GEOVERDE

AI-Driven Farming for a Sustainable Future

Team : HBS Visionaries

- Laasyasree Bokkasam
- Helen Vien
- Sara Hazari
- Ruilan Ma
- Isabella Krovvidi

Table of Contents

1. Executive Summary

The global demand for food is rising, putting pressure on farmers to increase crop production. However, traditional farming often relies on outdated methods, leading to poor soil care and late disease detection. The farmers indeed face two key challenges: inconsistent soil data for irrigation and fertilisation, and late identification of plant diseases. To address these issues, we have developed GeoVerde, an Artificial Intelligence (AI)-driven solution that integrates Internet of Things (IoT) sensors for real-time soil monitoring and Unmanned Aerial Vehicle (UAV) technology for early plant disease detection. By leveraging emerging technologies, such as IoT, UAVs and AI, GeoVerde offers innovative solutions to improve resource management in agriculture for a sustainable future.

This project focuses on developing two prototypes: one that utilises IoT sensors for soil monitoring and another that employs an AI machine learning model for disease detection through high-resolution images captured by UAVs. Our key objectives include designing reliable sensors for effective monitoring, implementing plant disease detection and ensuring seamless data accessibility. By providing farmers with advanced decision-making tools, the GeoVerde helps reduce waste and improve crop yields, which ultimately contributes to global food security.

Furthermore, the GeoVerde is a comprehensive soil monitoring system, which will be further developed to provide farmers with real-time soil data, aiding irrigation and fertilisation techniques, while promoting sustainable farming practices. This enables the farmers to adopt more efficient techniques for producing better crop yields at a lower cost and offering significant environmental benefits, such as water conservation and reduced chemical usage including pesticides.

2. Introduction

2.1. Background and Motivation

The rapid increase in global food demand, along with climate change and soil degradation, has put immense pressure on agricultural systems to produce bigger yields while ensuring their practices are not harmful to the environment. Emerging technologies, such as the Internet of Things (IoT), Unmanned Aerial Vehicles (UAVs) (Javaid, M. *et al.*, 2024, Pamuklu, T. *et al.,* 2023) and Artificial Intelligence (AI), have emerged as a solution to optimise resources, enhance sustainability and improve agricultural techniques. However, conventional farming practices respond to problems after they arise, rather than identifying certain risks before they manifester; This can lead to inefficient soil management and delayed disease detection, which hugely impacts crop productivity and resource efficiency.

The traditional methods of soil monitoring and health assessment of crops are laborious, time-consuming and more likely prone to inaccuracies.

Here are the two major challenges that farmers face:

1. How to prevent soil degradation and optimise resource efficiency, since irrigation and fertiliser usage remain inconsistent?
2. How to detect plant diseases at an early stage before visible symptoms appear, preventing the loss of crops and reducing excessive pesticide use?

The problem is caused by a lack of real-time soil data. Furthermore, in traditional farming techniques not all the factors are taken into consideration.

To address these common challenges, we have introduced an AI-driven soil monitoring system which uses IoT sensors (Li, T. *et al.*, 2024) to detect factors such as temperature, humidity levels, soil moisture and many more; moreover, we use UAV drones (Rejeb, A. *et al*, 2022) that detect plant diseases, using high resolution cameras. This system helps farmers to make decisions based on data and also helps to promote sustainable farming practices and enhances agricultural techniques.

2.2. Aims and Objectives

1. Aim

The aim of this project is to develop a system that uses IoT sensors for soil monitoring and AI to detect plant diseases using images captured from a phone camera. This prototype can later be adapted to detect plant diseases by using a UAV drone to capture images using a high resolution camera.

2. Objectives

- *Objective 1*: Build an IoT based prototype containing a temperature sensor which can be expanded to include soil moisture, pH, and nutrient level sensors among many others in the future.
- *Objective 2*: Create an AI powered system for early disease detection of plants that can be integrated with drones equipped with high-resolution cameras for more seamless image capturing.
- *Objective 3*: Assess how well the prototype works with its targeted features of temperature measurement and plant disease detection to see if it can be developed further for use in modern farming technologies.

2.3. Significance of the Project

This project helps introduce better farming methods by being more resource efficient, improving disease management and helping farmers in making better decisions based on the data provided by GeoVerde. With GeoVerde, farmers in developing countries can optimise their resource usage, significantly reducing expenses on fertilisers, pesticides, and water, thereby improving both economic and environmental sustainability. By using IoT and UAV technologies with AI, GeoVerde aims to promote sustainable farming practices, ultimately boosting agricultural productivity and supporting global food security.

3. Methodology

Data collection and analysis

The GeoVerde project applies an integrated approach to soil health monitoring by combining sensor networks, bio-interactive technology and advanced data analytics.

Our prototypes collect data through using an array of clay-ambed sensors designed to measure major parameters such as moisture levels, nutrient content, pH and microbial activity.They provide real-time data to cloud platforms on a continuous basis, which is gathered and processed for analysis.

In addition to ground sensor data, drones (Shenoy, H., & Chatterjee, S., 2024) equipped with high-resolution cameras are dispatched to capture aerial photographs of crops. The aerial photographs are analysed to assess the health of the plant by using plant health and disease detection algorithms, photogrammetry and multispectral imaging techniques and explore early signs of the disease.

Data analysis is utilised by using machine learning algorithms and statistical models to identify trends, predict soil health results and generate actionable insights.

Process data is imagined through an interactive dashboard for real-time data that displays users the state of the soil and allows them to modify land management approaches based on the real-time data.

This is our proposed methodology for the project. While we have implemented key components, our approach can be adapted or expanded for various real-world applications. For instance, a real drone equipped with a high-resolution camera could be deployed to capture plant leaf images instead of a phone, as in our prototype. Also, we might have advanced sensors for better monitoring and analysing the soil.

Hardware, technology, and infrastructure Sensor Technology:

Internet of Things-enabled soil sensors provide precise, real-time measurements of critical soil parameters such as moisture, temperature, pH, and nutrient status to enable real-time monitoring and analysis.

Cloud Computing:

A cloud computing platform processes, stores, and analyses the data in an efficient manner, offering remote access and scalability for wider use.

Machine Learning algorithms:

Complex predictive modeling and recovery analysis techniques such as neural networks are utilised to identify trends in soil condition and forecast future soil conditions.

Landstone analysis:

Geographic information system mapping technology is applied to map the soil condition and facilitate efficient and targeted soil management practice

4. GeoVerde Development

4.1. Concept

The GeoVerde project introduces an innovative approach to accurate agriculture by integrating IoT-enabled soil sensors, drone imaging and AI-operated data analysis (Zhang, S. *et al,* 2024) to optimise soil health and crop productivity. The unique thing about this innovation lies in soil conditions and the ability to provide real -time data with high resolution on plant health, so that farmers can make data -related decisions that improve stability and efficiency. Unlike traditional soil tests, which are often labor-intensive and unnatural, our system provides continuous monitoring with future insights, which allows active intervention.

Diagram of Installed GeoVerde

The sketch in Figure 4.1 displays two scenarios in which GeoVerde would be installed in a garden or a green space. Scenario one is when the physical box Containing the hardware is just about above ground, with the sensors Monitoