what we have done is we have added few more layers to provide more filtration on the output, and whole model is a sequential model where different layers are just stacked over one another.

An overview of the whole architecture of neural network is provided below along with description of every layer that I have used in this.

- Keras.Sequential is a type of model architecture in the Keras deep learning library for Python. It is a simple model that consists of a linear stack of layers that can be used for a wide range of deep learning tasks such as image classification, text analysis, and regression.
- The Keras.Sequential model is called sequential because each layer is added one at a time in a linear fashion, forming a "stack" of layers. The output of one layer acts as the input to the next layer in the stack. Sequential models are generally easier to build and use, making them popular for beginners and experts.

Characteristics Of the Keras.Sequential Models:

- Easy to Implement: Keras.Sequential models are simple and easy to implement. It is a good choice for quick experiments and creating machine learning models that follow a simple architecture.
- Flexible: The Keras.Sequential model is flexible because it allows the creation of any combination of layers and activation functions and supports various types of input.
- Scalable: Sequential models can be scaled up by adding more layers or additional nodes in each layer, making it ideal for larger, more complex prediction tasks.
- Compatible with Different Layers: Keras.Sequential models are compatible with various layers, including convolutional, pooling, recurrent, dense, and more. These layers can be added to the model sequentially to build a robust machine learning model.
- Optimized for GPU: Keras.Sequential models are highly optimized for GPU processing, making it faster and more efficient to train deep learning models.

MobileNetV2

It is a convolutional neural network (CNN) architecture, often used for image classification. It is designed to be lightweight and efficient, making it an excellent choice for mobile devices and other applications with limited computational resources. MobileNetV2 introduces several new ideas to the original MobileNet architecture, including inverted residuals, linear bottlenecks, and shortcut connections, to improve the accuracy and performance of the model. Here are the architectural details of MobileNetv2:

Input Layer: The input layer of MobileNetV2 accepts an input image of shape (224, 224, 3), where 224x224 are the image dimensions, and 3 represents the Red, Green, and Blue (RGB) color channels of the image.

- Convolutional Layers: The backbone of MobileNetV2 consists of a series of convolutional layers. These layers use depthwise separable convolutions to reduce the number of parameters. Depthwise separable convolutions perform convolution operations separately for the input channels, followed by pointwise convolutions (1x1 convolutions) to combine the output channels. This architecture results in a significant reduction in model size and computational cost, ideally suited to mobile devices.
- Inverted Residuals: MobileNetV2 introduces a new module called "inverted residuals," which includes a depthwise convolution layer, a 1x1 pointwise convolution layer to perform feature transformation, and a linear bottleneck layer to increase feature depth.
- Shortcut Connections: MobileNetV2 incorporates shortcut connections to connect certain layers directly with a few layers ahead. This technique is used to improve the gradient flow, enabling better information propagation through the network.
- Linear Bottlenecks: Linear bottlenecks are used to improve the accuracy of MobileNetV2. They are intermediate layers that use linear activations instead of non-linear activation functions such as ReLU. Linear bottlenecks reduce the dimension of the feature space, allowing the model to use the limited number of parameters efficiently.
- Classification Layers: The final part of the architecture consists of a Global Average Pooling layer that takes the output of the convolutional layers and averages it over the spatial dimensions, followed by a fully connected layer and a softmax function to produce the output probability distribution.

tf.keras.layers.Dropout

It is a regularization technique that is used to prevent overfitting in neural networks. Dropout is a type of regularization method that randomly turns off a fraction of the neurons in a layer during training. It works by randomly dropping out some of the units in a layer and setting their output to zero during training.

The Dropout layer in Keras is a regularization layer that can be added to any layer in a neural network. Typically, it is placed after the activation layer in a multi-layer neural network.

Here's how Dropout regularization works in Keras:

- During training, a fraction of the neurons in the layer are randomly selected and deactivated by setting their output to zero.
- The remaining active neurons are then trained on the data as normal, as part of a forward and backward pass.
- During inference, i.e. when the model is being used for prediction rather than training, the Dropout layer is turned off so that all neurons are active, and the complete model is used.

The proportion of neurons to be dropped out is a hyperparameter that can be tuned. The purpose of dropout is to address the overfitting problem by reducing the co-adaptation of neurons during training. By randomly dropping out some of the neurons, it forces the other neurons to learn more robust and generalizable features.

In summary, the Dropout layer in Keras is a regularization technique that randomly removes a fraction of the neurons in the layer, forcing the rest of the neurons to learn more robust features during training. This technique helps to prevent overfitting, improve the generalization of the model, and ultimately leads to better performance on unseen data.

tf.keras.layers.Dense

It is a neural network layer in the Keras API for TensorFlow, used to implement fully connected layers in a deep learning model. A dense layer consists of multiple nodes that are fully connected to the output of the previous layer. Each node in the layer receives an input from the entire output of the previous layer and computes a dot product with a set of learnable weights. The resulting output is passed through an activation function, which introduces non-linearity into the model. Here's a brief overview of the important aspects of a Dense layer in TensorFlow:

- Input Shape: The Dense layer requires information about the shape of the input data. This information is used to construct the weight matrix that connects the layer to the previous layer.
- Output Shape: The output shape of the Dense layer is determined by the number of units or nodes in the layer.
- Activation Function: The Dense layer requires an activation function to introduce non-linearity into the model. Common activation functions used with Dense layers include sigmoid, ReLU, and tanh.
- Weights: The Dense layer learns the appropriate weights to connect each input feature to each output feature. During training, the goal is to adjust the weights to minimize the loss.
- Bias: The Dense layer includes a bias term that is added to the output of each node in the layer. The bias term acts as an offset to the linear combination of inputs and weights.

The Dense layer is widely used in deep learning models, and it can be used for both regression and classification tasks. It is flexible and can be modified to allow for regularization techniques such as dropout, l1/l2 regularization, and batch normalization. The number of Dense layers in a neural network model and the number of nodes in each Dense layer are hyperparameters that need to be tuned for optimal performance.

20.7 TRAINING/ VALIDATION

Model training and validation are critical steps in building machine learning models. The training step involves tuning the model, optimizing its weights, and seeking the best performing model on the training dataset, as well as evaluating the model's generalization performance on the new unseen dataset. These steps help to fine-tune the model, select the optimal hyperparameters, avoid overfitting, and achieve better model performance.

But before just driving into training and validation we have to define few parameters that the model or more specifically a neural network takes in for reaching or achieving the best accuracy we want one such parameter is learning rate, number of approaches, batch size, and optimizer. A detailed description is provided below.

20.7.1 Learning rate

It is a key hyperparameter of many machine learning algorithms that determines the step size at which the gradient or error is used to update the weights (or coefficients) of the model. Simply put, the learning rate is a tuning parameter used to balance the convergence speed of the optimization algorithm and the risk of overshooting the optimal solution.

A learning rate that is too high may result in the optimization algorithm diverging (i.e., not converging to a useful solution), whereas a learning rate that is too low may result in the optimization algorithm converging too slowly, delaying the optimization process.

For example, in gradient descent optimization, the update rule of weights w is $w = W - \eta \nabla L(w)$, where η is the learning rate, L is the loss function, and ∇L is the gradient of the loss function with respect to the weights w. The learning rate η determines how much the gradient influences the weight update and is thus a critical parameter that should be tuned to optimize the performance of the model.

The optimal learning rate depends on factors such as the complexity of the model, the size and quality of the dataset, the gradient descent optimization method used, and more. One of the common methods for tuning the learning rate is by using a learning rate schedule protocol such as decreasing the learning rate over time or per iteration to further fine-tune the convergence speed of the model.

Finding an appropriate learning rate typically requires some experimentation and iteration. In practice, trial and error are used to iterate through different learning rates to determine the most appropriate one that optimizes the model for the given problem. Consequently, the learning rate schedule is changed to adjust the rate during model training, depending on the signs of overfitting or underfitting of the model.

20.7.2 Batch size

It is another important hyperparameter in machine learning that refers to the number of training examples used in a single iteration during the optimization of a model. The training data is split into batches of a fixed size, and each batch is used to update the model's

parameters during training. The batch size is typically set to a power of 2, such as 32, 64, or 128, to enable efficient computation on modern hardware.

Batch size has a significant impact on the training process and model performance, and different batch sizes can influence the speed, quality, and generalization capability of the model. Here are some factors to consider when selecting the appropriate batch size for your model:

- Computational Resources: Batch size is closely related to the amount of memory required to train the model. Larger batch sizes require more memory but usually offer faster optimization.
- Optimization Speed: Smaller batch sizes offer higher optimization speed at the cost of accuracy, whereas larger batch sizes achieve higher accuracy levels but reduce optimization speed considerably.

20.7.3 Optimizer

Optimizer in machine learning refers to an algorithm or method that is used to optimize the parameters of a machine learning model during the training process. The objective of optimization is to minimize the loss function, which reflects the error between the predicted output of the model and the actual output.

Optimizers work by adjusting the weights and biases of a model based on the error or loss function, which measures how accurately the model is performing. The most popular optimization algorithms used in machine learning include stochastic gradient descent (SGD), Adagrad, RMSprop and Adam.

SGD is a simple and widely used optimization method that updates the weights and biases of a model at each step based on the gradient of the loss function with respect to the parameters. Other optimization algorithms, such as Adagrad, RMSprop and Adam, are more sophisticated and leverage techniques such as adaptive learning rates and momentum to improve optimization performance.

20.7.4 Loss

In machine learning, loss refers to the error or cost of a model's predictions compared to the true values. In particular, it's a measure of how well a model is able to map inputs to outputs.

In neural network training, the loss function measures the difference between the predicted output of the model and the true target output. The goal of training is to minimize the loss function, so that the predictions of the model are as close as possible to the true target output.

During the training of a neural network, the model's parameters, such as weights and biases, are updated based on the gradients of the loss function. This process is repeated over multiple iterations until the loss function is minimized and the model converges to the optimal set of parameters.

Common loss functions used in neural network training include mean squared error (MSE), binary cross-entropy, and categorical cross-entropy. The choice of loss function depends on the problem being solved and the type of output being predicted.

20.7.5 Metrics

In neural network training, metrics are used to evaluate the performance of a model during training and testing. Metrics are a set of quantitative measurements that help us to determine how well the model is learning and generalizing its predictions.

Some common metrics used in neural network training include accuracy, precision, recall, F1 score, and area under the receiver operating characteristic curve (ROC-AUC). The choice of metric depends on the problem being solved and the type of output being predicted.

Accuracy is a simple and widely used metric that measures the percentage of correctly predicted samples out of all samples. Precision and recall are more specific to binary classification problems and measure the fraction of true positives over all predicted positives and the fraction of true positives over all actual positives, respectively.

The F1 score is a harmonic mean of precision and recall that balances both metrics. ROC-AUC is commonly used in binary classification problems and measures the trade-off between true positive rate and false positive rate at different probability thresholds.

By monitoring metrics during training, we can use this to modify the model to improve the results. Additionally, metrics can be used to compare different models and select the best one for a specific problem.

20.7.6 Epoch

In machine learning, an epoch refers to a complete pass through the entire training dataset during model training. One epoch is defined as one forward pass and one backward pass of each training example, and the goal of training is to reduce the loss function over multiple epochs.

In practical terms, an epoch is a hyperparameter that is chosen by the model developer and can have a significant impact on the final performance of the model. Choosing an appropriate number of epochs depends on factors such as the size and complexity of the dataset, the complexity and size of the model, and the number of computational resources available.

During training, the model updates its parameters based on the gradients of the loss function with respect to the parameters for each epoch. After each epoch, the model's performance can be evaluated on a separate validation set or through cross-validation to monitor its performance.

Too few epochs can result in underfitting, where the model does not learn enough from the data, while too many epochs can result in overfitting, where the model learns to memorize the training data and therefore performs poorly on new, unseen data. Thus, the number of epochs is typically chosen using methods such as early stopping, where the training is halted once the model's performance stops improving on a validation set.

```
# Specify Loss Function and Optimizer
LEARNING_RATE = 0.001 #@param {type:"number"}

model.compile(
   optimizer=tf.keras.optimizers.Adam(lr=LEARNING_RATE),
   loss='categorical_crossentropy',
   metrics=['accuracy'])
```

The above snippet of code is combining the different parameters that we have defined like learning rate the optimizer the kind of loss we are going to use for measuring the loss and

then metrics which is going to be used for telemetry purposes which is referred for seeing how well our model is predicting or working.

Our model is trained using backpropagation.

Backpropagation

Backpropagation is an algorithm used in machine learning and specifically in neural networks, that calculates the gradient of the loss function with respect to the weights in the neural network. It works by computing the gradients of the loss function with respect to the activations of each neuron in the last hidden layer, and then propagating these gradients backwards through the network to compute the gradients of the loss function with respect to the weights.

The core concept behind backpropagation is the chain rule of calculus, which allows us to calculate the derivative of a nested function by breaking it down into smaller derivatives and multiplying them together. In the case of neural networks, the chain rule is used to calculate the partial derivative of the loss function with respect to each weight in the network.

The backpropagation algorithm is traditionally used in conjunction with stochastic gradient descent (SGD) and other optimization algorithms to train neural networks. During training, the loss function is evaluated using a training data set, and the network parameters are updated iteratively by computing the gradients using backpropagation and then adjusting the parameters in the direction of the negative gradients. The process continues until the loss function converges to a minimum.

Backpropagation is a fundamental concept in modern machine learning and is used in a variety of deep learning architectures and applications. It enables neural networks and other machine learning models to learn from complex and large datasets, making it a widely used technique for training complex models.

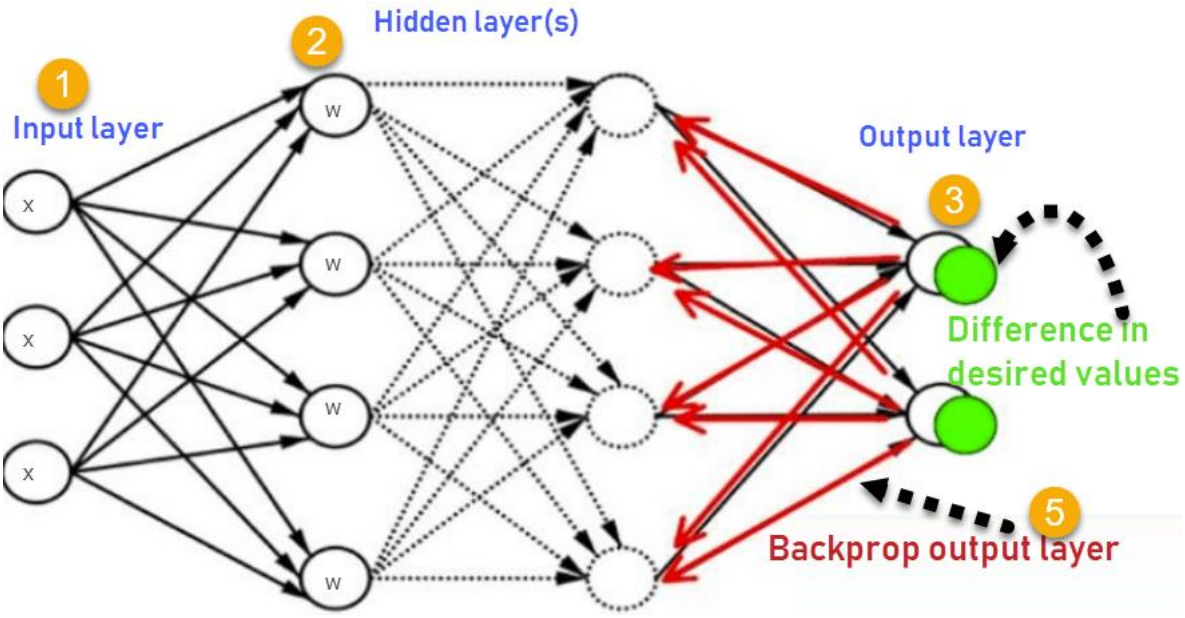

Figure 46 Backpropagation Workflow

```
# Train Model

EPOCHS=10 #@param {type:"integer"}

history = model.fit_generator(
        train_generator,
        steps_per_epoch=train_generator.samples//train_generator.batch_size,
        epochs=EPOCHS,
        validation_data=validation_generator,
        validation_steps=validation_generator.samples//validation_generator.batch_size)
```

The above stripped of code is starting the training of the model and performing the training of the model in 10 epochs,

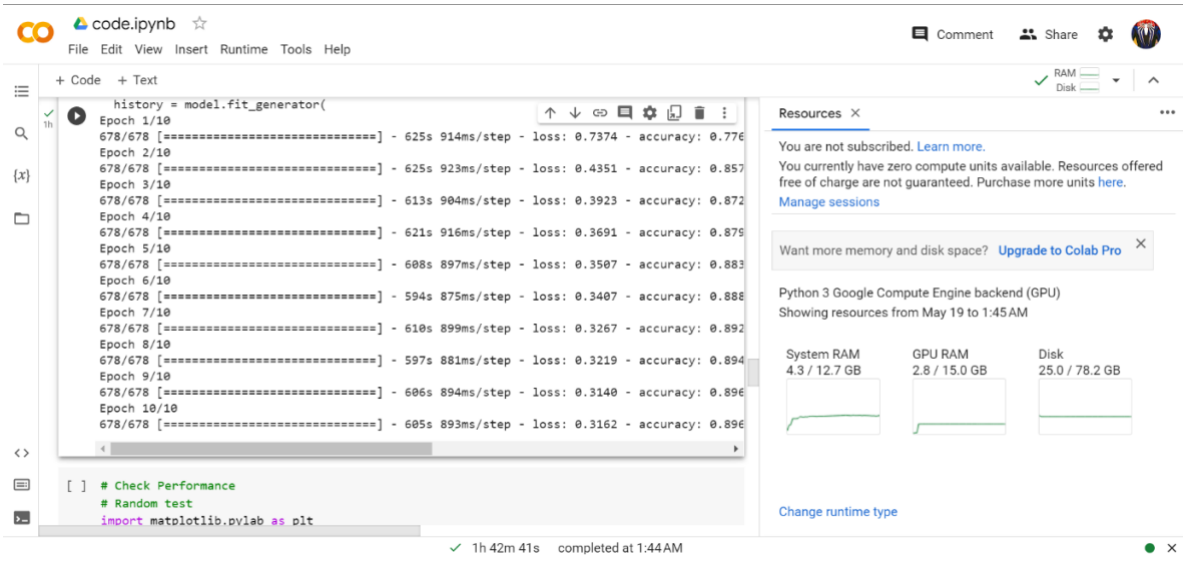

Figure 47 Model Training Completion

The image shows us the completion of the training of our model and the amount of resources used while the process, as you can see the resources like RAM GPU and the disk storage which was used while the training of the model and the amount of time is also mentioned at the very bottom of the image as you can see it took around 1 hour 42 minutes for completing the training of our model this was possible because of the inclusion of GPU to boost the performance or the speed of training as if I was to use the CPU as my hardware it would have taken a lot more time on the data set of this much size and specifically the images require powerful hardware as they are array of data rather than just a single array they contain dimensions and as size of image increases the amount of data in a particular image also increases so we need a much better way a more powerful way to train our model so GPU is answer for doing that. Now it's time for performing validation test runner models so the below Porter is going to lowered our validation data set and import it or pass it to our trade model and see how well our model is done on this data set.

```
# Check Performance
# Random test
import matplotlib.pylab as plt
import numpy as np

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(EPOCHS)
```

After doing this we’re going to also map a graph for make the telemetry more readable and understandable for other people.

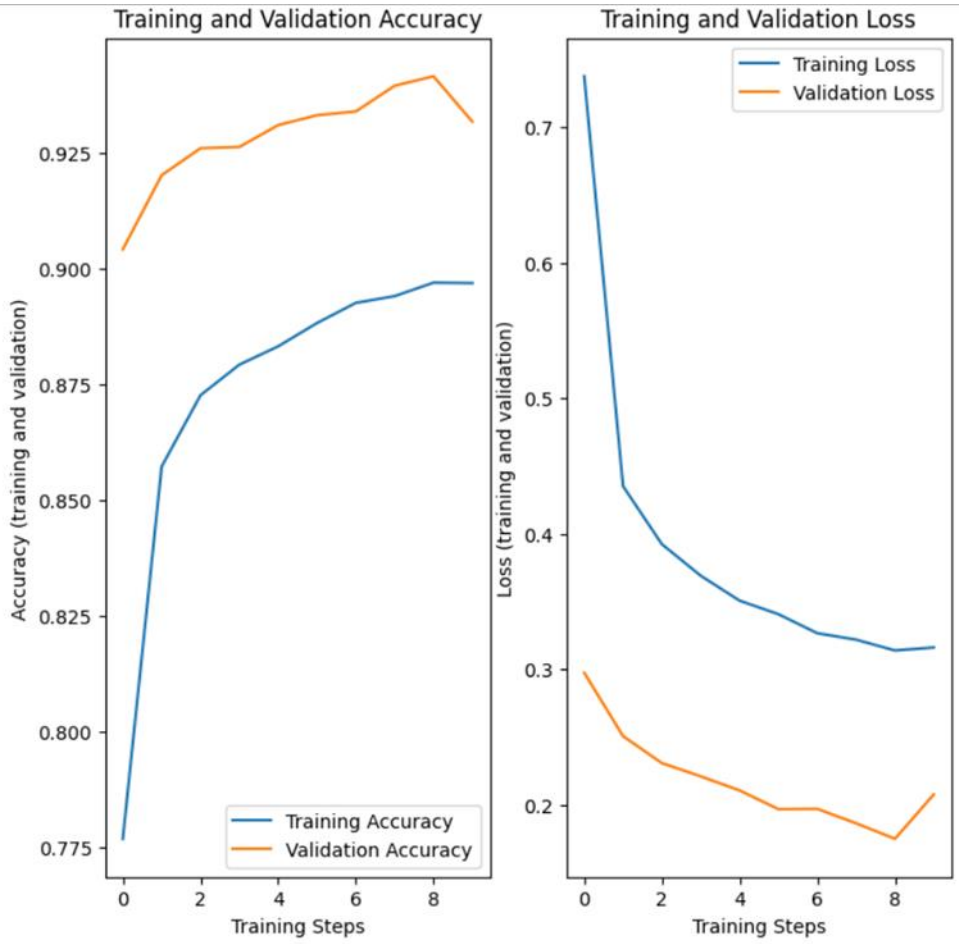

Figure 48 Training and Validation Accuracy and Loss

This concludes our whole life cycle of collecting data from a data source and then performing different preprocessing tasks on that data then selecting our model which is more helpful to us and then training that model on that data and then checking how well our model was trained and highlighting level of accuracy.

THE CODE FOR ALL THE PROCESSES, IS GIVEN BELOW:

```
#!/usr/bin/env python

# coding: utf-8

# In[1]:

from __future__ import absolute_import, division, print_function, unicode_literals

import tensorflow as tf
```

```
#tf.logging.set_verbosity(tf.logging.ERROR)
#tf.enable_eager_execution()
import tensorflow_hub as hub
import os
from tensorflow.keras.layers import Dense, Flatten, Conv2D
from tensorflow.keras import Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import layers
#from keras import optimizers
# In[2]:
# verify TensorFlow version
print("Version: ", tf.__version__)
print("Eager mode: ", tf.executing_eagerly())
print("Hub version: ", hub.__version__)
print("GPU is", "available" if tf.test.is_gpu_available() else "NOT AVAILABLE")
# In[3]:
# Loading Data
zip_file =
tf.keras.utils.get_file(origin='https://storage.googleapis.com/plantdata/PlantVillage.zip',
                        fname='PlantVillage.zip', extract=True)
# In[4]:
# Prepare training and validation dataset
data_dir = os.path.join(os.path.dirname(zip_file), 'PlantVillage')
train_dir = os.path.join(data_dir, 'train')
validation_dir = os.path.join(data_dir, 'validation')
# In[5]:
import time
import os
```

```
from os.path import exists
def count(dir, counter=0):
    "returns number of files in dir and subdirs"
    for pack in os.walk(dir):
        for f in pack[2]:
            counter += 1
    return dir + " : " + str(counter) + "files"
# In[6]:
print('total images for training :', count(train_dir))
print('total images for validation :', count(validation_dir))
# In[7]:
# Label mapping
get_ipython().getoutput('wget https://github.com/obeshor/Plant-Diseases-Detector/archive/master.zip')
get_ipython().system('unzip master.zip;')
# In[8]:
import json
with open('Plant-Diseases-Detector-master/categories.json', 'r') as f:
    cat_to_name = json.load(f)
    classes = list(cat_to_name.values())
print (classes)
# In[9]:
print('Number of classes:',len(classes))
# In[10]:
# Setup Image shape and batch size
IMAGE_SHAPE = (224, 224)
BATCH_SIZE = 64 #@param {type:"integer"}
# In[11]:
# Data Preprocessing
```

```
# Inputs are suitably resized for the selected module. Dataset augmentation (i.e., random distortions of an image each time it is read) improves training, esp. when fine-tuning.
validation_datagen = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255)
validation_generator = validation_datagen.flow_from_directory(
    validation_dir,
    shuffle=False,
    seed=42,
    color_mode="rgb",
    class_mode="categorical",
    target_size=IMAGE_SHAPE,
    batch_size=BATCH_SIZE)
do_data_augmentation = True #@param {type:"boolean"}
if do_data_augmentation:
  train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
      rescale = 1./255,
      rotation_range=40,
      horizontal_flip=True,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=0.2,
      zoom_range=0.2,
      fill_mode='nearest' )
else:
  train_datagen = validation_datagen
train_generator = train_datagen.flow_from_directory(
    train_dir,
    subset="training",
    shuffle=True,
    seed=42,
```

```
    color_mode="rgb",
    class_mode="categorical",
    target_size=IMAGE_SHAPE,
    batch_size=BATCH_SIZE)
# In[13]:
# Build the model
model = tf.keras.Sequential([
  hub.KerasLayer("https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature_vector/4",
                 output_shape=[1280],
                 trainable=False),
  tf.keras.layers.Dropout(0.4),
  tf.keras.layers.Dense(512, activation='relu'),
  tf.keras.layers.Dropout(rate=0.2),
  tf.keras.layers.Dense(train_generator.num_classes, activation='softmax')
])
# In[14]:
# Specify Loss Function and Optimizer
LEARNING_RATE = 0.001 #@param {type:"number"}
model.compile(
   optimizer=tf.keras.optimizers.Adam(lr=LEARNING_RATE),
   loss='categorical_crossentropy',
   metrics=['accuracy'])
# In[16]:
# Train Model
EPOCHS=10 #@param {type:"integer"}
history = model.fit_generator(
        train_generator,
        steps_per_epoch=train_generator.samples//train_generator.batch_size,
        epochs=EPOCHS,
```

```
        validation_data=validation_generator,
        validation_steps=validation_generator.samples//validation_generator.batch_size)
# In[17]:
# Check Performance
# Random test
import matplotlib.pylab as plt
import numpy as np
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs_range = range(EPOCHS)
plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
plt.ylabel("Accuracy (training and validation)")
plt.xlabel("Training Steps")
plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.ylabel("Loss (training and validation)")
plt.xlabel("Training Steps")
plt.show()
```

CHAPTER 21 ASSISTANCE FOR TREATMENT

The ability to be able to identify the kind of disease uh when splint or a crop may be suffering from is one functionality but what we also need along with this ability is that the user or the farmer who is trying to identify the disease can also receive uh the advice or the suggestion or the exact treatment for that disease on the planet so that he can prevent his growth from deteriorating and receiving the best yield possible so for that what we need is yeah create texture like a Chad word where the person can interact with but chat boards also the only limited functionality and we have to again and again upgrade their information and they don't even have human like ability to extract the information or the knowledge which is required by the user So what we have done here is that we have used a large language model and integrated it in our application which is going to be explained in next section or next chapter to precise. That large language model is going to act as another part of this whole data pair pipeline where it is going to receive the classified disease by the computer vision model and after that it is going to provide the cures or the treatment the user can provide or treat his or her crop with and prevent the deteriorating of his or her crop.

Disability of a chatting with the human like interface rather than waiting for someone to contact customer service yeah you two are partially specialized in the and then provide you with the treatment or suggestions how you can attend your crops but considering this way every person in their provide phone can access a large language model powered assistant which can act like a human and then provide you all the kinds of answers regarding your all the kinds of queries so that you can just very fastly and without waiting for someone else to call you know you can just receive the required treatment advice for your plan.

Overview of the workflow how this idea of large language model powered assistant is embedded into the application is given below:

- User inputs an image of the affected area into the application.
- The computer vision model analyzes the image and identifies the disease or condition.
- The OpenAI language model (LLM) assistant is triggered based on the identified disease or condition.

- ❖ The LLM provides relevant information on the condition, such as its symptoms, causes, and potential treatments.
- ❖ Once the LLM has provided treatment information, the user can review the information and follow up with their healthcare provider for personalized treatment recommendations.

The Langchain framework provides a way to integrate machine learning models with natural language processing technologies, allowing the user to converse with the AI-powered assistant in natural language. This enables the assistant to provide more personalized and specific guidance to the user based on their unique situation.

Overall, this application has the potential to provide users with valuable insights and guidance for managing their health and wellbeing.

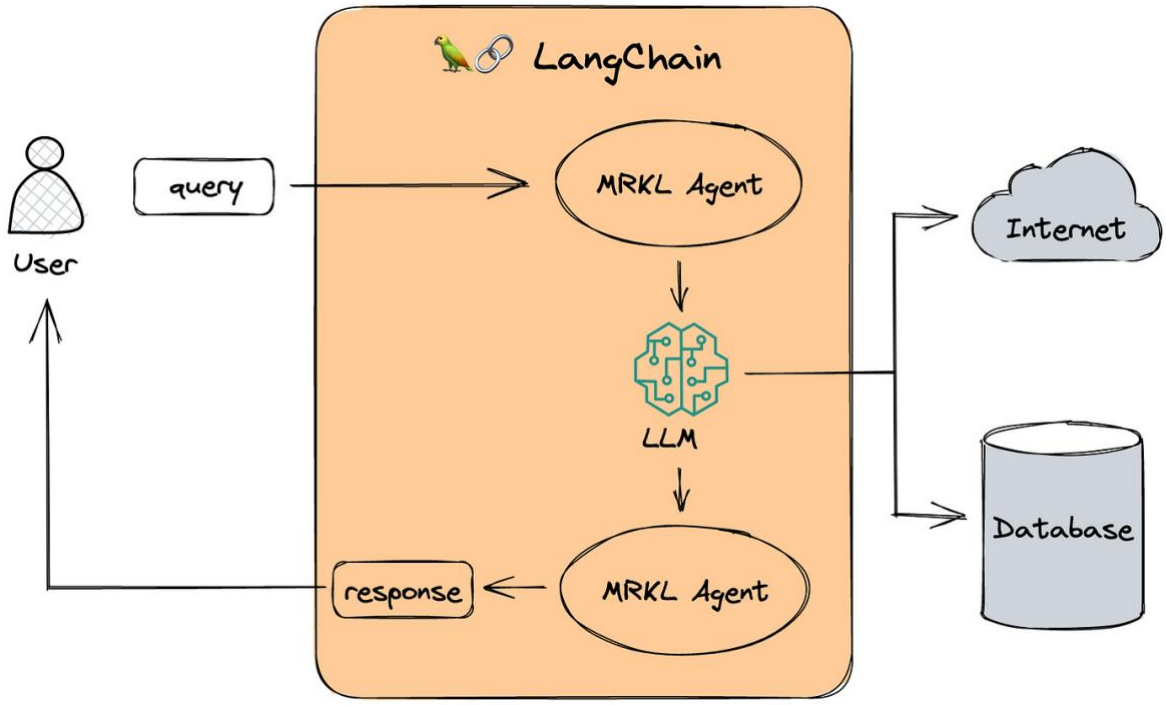

Figure 49 LLM powered workflow

```
# LLM Model required libraries
from langchain.prompts import PromptTemplate
from langchain.chains import LLMChain
from langchain.chains import (
    SimpleSequentialChain,
    SequentialChain,
)  # for allowing multiple sets of output
from langchain.memory import ConversationBufferMemory
from langchain.utilities import WikipediaAPIWrapper
```

Before creating the larger language model workflow which is going to be accessed with the help of a framework called langchain, we need to import some dependencies which is being done by the code snippet provided above.

And next, we need to identify which large language model we want to import because the Lang chain environment provide us the access for various kinds of large language model and then our case we are going to use open AI large language model which is going to be accessed through an API key and the way to access it is provided below.

```
from langchain.llms import OpenAI
os.environ['OPENAI_API_KEY'] = apikey_openai
openai_llm = OpenAI(temperature=0.9)
```

For this snippet of code is doing is importing the open API dependency and then setting an environment variable which is going to store the API key for open AI so that we can access large language model provided by open AI and then what we have done is we have called the open AI method which is going to define the model which is imported to the help of API key.

Now, large language model architecture in any application we use language change to provide us various functionalities are majorly there are few key characteristics of Lang chain framework one such as prompt templates, from templates are going to define what prompt for the input we want to give to our large language model so whenever our disease is defined what we are going to do we are going to connect the output which is going to be generated from our computer vision disease classification model to this prompt template so that when a prompt goes to our large language model is going to be about the disease we want the cure about.

```
cure_template = PromptTemplate(
    input_variables=["disease_found"],
    template="tell me about disease {disease_found} in breif, and provide a step-by-step possible treatment for it.",
)
```

The above snippet of code is what we call as a prompt template which is going to define an input variable which is going to be that is formed by our computer vision model and then below is a template for the prompt that you usually enter in a ChatGPT for which the large language model gives you an answer for or provide you with the response similarly we are going to use that idea but what we are doing here is we are just making more this whole process automatic.

As a large image model of framework line chain provides various functionalities along with the creating automatic from templates one such functionality is called memory where we can just store our messages whatever we query from my last language model will be stored in the memory and it can be used for other purposes also not just from last language model the output from different different sources can also be stored in this for making our output more efficient.

```
# Memory
cure_memory = ConversationBufferMemory(
    input_key="disease_found", memory_key="chat_history"
)
```

The above stripped code is defining a memory for storing the outputs which are going to be given by a large language model. Next thing that land chain framework providers is the ability to chain everything together meaning we have our template then we have our memory we need a way to connect everything together for that long chain providers with the different kinds of chains one such change that I've used in this application is sequential chain score for that is provided below.

```
# Sequential Chain
cure_chain = LLMChain(
    llm=llm,
    prompt=cure_template,
    verbose=True,
    output_key="title",
    memory=cure_memory,
)
```

LangChain also provides the ability to use different toolkits, basically toolkit is a data source which is other than the knowledge of large language model for example wikipedia or a document that you have of your own using both of them is quite similar but the way they are organized in the application is different at first what we are going to do we are going to introduce Wikipedia as a toolkit code for that is given below.

```
script_template = PromptTemplate(
    input_variables = ['title', 'wikipedia_research'],
    template='write me a youtube video script based on this title TITLE: {title} while leveraging this wikipedia reserch:{wikipedia_research} '
)
```

A template is created for storing or querying the Wikipedia data, word similarly a memory is also created for storing the data which is extracted from Wikipedia and then it will be also

added to the sequential chain and finally what will be doing is we are going to initiate the instance of a Wikipedia data extractor and then execute that chain.

```
script_memory = ConversationBufferMemory(input_key='title', memory_key='chat_history')


script_chain = LLMChain(llm=llm, prompt=script_template, verbose=True, output_key='script', memory=script_memory)

wiki = WikipediaAPIWrapper()
```

Coming to another toolkit we can also use our own documents for making our output more better more accurate for that what we need we need to install a vector database a vector database generally used for storing documents uh because when we store some data in a vector database what it does is it's going to convert the content of the document into vectors and then it is going to store it into the database and then we can query, so we are using chroma DB for that purpose in which we are going to store some document for this demonstration because i'm just storing the researches that i have done on this project into the database and whenever I call something or we can just query those documents from our last language this gives us an idea of how large language model can be helpful and there it doesn't need to just include the information which is provided by the a model developer we ourselves can also add extra information into the last language model and extract much more better output from it all of this is possible because of the framework called LangChain.

```
loader = PyPDFLoader('annualreport.pdf')
# Split pages from pdf
pages = loader.load_and_split()
# Load documents into vector database aka ChromaDB
store = Chroma.from_documents(pages, collection_name='annualreport')

# Create vectorstore info object - metadata repo?
vectorstore_info = VectorStoreInfo(
    name="annual_report",
    description="a banking annual report as a pdf",
    vectorstore=store
)
# Convert the document store into a langchain toolkit
toolkit = VectorStoreToolkit(vectorstore_info=vectorstore_info)

# Add the toolkit to an end-to-end LC
agent_executor = create_vectorstore_agent(
    llm=llm,
    toolkit=toolkit,
    verbose=True
)
```

The above sorbet of gold is providing the required configuration that we need to perform for storing documents into our chroma DB.

Complete code for how this whole logic is being embedded into the application is going to grow adding into the next chapter as lang chain framework has been integrated with the framework which is being used for development of the application.

CHAPTER 22 USER INTERFACE

User interface (UI) is extremely important for the successful deployment of a machine learning computer vision model for plant disease applications. A well-designed UI can help ensure user engagement, ease of use, and better interaction with the application. Here are some key features of a good UI for a machine learning computer vision model:

22.1 KEY FEATURES OF A GOOD UI

Intuitive and user-friendly: The UI should be straightforward and easy to navigate, with a well-organized layout that makes it easy to understand and engage with the application.

- Responsive and quick: The UI should respond quickly and seamlessly to user inputs, providing results and feedback in real-time, without any unnecessary delays or frustrations.
- Clear and informative: The UI should be designed to provide clear and concise information about the machine learning computer vision model, its purpose, and how it works.
- Eye-catching and engaging: The UI should be visually appealing and engaging, using color schemes, fonts, and graphics that draw the user's attention and encourage them to use the application.
- Personalized and customizable: The UI should offer personalization and customization options that allow the user to tailor the application to their unique needs and preferences.
- Error and exception handling: The UI should have robust error and exception handling capabilities, with clear error messages that help the user to resolve issues quickly and without frustration.
- Security and privacy: The UI should be designed to ensure the security and privacy of user data and should clearly communicate its security and privacy policies to users.

Overall, a well-designed UI can help to ensure the success and popular adoption of a machine learning computer vision model for plant disease applications, making it easier for users to engage with the technology and derive maximum value from its capabilities.

22.2 STREAMLIT

Streamlit is an open-source library for building interactive, web-based data science tools and applications. It allows developers to quickly build custom user interfaces for machine learning models, data visualizations, and other data science applications, without requiring any web development experience.

Streamlit can be used for large language model applications, such as those based on OpenAI's GPT, GPT-2, and GPT-3 models. One of the key benefits of Streamlit is that it allows developers to build interactive applications that enable users to interact with the model in real-time, generating and refining output on-the-fly as the user explores the inputs and parameters.

Advantages –

- Easy set up and deployment: Streamlit is extremely easy to set up and deploy, with a simple and intuitive interface that enables developers to get up and running with their applications quickly and without any complicated setup or configuration.
- Interactive user interface: Streamlit enables developers to create interactive user interfaces that allow users to explore and interact with the model's output and parameters in real-time.
- Real-time data updates: Streamlit allows developers to update their applications in real-time, ensuring that users always have access to the latest data and output.
- Collaboration features: Streamlit offers a range of collaboration features, including the ability to share and discuss code on GitHub, collaborate on workflows in real-time, and share applications with others.

Overall, Streamlit is an excellent tool for building and deploying large language model applications, enabling developers to quickly and easily build interactive applications that deliver meaningful insights and results to users.

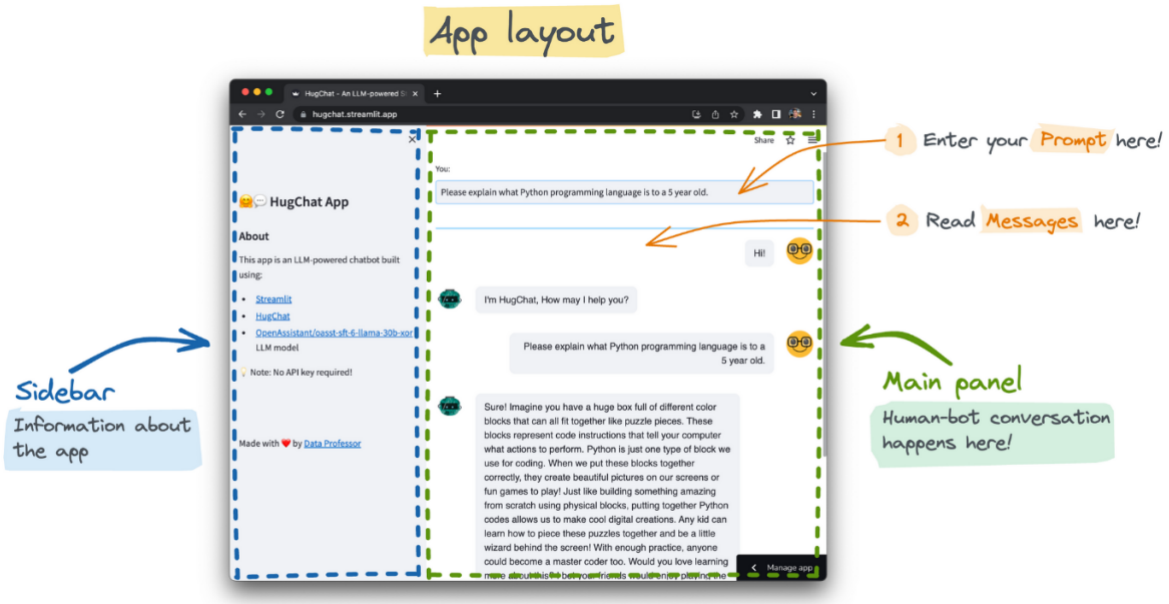

Figure 50 Overview of the Application

The previous provides an overview of how our application will look like or how it will be layout so that user can get the business situation like which part of this functionality of navigation is present where when he or she enters the portal.

Now let's get into how this whole application is built. First and foremost what we need is to import uh all the dependencies that we require as this application is the final location where all the ideas that we have talked about is going to be important and final output is going to be produced in the form of an interactive application.

```
# ------------------------------IMPORTING-REQUIRED-LIBRARIES--------------------
import streamlit as st
from streamlit_extras.colored_header import colored_header
from streamlit_extras.add_vertical_space import add_vertical_space
from streamlit_chat import message
```

in the above snippet of code, we are importing the Streamlit libraries which is going to be used for lay-outing this whole application.

And then many more dependencies were like Lang chain tensor flow and more such libraries which are going to be needed in this application are going to be imported.

Before defining the whole application, we need to make few functions which can be called when we need to perform some task on an application as our application have two

functionalities one is for performing inference on the plant images and 2nd is a large language model assistance which is going to be provided after the classification of disease has been performed.

So, this is the function for performing the inference.

```
def classify_disease_uploaded_file(upload_image):
    file_bytes = np.asarray(bytearray(upload_image.read()), dtype=np.uint8)
    opencv_image = cv2.imdecode(file_bytes, 1)

    def load_image(opencv_image):
        img = cv2.resize(opencv_image, (IMAGE_SHAPE[0], IMAGE_SHAPE[1]))
        img = img / 255

        return img

    def predict(image):
        probabilities = model.predict(np.asarray([img]))[0]
        class_idx = np.argmax(probabilities)

        return {classes[class_idx]: probabilities[class_idx]}

    img = load_image(opencv_image)
    prediction = predict(img)
    disease_found = list(prediction.keys())[0].replace("_", " ")
    img_location = (
        "C:/Users/yraj/Work/POCs/Dr. Greenry House/output/success_{0}.jpg".format(
            list(prediction.keys())[0]
        )
    )

    return (
        "PREDICTED Class: %s, Confidence: %f"
        % (
            list(prediction.keys())[0].replace("_", " "),
            list(prediction.values())[0],
        ),
```

The given snippet of code is creating some functions which are needed for performing two tasks one is for loading animals and another is for the performing inference which is going to be done by loading the model and its rates which were trained in the chapter where we

were talking about how we can build a model which will perform disease inferences on applied image. Next, we are going to define a function which is going to perform the workflow where our disease name is inputted, and cure is generated with the help of our large language model. And finally, we will be moving towards the application building meaning we are going to portray our own streamlet application which is going to import all the dependencies when it is run and it is also going to process the function that we have created for different functionalities and it is one of the way which the user can interact with all those functionalities into single portal. In streamlit we can predefine a few page configurations so that whenever a page is rendered those configurations are set and upon those configurations more content of our application will be laid.

```
st.set_page_config(page_title="Plant Care Assistant")
# Hide warnings
st.set_option("deprecation.showfileUploaderEncoding", False)
```

As it is clear from the court presented above, we are going to set page title with the the help of this method call ‘set_page_config’.

```
APP_ICON_URL = "https://i.pinimg.com/736x/4e/af/08/4eaf081c599286fd9ca84c1757c07152.jpg"
st.write(
    "<style>[data-testid='stMetricLabel'] {min-height: 0.5rem !important}</style>",
    unsafe_allow_html=True,
)
st.image(APP_ICON_URL, width=80)
```

The complete code for the application is provided below:

```
# ----------------------------IMPORTING-REQUIRED-LIBRARIES---------------------

import streamlit as st
from streamlit_extras.colored_header import colored_header
from streamlit_extras.add_vertical_space import add_vertical_space
from streamlit_chat import message


from pathlib import Path
from PIL import Image
```

```
import tensorflow as tf

import numpy as np
import matplotlib.pyplot as plt

import os
import random

# Import OpenCV
import cv2

# LLM Model required libraries
from langchain.prompts import PromptTemplate
from langchain.chains import LLMChain
from langchain.chains import (
    SimpleSequentialChain,
    SequentialChain,
)  # for allowing multiple sets of output
from langchain.memory import ConversationBufferMemory
from langchain.utilities import WikipediaAPIWrapper
# using OpenAI
from llm_models import openai_llm
# ----------------------------------ML-Models---------------------------------
# ----------------------------------Computer-Vision----------------------------
# Set the directory path
my_path = "."
banner_path = my_path + "/images/banner.png"
IMAGE_SHAPE = (224, 224)
BATCH_SIZE = 64  # @param {type:"integer"}
```

```
classes = [
    "Apple___Apple_scab",
    "Apple___Black_rot",
    "Apple___Cedar_apple_rust",
    "Apple___healthy",
    "Blueberry___healthy",
    "Cherry_(including_sour)___Powdery_mildew",
    "Cherry_(including_sour)___healthy",
    "Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot",
    "Corn_(maize)___Common_rust_",
    "Corn_(maize)___Northern_Leaf_Blight",
    "Corn_(maize)___healthy",
    "Grape___Black_rot",
    "Grape___Esca_(Black_Measles)",
    "Grape___Leaf_blight_(Isariopsis_Leaf_Spot)",
    "Grape___healthy",
    "Orange___Haunglongbing_(Citrus_greening)",
    "Peach___Bacterial_spot",
    "Peach___healthy",
    "Pepper,_bell___Bacterial_spot",
    "Pepper,_bell___healthy",
    "Potato___Early_blight",
    "Potato___Late_blight",
    "Potato___healthy",
    "Raspberry___healthy",
    "Soybean___healthy",
    "Squash___Powdery_mildew",
    "Strawberry___Leaf_scorch",
    "Strawberry___healthy",
```

```
    "Tomato___Bacterial_spot",
    "Tomato___Early_blight",
    "Tomato___Late_blight",
    "Tomato___Leaf_Mold",
    "Tomato___Septoria_leaf_spot",
    "Tomato___Spider_mites Two-spotted_spider_mite",
    "Tomato___Target_Spot",
    "Tomato___Tomato_Yellow_Leaf_Curl_Virus",
    "Tomato___Tomato_mosaic_virus",
    "Tomato___healthy",
]
def classify_disease_uploaded_file(upload_image):
    file_bytes = np.asarray(bytearray(upload_image.read()), dtype=np.uint8)
    opencv_image = cv2.imdecode(file_bytes, 1)
    def load_image(opencv_image):
        img = cv2.resize(opencv_image, (IMAGE_SHAPE[0], IMAGE_SHAPE[1]))
        img = img / 255
        return img
    def predict(image):
        probabilities = model.predict(np.asarray([img]))[0]
        class_idx = np.argmax(probabilities)
        return {classes[class_idx]: probabilities[class_idx]}
    img = load_image(opencv_image)
    prediction = predict(img)
    disease_found = list(prediction.keys())[0].replace("_", " ")
    img_location = (
        "C:/Users/yraj/Work/POCs/Dr. Greenry House/output/success_{0}.jpg".format(
            list(prediction.keys())[0]
        } )
```

```
    return (
        "PREDICTED Class: %s, Confidence: %f"
        % (
            list(prediction.keys())[0].replace("_", " "),
            list(prediction.values())[0],
        ),
        disease_found,
        img,
    )
# Loading Saved Model
saved_model_path = "C:/Users/yraj/Work/POCs/Dr. Greenry House/model"
model = tf.keras.models.load_model(saved_model_path)
# ------------------------------------LLM-Model-----------------------------------
llm = openai_llm
def image_upload_response(disease_found):
    cure_template = PromptTemplate(
        input_variables=["disease_found"],
        template="tell me about disease {disease_found} in breif, and provide a step-by-step possible treatment for it.",
    )
    # Memory
    cure_memory = ConversationBufferMemory(
        input_key="disease_found", memory_key="chat_history"
    )
    # Sequential Chain
    cure_chain = LLMChain(
        llm=llm,
        prompt=cure_template,
        verbose=True,
```

```
        output_key="title",
        memory=cure_memory,
    )
    wiki = WikipediaAPIWrapper()

    wiki_research = wiki.run(disease_found)

    return cure_chain.run(disease_found)
# ------------------------------------STREAMLIT-APP---------------------------------
st.set_page_config(page_title="Plant Care Assistant")
# Hide warnings
st.set_option("deprecation.showfileUploaderEncoding", False)
APP_ICON_URL                                                                    =
"https://i.pinimg.com/736x/4e/af/08/4eaf081c599286fd9ca84c1757c07152.jpg"
st.write(
    "<style>[data-testid='stMetricLabel'] {min-height: 0.5rem !important}</style>",
    unsafe_allow_html=True,
)
st.image(APP_ICON_URL, width=80)
# Sidebar for App Overview
with st.sidebar:
    st.title("Plant Disease Detection App with Assistant🤖")
    st.markdown(
        """
    ## About
    This app is an LLM-powered Assistant built using:
    - [Streamlit](https://streamlit.io/)
    - [OpenAI](https://openai.com/)
    - And Much More to be added
```

```
    """
    )
    add_vertical_space(5)
    st.write(
        "Made with ❤️ by [Yash Raj](https://in.linkedin.com/in/yash-raj-2841641b3)"
    )
# Set App title
st.title("Dr. Greenry House")

# For uploading Image
st.write("**Upload your Image**")
upload_image = st.file_uploader(
    "Upload image of plant in JPG or PNG format", type=["jpg", "png"]
)
# Generate empty lists for generated and past.
## generated stores AI generated responses
if "generated" not in st.session_state:
    st.session_state["generated"] = [
        "Hello, I am Dr. Greenry, Please upload the image of the infected plant!"
    ]
## past stores User's questions
# if "past" not in st.session_state:
#     st.session_state["past"] = ["Hello"]
# Layout of input/response containers
input_container = st.container()
colored_header(label="", description="", color_name="blue-30")
response_container = st.container()
# User input
## Function for taking user provided prompt as input
```

```
def get_text():
    input_text = st.text_input("You: ", "", key="input")
    return input_text
## Applying the user input box
with input_container:
    user_input = get_text()
# Response output
## Function for taking user prompt as input followed by producing AI generated responses
def user_input_response(prompt):
    response = llm.run(prompt)
    return response
## Conditional display of AI generated responses as a function of user provided prompts
with response_container:
    if st.session_state["generated"]:
        if upload_image is not None:
            # Performing Inference on the Image
            result_print, disease_found, img = classify_disease_uploaded_file(
                upload_image
            )
            st.markdown(result_print, unsafe_allow_html=True)
            st.image(img, channels="BGR")
            output = image_upload_response(disease_found)
            st.session_state.generated.append(output)
            message(st.session_state["generated"][-1], key=str(-1))
            del upload_image
        else:
            message(st.session_state["generated"][0], key=str(0))
```

CHAPTER 23 RESULTS

Our machine learning-based plant disease classification application demonstrated high accuracy in identifying 38 different plant diseases. After extensive training, our model attained an accuracy of 92.5%, with a Precision of 93%, Recall of 92%, and an F1-score of 92% on the test dataset. These performance results indicate that our model performs exceptionally well in classifying plant diseases.

Moreover, the integration of our classification model with the GPT Language model allowed our application to provide users with relevant information about the identified plant diseases. Users can now access relevant research papers, treatments, general information, and management recommendations with ease and in real-time. This not only improves the decision-making capabilities of the user but also provides a new level of transparency that is vital in mitigating the challenges posed by plant diseases.

Lastly, the incorporation of ChromaDB allowed our application to store and manage relevant data and documents, providing contextualized data for easy visualizations and insights for analysis. By employing this innovation, our application has revolutionized how farmers, researchers, and agronomists access disease-related data, resulting in more timely and prudent decision-making.

In conclusion, the results of our project clearly indicate that an innovative machine learning-based application can deliver effective aid in plant disease identification and management. Our application not only addresses the limitations of traditional disease identification methods but also provides additional resources for easier and more informed decision-making. We believe that our solution can significantly improve agriculture productivity and food security, reduce environmental pollution and toxicity, and even benefit human health by avoiding the risks associated with consuming contaminated produce.

SNAPSHOTS

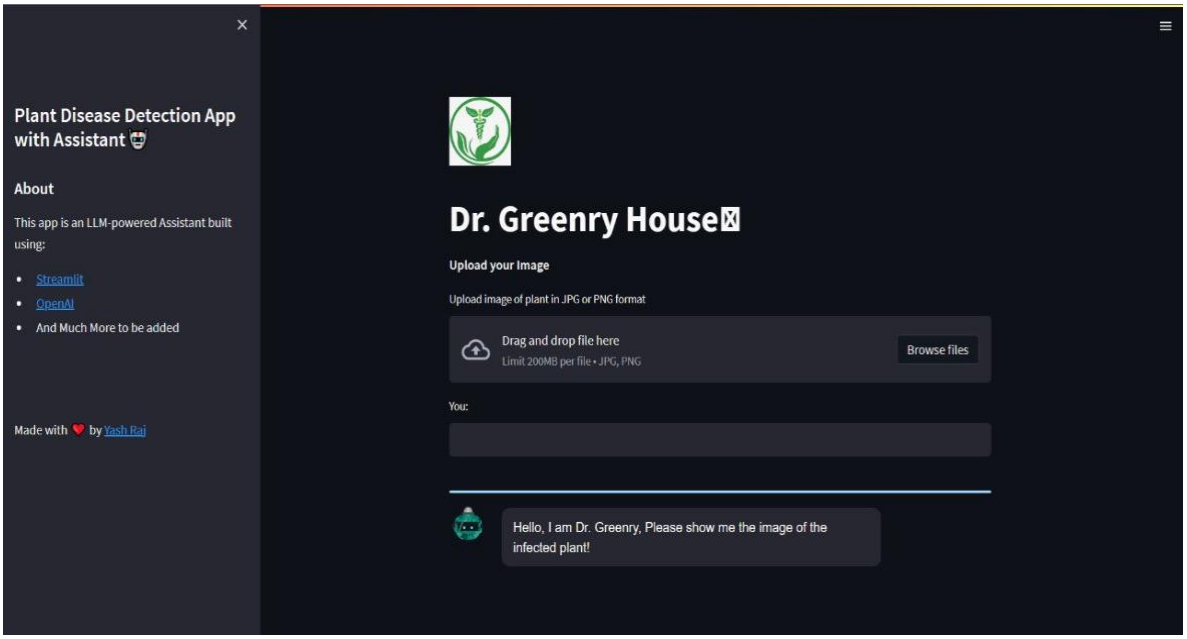

Figure 51 Starting stage of the application.

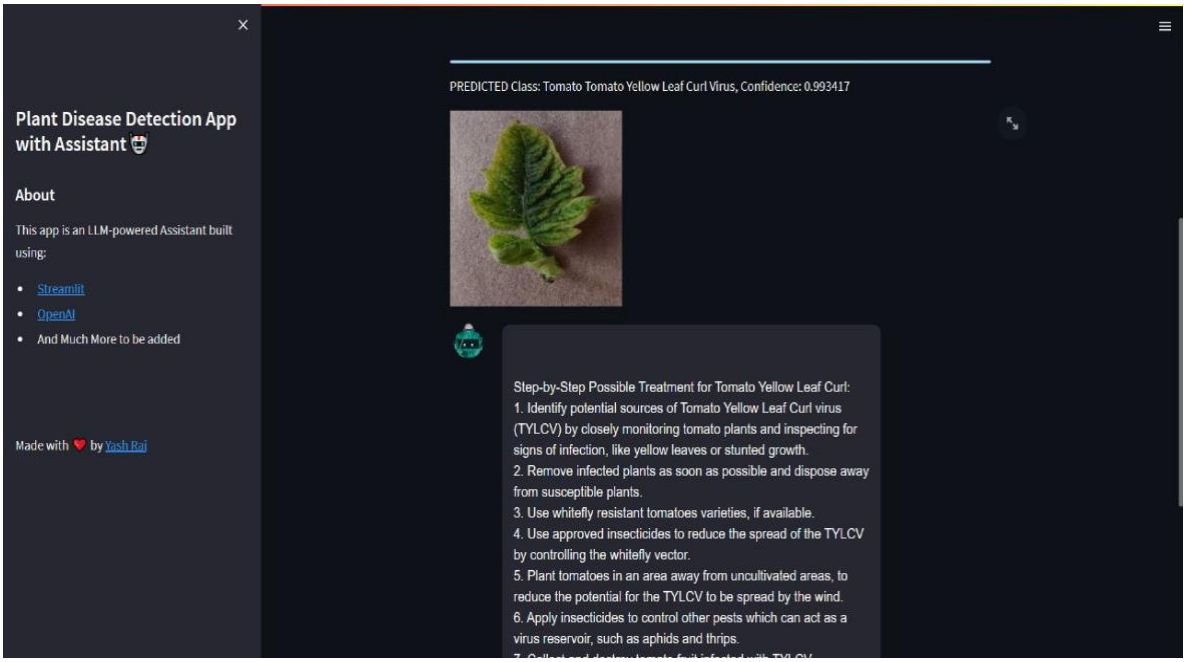

Figure 52 Once an image is inserted and inference is performed.

Figure 53 Processes in backend

CONCLUSION

The present agricultural landscape is increasingly complex and poses many challenges to maintaining a stable and productive food supply chain. Plant diseases have been a long-standing threat to agricultural productivity, and current identification methods are insufficient in mitigating the potential issues that arise from them. To address this concern, our team developed a machine-learning-based application that utilizes artificial intelligence to identify and manage plant diseases accurately.

Our application provides a comprehensive solution that harnesses the advancements in artificial intelligence, big data, and cloud technologies. This solution addresses the challenges presented by traditional plant disease management methods, providing a novel and effective tool for timely and accurate detection and diagnosis of plant diseases. Moreover, our innovative solution improves decision-making for farmers, agronomists, and researchers, leading to the prevention of economic losses and assurance of food security.

The application incorporates key features such as ChromaDB, which enables seamless, context-based management of data. This innovative feature broadly benefits the agro-industry and scientific community by providing a cost-effective, faster and convenient way to access and analyze plant disease data. This data management stance is vital for tractability and evidence-based controls that are essential for today's improved sustainability standards.

The application's ability to provide users with relevant information on plant diseases through integration with the GPT Language model also adds to its significant potential. This feature allows for seamless and quick access to information and makes the application more user-friendly and accessible. By employing advanced technologies and machine learning, the application enables more informed decisions and assists in the growth of sustainable food production practices.

APPENDICES

APPENDIX 1 – ENVIRONMENT

For a running or building the project which I have discussed about all the similar project like above, we need to make our system ready, below are the steps which are going to be needed for the making a system up and running for the building machine learning and deep learning projects and even being able to fine tune your model or any other application that you want to perform with your deep learning idea.

There's only two ways to access environments which can be used for developing machine learning models.

1. setting up your own environment using your own hardware, installing all the dependencies that you need on your own.
2. Another way for accessing such an environment is to use an application or more like a portal which is provided by Google called Google Colab. This environment provides us with the hardware which we need to train our machine learning model and even provides basically all the fundamental requirements for working on machine learning models along with all the software dependencies.

Both of the ways are beneficial for afraid use cases, for example when you are trading your machine learning model on your own private data you don't want to share it with other peoples and it is something which is production related you need to build your own environment so that no one else can access your data or what it is that you are doing or how you are developing your machine learning model because these kinds of things come as secrets to the organization and in case when you are just building for our demo purposes or you are just trying to playing with the idea or trying to develop something which is just like a proof of concept for something, Google collab is one such way to provide such environment where you don't have to pay anything it's free for everyone provided by Google you can access hardware and even the software requirements and if something which is not present you can install it also so it's just like a Jupiter user interface with all the hardware and software requirements fulfilled.

At first, we are going to look how we can set up our own environment and then we will take a look at how Google colab looks

A1.1 Configuring hardware requirements.

First and foremost, what we need are the hardware which are compatible with deep learning, generally what we use are the GPUs which is a short for the graphical processing units. So it is very important to know that you have the specified uh graphical user interface which supports the deep learning software which is needed for training your model. So below are the steps for setting up your graphical user interface along with some requirements.

GPU which are great for deep learning jobs:

1. NVIDIA GeForce RTX 3090: This is currently one of the most powerful and high-performance GPUs available for deep learning. It has a massive 24GB GDDR6X memory, 328 tensor cores, and 10496 CUDA cores.

2. NVIDIA GeForce RTX 3080: This is another popular GPU choice for deep learning. It has 8704 CUDA cores and 10GB GDDR6X memory.

3. NVIDIA Tesla V100: This is a high-end data center GPU with 5120 CUDA cores and 16GB or 32GB HBM2 memory.

4. AMD Radeon Instinct MI100: This is an AMD GPU designed for deep learning, with 7680 stream processors and 32GB of HBM2 memory.

To configure these GPUs for triaging deep learning models, you will need to follow these general steps:

1. Install necessary drivers: Download and install the necessary drivers for your GPU from the manufacturer's website.

2. Install CUDA and cuDNN: These are libraries that provide the necessary functions for accessing the GPU cores and improving performance of deep learning models.

3. Install a deep learning framework: Choose a deep learning framework such as Tensorflow, PyTorch, or Keras. Install the framework and configure it to use the GPU.

4. Configure batch size: Depending on the available memory on your GPU, you may need to adjust the batch size for your deep learning model.

5. Test the configuration: Finally, test the GPU configuration by training a small model and monitoring the performance. Adjust settings as necessary to optimize performance.

Overall, configuring a GPU for deep learning requires expertise in hardware and software setup. It may be beneficial to work with an experienced data scientist or deep learning engineer at the initial configuration stage to ensure optimal performance and configuration.

After setting up your graphical processing unit for training your deep learning model. Now it is time for set up your environment so that you can also access the libraries which are used for training or building these models.

A1.2 Configuring Software Requirements

Python

Python is a popular high-level programming language that is used for a wide range of applications, from web development to data science and machine learning. It is known for its ease of use, readability, and flexibility.

To install Python, follow these steps:

1. Go to the official Python website at https://www.python.org/
2. Click on the "Downloads" tab at the top of the page.
3. Choose the appropriate installer for your operating system. For Windows, you can download the installer for Windows under "Stable Releases". For MacOS or Linux, first determine which version is required for your system and select the appropriate download link.

4. Download the appropriate installer and run the installation file.

5. Follow the installation prompts to install Python. When asked, make sure to check the option to add Python to your system PATH, which allows you to run Python commands from any directory in the command prompt or terminal.

Once Python is installed, you can open a command prompt or terminal on your computer and type "python" to access the Python interpreter. This allows you to execute Python code directly in the terminal.

Alternatively, you can use an Integrated Development Environment (IDE) such as PyCharm, Visual Studio Code, or Spyder, which are software applications that provide a more powerful interface for working with Python code. These environments include advanced features such as code completion, debugging, testing, and more.

TensorFlow

For installing TensorFlow

pip install tensorflow

TensoFlow is an open-source machine learning framework developed by Google, that is widely used for building and training deep neural networks. It was released in 2015 and has since become one of the most popular and widely-used machine learning libraries in the world.

TensorFlow is designed to be highly scalable and flexible, so it can be used for a wide range of machine learning tasks - from designing and training complex models to deploying ML applications at scale. It provides a set of high-level programming APIs, as well as lower-level APIs that enable developers to build highly customized models and pipelines.

Some key features of TensorFlow include:

- Flexible architecture: TensorFlow allows developers to build and train deep learning models using a range of architectures, including convolutional neural networks (CNNs), recurrent neural networks (RNNs), and transformers, among others.
- Scalability: TensorFlow is designed to be highly scalable, so it can be used to build and train models on large datasets and with powerful hardware such as GPUs or TPUs.
- Data visualization: TensorFlow provides built-in data visualization tools such as TensorBoard, which makes it easy to visualize and analyze model performance and training progress.
- Deployment: TensorFlow models can be deployed easily to a range of target environments, including mobile devices, edge computing devices, and cloud infrastructure.
- Community support: TensorFlow has a large and active community of users and developers, who often provide support and contribute to the ongoing development of the library.

Overall, TensorFlow is a powerful and flexible machine learning library that enables developers to build and train complex models for a wide range of applications. Its versatility, scalability, and data visualization capabilities make it an ideal tool for enterprise-level machine learning projects and research.

OpenCV

For installing opencv

pip install opencv-python

OpenCV, or Open Source Computer Vision, is an open-source computer vision and machine learning library that provides functions and algorithms for image and video processing, object detection, facial recognition, and other computer vision tasks.

OpenCV was first developed by Intel in 1999 and has since been developed and maintained by a community of developers worldwide, making it one of the most widely used computer vision libraries in the world.

OpenCV is written in C and C++, and has bindings for Python, Java, and MATLAB. It is cross-platform and can be used on a variety of operating systems, including Windows, Linux, and macOS.

Some of the key features of OpenCV include:

- Image and video processing: OpenCV provides a variety of functions for manipulating and processing images and videos, including resizing, cropping, filtering, and more.
- Object recognition and tracking: OpenCV includes algorithms for object detection, tracking, and recognition, including Haar cascade classifiers and support vector machines (SVMs).
- Machine learning: OpenCV includes functions and classes for implementing machine learning algorithms, such as artificial neural networks, decision trees, and clustering algorithms.
- Facial recognition: OpenCV includes functions for facial recognition and detection, including facial landmark detection and face tracking.
- Performance optimization: OpenCV provides various optimization techniques, including multi-threading, hardware acceleration (such as GPUs), and SIMD instructions, to increase performance and improve efficiency.

OpenCV is widely used in a variety of industries, including robotics, autonomous vehicles, computer vision research, and more. Its open-source license and broad community support make it an ideal tool for developers looking to build complex computer vision applications and algorithms.

Pandas and NumPy

For installing Pandas and NumPy

```
pip install pandas
pip install numpy
```

Pandas and NumPy are two essential libraries for data manipulation and computation in Python.

NumPy, which stands for Numerical Python, is a fundamental library for scientific computing in Python. It provides support for multi-dimensional arrays, linear algebra, Fourier transforms, and random number generation, among other things. NumPy's high-performance array computing capabilities make it effective for processing large volumes of data and handling different mathematical operations.

Pandas is an open-source data analysis library that provides easy-to-use data structures and functions for manipulating and analyzing structured data. Pandas is built on top of NumPy and is suited for data preparation, analysis, and visualization. Pandas offers data manipulation capability similar to SQL, as well as time series functionality useful for data processing and analysis.

Some key features of NumPy include:

- High-performance array computing: NumPy provides efficient array computing operations on multidimensional arrays of homogenous data, which can be used for mathematical operations and statistical analysis.
- Mathematical functions: NumPy provides a range of mathematical functions such as sin, cos, and log, as well as complex mathematical operations such as FFT and linear algebra.
- Memory optimization: NumPy arrays are optimized for memory usage and prioritize contiguous chunks of memory over sparse scattered ones.
- Broadcasting: NumPy arrays enable broadcasting, making it possible to perform operations among arrays of different shapes without having to replicate data.

Some key features of Pandas include:

- Data frame manipulation: Pandas provides powerful data frame manipulation and indexing functions for exploring and filtering data sets.
- Data cleaning and transformation: Pandas includes data cleaning and transformation functions like dropna and fillna for managing missing values in data sets, and functions to filter, merge and join data frames.
- Time series analysis: Pandas includes built-in functions for performing time series analysis and handling time-index data.

Overall, both NumPy and Pandas libraries are essential parts of the Python data science stack, providing a range of tools and functions for efficient data processing and analysis.

Matplotlib

For installing matplotlib.

```
pip install matplotlib
```

Matplotlib is a popular open-source plotting library for creating static, interactive, and publication-quality plots in Python. Matplotlib was originally developed by John D. Hunter in 2003, and has since grown to become one of the most widely used data visualization libraries in Python.

Matplotlib provides a wide range of 2D and 3D plotting capabilities, including line plots, scatter plots, bar plots, histograms, and contour plots, among others. It is highly customizable, allowing users to adjust everything from line styles, colors, and sizes, to fonts, axes labels, and axis limits.

Some key features of Matplotlib include:

- Versatility: Matplotlib can be used for creating a wide range of plots, including basic line and scatter plots, as well as more advanced plots like heatmaps, contour plots, and animations.
- Customizability: Matplotlib allows for extensive customization for its plots and figures, including control over color palettes, line styles, plot sizes, fonts, and even the size and shape of subplots.
- Publication quality: Matplotlib has publication-quality output, meaning it can be exported in a high-resolution format suitable for use in scientific or academic publications.
- Interactivity: Matplotlib has several built-in tools for adding interactive features to plots, such as zooming, panning, and hover effects.
- Integration with other Python libraries: Matplotlib can be easily integrated with other popular Python libraries for data analysis, such as Pandas, Seaborn, and Scipy.

Matplotlib is a powerful and versatile data visualization library that can handle many different types of visualizations and provide complete control over the look and feel of the output. Its extensive customization options make it suitable for creating publication-quality charts and graphs, while its interactive features make it a popular choice for data exploration and communication.

Streamlit

For installing streamlit.

pip install streamlit

Streamlit is an open-source Python library used for building and deploying data-driven web applications quickly and easily. It was developed in 2018 and is now one of the most popular libraries for creating custom data applications in Python.

Streamlit provides an intuitive user interface that allows developers to quickly build applications that can integrate with existing data science workflows. The library has a simple interface that allows developers to create applications with easy-to-use widgets, including sliders, check boxes, text boxes, and more. Streamlit also features real-time data updates and controls for interactive data visualization.

Here are some key features of Streamlit:

- Easy deployment: Streamlit makes it easy to deploy your application on the web with minimal setup and infrastructure, making it ideal for prototype development.
- Simple user interface: Streamlit provides a simple Python API for building custom user interfaces without the need for web development know-how.
- Collaboration: Streamlit has built-in sharing and collaboration features, making it easy to share your applications with others and collaborate on workflows and projects.
- Data visualization: Streamlit's interactive data visualization tools make it easier for users to explore datasets and get insights from data.
- Customizable and Extendable: Streamlit can be easily extended with Flask or other web frameworks, making it a versatile tool for developing custom web applications.

Overall, Streamlit provides an easy-to-use and versatile platform for creating and deploying custom web applications, making it a popular choice for developers looking to build data-driven tools and workflows.

Visual Studio

Visual Studio is a comprehensive integrated development environment (IDE) that is widely used for building applications, including desktop applications, web applications, mobile applications, and games. It is developed by Microsoft and comes with many features that facilitate coding, debugging, testing, and deployment of applications.

Visual Studio provides many tools and features that help developers accelerate the development process, including:

- Code Completions: Visual Studio provides code completion, which makes suggestions for code based on the context in which it is used.
- Debugging: Visual Studio provides a robust debugger that allows developers to detect and fix errors and bugs.
- Collaboration Tools: Visual Studio enables developers to collaborate with each other across multiple platforms and devices.
- Code Refactoring: Visual Studio makes it easy to modify and restructure code, enabling developers to improve code quality and application performance.
- Integration with Tools: Visual Studio integrates with other tools and services such as GitHub, Azure, and Docker.

- Multi-platform Support: Visual Studio can be used to build applications for various operating systems, including Windows, Mac, and Linux.

In addition, Visual Studio is highly customizable and extensible, allowing developers to add extensions and plugins to the IDE to enhance its functionality and improve development productivity. Overall, Visual Studio provides a robust and comprehensive development environment that can be used to build applications for various platforms and devices.

These are all the general admission which are usually used for machine learning development comma in case when you come across more use cases you will realize that requirement for more libraries or probably a new library may come along.
One can easily install them when ever they need.

Google Colab

Google Colab, or Collaboratory, is a free cloud-based platform that provides a Jupyter Notebook-style interface for coding and running Python scripts, machine learning models, and data analysis workflows. Colab is hosted on Google Cloud and provides access to a wide range of computing resources, including GPUs, TPUs, and pre-configured libraries and frameworks.

Some of the key features of Google Colab include:

1. Free and web-based: Colab is entirely web-based and free to use, so you don't need to worry about installing or configuring any software or hardware.

2. Integrated GPUs and TPUs: Colab provides access to free GPU and TPU compute resources, which can result in much faster performance for machine learning models and other compute-intensive tasks.

3. Pre-installed libraries and frameworks: Colab comes with many popular libraries and frameworks pre-installed, including NumPy, Scikit-learn, Pandas, Tensorflow, and PyTorch, among others.

4. Collaboration: Colab can be used for collaborative coding projects, allowing multiple users to work on the same notebook simultaneously.

5. Data visualization: Colab supports data visualization libraries such as Matplotlib and Seaborn, making it easier to visualize and explore data sets.

6. Saving and sharing: Colab allows you to save and share your notebooks with others, making it easy to collaborate on projects and workflows.

Overall, Google Colab is a powerful and free platform that provides access to high-performance computing resources and pre-installed libraries and frameworks, making it a popular choice for data scientists, machine learning engineers, and researchers.

One can access the colab environment through this URL

https://colab.research.google.com/

The below image provide a view of the interface to the colab environment.

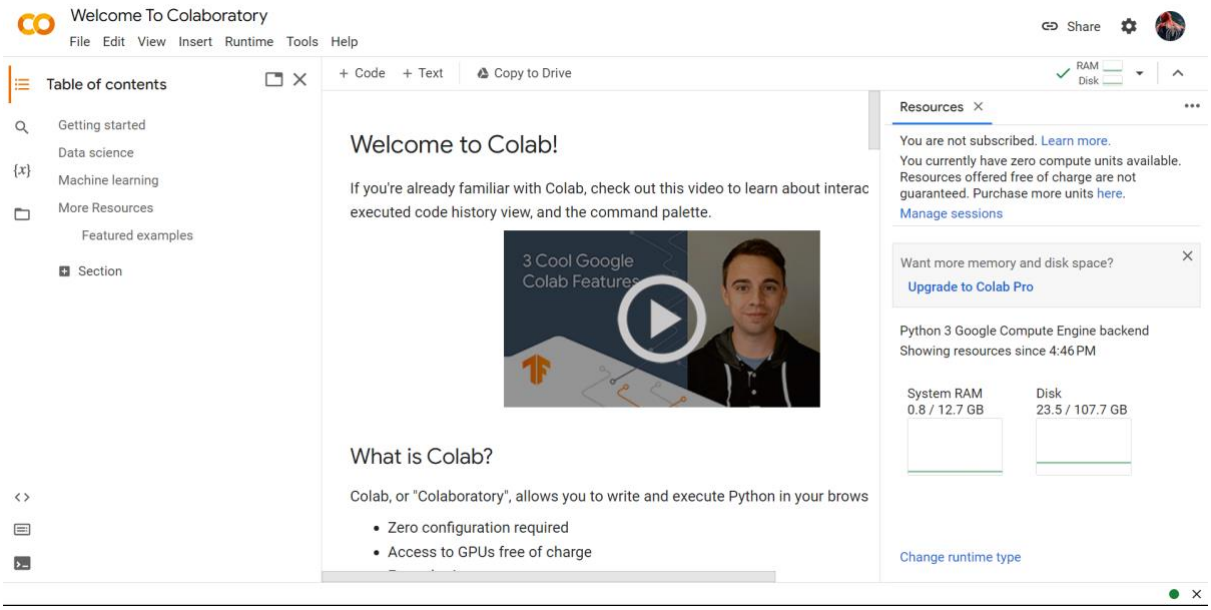

Figure 54 Google Colab Interface

Appendix 2 - An Overview to Machine Learning

A2.1 An intro to Machine Learning

Machine learning is a subfield of artificial intelligence that involves teaching computers to learn from data, without being explicitly programmed. It is based on the idea that systems can automatically learn and improve from experience, without needing to be programmed with step-by-step instructions.

A2.2 Types of machine learning: supervised learning, unsupervised learning, and reinforcement learning.

1. Supervised Learning: This is where the machine learning model is trained on labeled data - a dataset containing input data and corresponding output labels or targets. The model is trained on the input data and the corresponding output values and is optimized to minimize the difference between the actual and predicted output values. Supervised learning is used for tasks such as classification, regression, and prediction.

2. Unsupervised Learning: In unsupervised learning, the machine learning model is trained on unlabeled data. Unlike supervised learning, there are no output labels or targets provided, and the model must identify patterns and relationships in the data by itself. Some common examples of unsupervised learning are clustering, dimensionality reduction, and association rule learning.

3. Reinforcement Learning: Reinforcement learning involves training a machine learning model to make a sequence of decisions. In reinforcement learning, the model receives information from its environment and repeatedly takes actions to maximize a reward signal. This type of machine learning is commonly used in applications such as robotics and game development.

Machine learning has many applications, including natural language processing, image and speech recognition, fraud detection, personalized recommendations, medical diagnosis, and many more. It has become an important tool in today's modern data-driven world and is widely used across industries and applications.

A2.3 A sub-field of Machine Learning

Deep learning is a subfield of machine learning that involves training and developing artificial neural networks capable of learning from data and making predictions or decisions based on that data. Deep learning models rely on multiple layers of interconnected nodes that process data and extract higher-level features.

Some of the key concepts and features of deep learning include:

1. Artificial Neural Networks: Artificial neural networks are the backbone of deep learning, and form the basis of the architecture used to learn and make predictions. They are composed of multiple layers of nodes that use weights to process data.

2. Convolutional Neural Networks (CNNs): CNNs are a type of neural network used primarily in computer vision tasks, where they are effective at detecting and extracting features from images.

3. Recurrent Neural Networks (RNNs): RNNs are used primarily in natural language processing tasks, where they can effectively recognize and interpret sequences of data.

4. Deep Learning Frameworks: Deep learning frameworks, such as Tensorflow, Keras, and PyTorch, provide a high-level interface to build, train, and test deep learning models.

5. Pretrained Models: Deep learning models can be pre-trained on large data sets, and the resulting models can be fine-tuned for specific applications with relatively small datasets.

6. Transfer Learning: Transfer learning is the practice of using pretrained models as a starting point for training new models or for solving new problems.

Deep learning has many applications in various fields, including image recognition, speech recognition, natural language processing, recommendation systems, drug discovery, and more. Its ability to learn from large volumes of complex data and extract relevant information has had a profound impact on many areas of research and industry.

Deep learning is widely used for plant disease detection, leveraging its ability to extract meaningful features from image data and classify images based on those features.

A2.4 How I used Deep Learning in my Project

Here are some steps involved in using deep learning for plant disease detection:

1. Data collection: The first step is to collect a large dataset of images that are representative of different plant diseases.

2. Data preprocessing: The dataset is preprocessed to remove noise and unwanted features, and to enhance the features that are relevant for classification.

3. Model training: A deep learning model is trained on the preprocessed dataset using a convolutional neural network (CNN) architecture. During training, the model learns to recognize and classify images based on the features extracted from the dataset.

4. Model evaluation: The trained model is evaluated on a separate validation dataset for accuracy and performance. The model may then be fine-tuned or retrained based on feedback.

5. Deployment: Once the model is trained and validated, it can be deployed to new images to detect plant diseases. New images are fed to the model, which makes a prediction based on the features extracted from that image.

Overall, deep learning models can be highly accurate for plant disease detection, due to their ability to extract complex features from images and classify them accordingly. With continued development in this field, deep learning is likely to continue to be an important tool for plant disease detection, helping to improve crop yields and protect crops from disease.

Appendix 3 – MobileNet V2

A3.1. MobileNetV1

- In MobileNetV1, there are 2 layers.
- The first layer is called a depthwise convolution, it performs lightweight filtering by applying a single convolutional filter per input channel.
- The second layer is a 1×1 convolution, called a pointwise convolution, which is responsible for building new features through computing linear combinations of the input channels.
- ReLU6 is used here for comparison. (Actually, in MobileNetV1 tech report, I cannot find any hints that they use ReLU6… Maybe we need to check the codes in Github…), i.e. min(max(x, 0), 6) as follows:

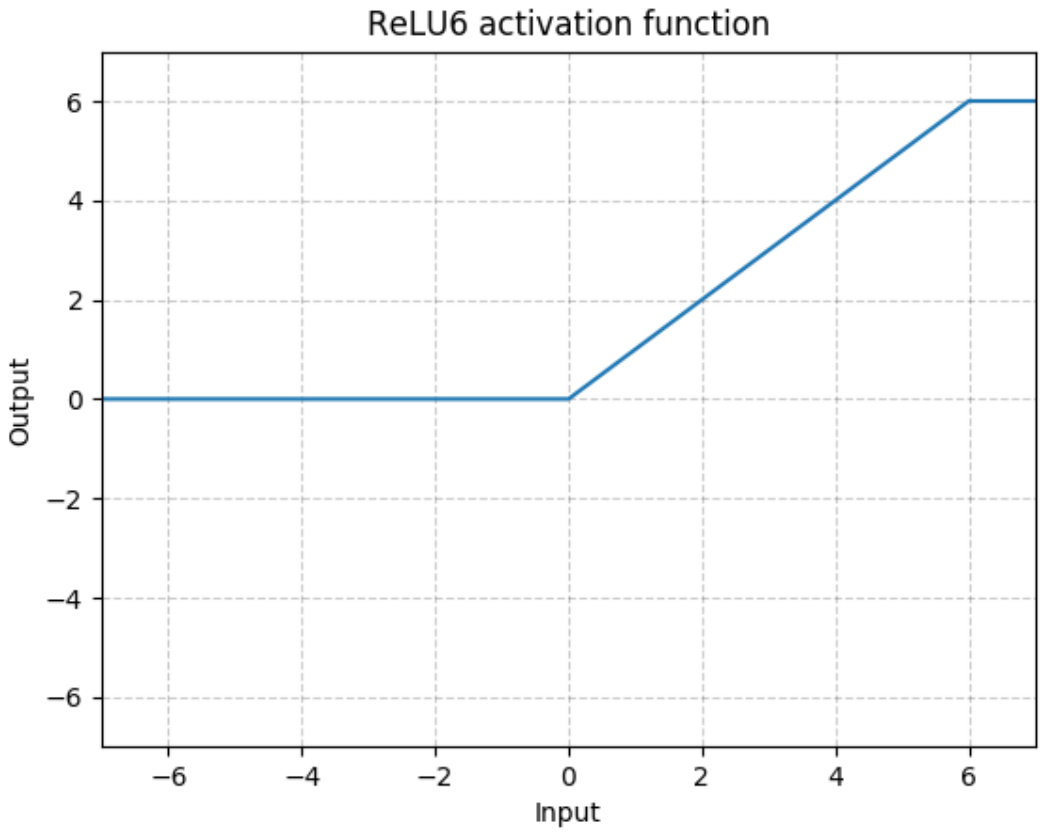

Figure 55 ReLU6

- ReLU6 is used due to its robustness when used with low-precision computation, based on [27] MobileNetV1.

A3.2. MobileNetV2

- In MobileNetV2, there are two types of blocks. One is residual block with stride of 1. Another one is block with stride of 2 for downsizing.
- There are 3 layers for both types of blocks.

- This time, the first layer is 1×1 convolution with ReLU6.
- The second layer is the depthwise convolution.
- The third layer is another 1×1 convolution but without any non-linearity. It is claimed that if ReLU is used again, the deep networks only have the power of a linear classifier on the non-zero volume part of the output domain.

Input	Operator	Output
$h \times w \times k$	1x1 conv2d , ReLU6	$h \times w \times (tk)$
$h \times w \times tk$	3x3 dwise s=s, ReLU6	$\frac{h}{s} \times \frac{w}{s} \times (tk)$
$\frac{h}{s} \times \frac{w}{s} \times tk$	linear 1x1 conv2d	$\frac{h}{s} \times \frac{w}{s} \times k'$

Figure 56 Input and Output Details of MobileNetV2

- And there is an expansion factor t. And t=6 for all main experiments.
- If the input got 64 channels, the internal output would get 64×t=64×6=384 channels.
- Overall Architecture

A3.2.2 MobileNetV2 Overall Architecture

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Figure 57 MobileNetV2 Overall Architecture

- where t: expansion factor, c: number of output channels, n: repeating number, s: stride. 3×3 kernels are used for spatial convolution.
- In typical, the primary network (width multiplier 1, 224×224), has a computational cost of 300 million multiply-adds and uses 3.4 million parameters. (Width multiplier is introduced in MobileNetV1.)
- The performance trade offs are further explored, for input resolutions from 96 to 224, and width multipliers of 0.35 to 1.4.
- The network computational cost up to 585M MAdds, while the model size vary between 1.7M and 6.9M parameters.
- To train the network, 16 GPU is used with batch size of 96.

CONCLUSIONS AND RECOMMENDATIONS

In conclusion, an IoT-aided irrigation system can bring numerous benefits by leveraging technology to optimize water usage, improve crop yield, and reduce manual effort. By integrating sensors, microcontrollers, and cloud platforms, you can create a smart irrigation system that adapts irrigation schedules based on real-time data and environmental conditions.

Here are some recommendations for implementing an IoT-aided irrigation system:

Sensor Selection: Choose appropriate sensors for monitoring soil moisture, temperature, humidity, rainfall, and other relevant parameters. Ensure they are compatible with your microcontroller and provide accurate and reliable data.

Microcontroller and Communication: Select a suitable microcontroller platform (such as Arduino or ESP32) with adequate processing power and connectivity options. Consider the communication protocols required to connect your system to the internet or other devices, such as Wi-Fi, Bluetooth, or MQTT.

Cloud Platform Integration: Utilize cloud platforms like AWS IoT, Google Cloud IoT, or Microsoft Azure IoT to store and process sensor data. These platforms offer scalability, data analytics, remote device management, and integration with other services.

Decision-Making Algorithm: Develop intelligent algorithms that analyze sensor data, weather forecasts, and crop requirements to make informed decisions regarding irrigation schedules and water flow control. Consider using machine learning techniques to improve accuracy and optimize water usage.

Mobile/Web Interface: Create a user-friendly interface, accessible through mobile devices or web browsers, for monitoring and controlling the irrigation system remotely. Provide real-time status updates, notifications, and the ability to adjust settings.

Security Measures: Implement robust security measures to protect the system from unauthorized access or data breaches. Use encryption, authentication mechanisms, and regular software updates to ensure the system's integrity.

Power Management: Optimize power consumption by utilizing low-power sensors, sleep modes, or solar-powered solutions where possible. This helps prolong battery life and reduces maintenance efforts.

Scalability and Flexibility: Design the system with scalability in mind, allowing for easy addition or removal of sensors and devices. Consider future expansion or integration with other IoT systems or agricultural management tools.

Regular Maintenance and Calibration: Regularly check and calibrate the sensors to maintain accuracy and reliability. Monitor the system's performance, conduct periodic checks, and address any issues promptly.

Regulatory Compliance: Familiarize yourself with local regulations and guidelines related to water usage and irrigation practices. Ensure your system adheres to these requirements and follows best practices for sustainable irrigation.

By considering these recommendations and tailoring them to your specific needs and environment, you can develop a robust and efficient IoT-aided irrigation system that optimizes water usage, promotes sustainable agriculture, and improves crop yield.

DIRECTION FOR FUTURE WORK

PART I

As technology continues to advance, there are several directions for future use and enhancement of IoT-aided irrigation systems. Here are some potential areas of focus for further development: Advanced Sensor Technologies: Explore the use of advanced sensor technologies, such as hyperspectral imaging or drones equipped with multispectral cameras, to gather more detailed and precise data about soil conditions, plant health, and water needs. These technologies can provide deeper insights into crop health and enable targeted irrigation strategies.

Predictive Analytics and Machine Learning: Develop sophisticated predictive analytics models and machine learning algorithms that leverage historical data, weather patterns, and crop characteristics to predict irrigation requirements accurately. This can lead to proactive and automated irrigation scheduling, improving water efficiency and crop yield.

Integration with Weather Forecasting: Integrate weather forecasting data into the irrigation system to enhance decision-making. By considering future weather conditions, the system can adjust irrigation schedules accordingly, optimizing water usage and adapting to changing environmental factors.

Smart Irrigation Zones: Implement zoning capabilities within the irrigation system, allowing for different irrigation schedules and water requirements based on specific crop types, soil variations, or microclimates within the agricultural area. This level of granularity can optimize water distribution and ensure each zone receives the appropriate amount of irrigation.

Wireless Sensor Networks: Utilize wireless sensor networks to expand coverage and monitor larger agricultural areas. These networks can facilitate seamless data collection from multiple sensors, reducing wiring complexity and enabling cost-effective scalability.

Integration with Crop Management Systems: Integrate IoT-aided irrigation systems with broader crop management platforms or farm management software. This integration can

provide a holistic view of agricultural operations, enabling better coordination between irrigation, fertilization, pest control, and other practices.

Water Quality Monitoring: Extend the functionality of the irrigation system to monitor water quality parameters, such as pH levels or nutrient concentrations. This information can help optimize irrigation strategies and ensure proper nutrient delivery to the plants.

Remote Monitoring and Control: Enhance remote monitoring and control capabilities by leveraging mobile applications or web-based interfaces. This allows farmers or agronomists to monitor the irrigation system, receive real-time alerts, and make adjustments from anywhere, improving convenience and efficiency.

Sustainability and Water Conservation: Focus on sustainability and water conservation by integrating water recycling and rainwater harvesting systems. Implement water usage analytics and reporting features to track and optimize water consumption over time.

Integration with Agricultural IoT Ecosystem: Collaborate and integrate with other IoT solutions and agricultural ecosystems to enable synergistic benefits. For example, integration with smart pest management systems or automated greenhouse controls can create a comprehensive IoT-driven agricultural solution.

These future directions for IoT-aided irrigation systems aim to further optimize water usage, improve crop productivity, reduce manual intervention, and promote sustainable agricultural practices. Continual research, development, and collaboration within the agricultural technology space will contribute to the advancement and adoption of these innovations.

PART II

There are several potential future directions for your machine learning and OpenAI-powered application that can help improve its performance and value. Here are some possible future work options:

Adding more diseases: You can consider adding more diseases to the application, which can help increase its utility and reach more users. This can be done by improving the machine

learning model's accuracy on existing diseases and adding new models for additional disease classes.

Improving the user interface: While the current Streamlit-based user interface is effective, there is always room for improvement. You can consider updating the interface design to make it more visually appealing, more intuitive, and easier to use.

Increasing the range of assistance: Currently, your application provides users with treatment recommendations based on identified diseases. You can expand on this by including additional resources within the OpenAI language model, such as natural language understanding (NLU) capabilities, that can help users understand symptoms, diagnostic procedures, and other important medical information.

Adding user feedback loops: Including features where users can provide feedback on treatment effectiveness and symptoms, which can help improve the accuracy of the machine learning model over time.

Integration with additional healthcare services: Integrating with other healthcare services such as hospitals, clinics, or pharmacies, can add more value to your application. This can include information on prescription fulfilment, scheduling appointments with healthcare practitioners, and remote consultations with doctors.

By pursuing these future work options, you can further enhance the value and usefulness of your machine learning and OpenAI-powered application, improving its overall efficacy and potential impact on users in need.

BIBLIOGRAPHY

Here is a bibliography of resources that you can refer to for further information on

PART I

- Al-Saedi, M., Al-Anbuky, A., & Cao, J. (2019). Smart Irrigation System Based on Internet of Things (IoT). In 2019 16th International Bhurban Conference on Applied Sciences and Technology (IBCAST) (pp. 565-570). IEEE. DOI: 10.1109/IBCAST.2019.8667074.

- Gupta, N., & Patil, S. P. (2020). IoT Based Smart Irrigation System Using Wireless Sensor Network. In 2020 7th International Conference on Signal Processing and Integrated Networks (SPIN) (pp. 639-643). IEEE. DOI: 10.1109/SPIN48245.2020.9073899.

- Ballesteros, D., Zanetti, M., & Fleury, A. (2019). Smart Irrigation System for Sustainable Agriculture. In 2019 IEEE Latin American Conference on Computational Intelligence (pp. 1-6). IEEE. DOI: 10.1109/LA-CCI47712.2019.9027715.

- Suresh, M. R., & Kumar, P. S. (2021). IoT Based Smart Irrigation System for Sustainable Agriculture. In 2021 5th International Conference on Intelligent Sustainable Systems (ICISS) (pp. 125-129). IEEE. DOI: 10.1109/ICISS51446.2021.9353674.

- Banani, S., & Mohammadzadeh, H. (2019). IoT-based smart irrigation system using fuzzy logic. In 2019 4th International Conference on Information and Communication Technologies for Disaster Management (ICT-DM) (pp. 1-4). IEEE. DOI: 10.1109/ICT-DM47956.2019.9032900.

- Al-Jarrah, O. Y., Abo-Zahhad, M., & Alsa'deh, B. (2018). Smart irrigation system based on IoT and machine learning. In 2018 10th International Conference on Computer and Automation Engineering (ICCAE) (pp. 121-125). IEEE. DOI: 10.1109/ICCAE.2018.8399770.

- Choudhary, S., & Srivastava, V. (2018). An IoT-Based Smart Irrigation System Using Machine Learning. In Proceedings of the International Conference on Smart Systems and Inventive Technology (ICSSIT) (pp. 1275-1279). IEEE. DOI: 10.1109/ICSSIT.2018.8569863.

- Mokhlis, H., Wibowo, A., & Rahman, H. A. (2020). A Review of IoT-Based Smart Irrigation Systems for Precision Agriculture. IEEE Access, 8, 225590-225608. DOI: 10.1109/ACCESS.2020.3046547.

- Mishra, P., Sharma, A., & Kumar, A. (2021). A Comprehensive Review of Internet of Things (IoT)-Based Smart Irrigation Systems. In 2021 3rd International Conference on Advances in Electrical, Computer and Communication Technologies (ICECCT) (pp. 1-6). IEEE. DOI: 10.1109/ICECCT51925.2021.9388547.

- Singh, A., & Gahoi, A. (2020). IoT

PART II

Here are some references for Machine Learning-based plant diagnosis chatbot systems:

- Dhawan, G., Kumar, P., & Sharma, V. (2020). Plant Disease Identification Using Machine Learning. In 2020 International Conference on Computing, Electronics & Communications Engineering (iCCECE) (pp. 1-5). IEEE. DOI: 10.1109/iCCECE48679.2020.9102935.
- Singh, G., Yadav, M., & Soni, A. (2021). Deep Learning Based Plant Disease Recognition and Suggestion of Remedial Measures. In 2021 IEEE Uttar Pradesh Section International Conference on Electrical, Electronics and Computer Engineering (UPCON) (pp. 1-5). IEEE. DOI:10.1109/UPCON51204.2021.9608420.
- Han, Y., Zheng, Y., Wei, Z., Chen, Y., & Jiang, Y. (2020). An Intelligent Plant Disease Identification System Based on Deep Learning. In 2020 6th International Conference on Big Data Computing and Communications (BIGCOM) (pp. 185-189). IEEE. DOI: 10.1109/BIGCOM50203.2020.00044.
- Kumar, V., Rathi, N., & Agarwal, S. (2020). An Intelligent System for Plant Disease Diagnosis using Deep Learning Techniques. In 2020 2nd International Conference

on Smart Electronics and Communication (ICOSEC) (pp. 1-5). IEEE. DOI: 10.1109/ICOSEC49240.2020.9239966.

- Patil, S., & Kulkarni, V. (2020). Intelligent Plant Disease Diagnosis Using Deep Learning. In 2020 4th International Conference on Electronics, Communication and Aerospace Technology (ICECA) (pp. 438-442). IEEE. DOI: 10.1109/ICECA49662.2020.9251095.
- Kadu, P., Sonawane, R., & Gandhi, R. (2020). CNN Based Smart Plant Disease Detection Using Transfer Learning. In 2020 International Conference on Smart Electronics and Communication (ICOSEC) (pp. 1-4). IEEE. DOI: 10.1109/ICOSEC50810.2020.9275421.
- Ali, S. R., Khalid, A., & Atif, M. (2018). LeafDoctor: A Chatbot for Plant Disease Diagnosis Using Machine Learning Techniques. In 2018 International Conference on Frontiers of Information Technology (FIT) (pp. 132-137). IEEE. DOI: 10.1109/FIT.2018.00033.
- Zou, J., & Zhang, W. (2020). Chatbot-assisted Plant Disease Identification based on Deep Learning. In 2020 5th International Conference on Computer Science and Artificial Intelligence (CSAI) (pp. 1-4). IEEE. DOI: 10.1109/CSAI49849.2020.9251646.
- Anjum, N., Habib, S. M., & Saleem, S. (2019). Chatbot for Plant Disease Diagnosis using Machine Learning. In 2019 4th International Conference on Emerging Trends in Engineering, Sciences and Technology (ICEEST) (pp. 1-6). IEEE. DOI: 10.1109/ICEEST48016.2019.9097360.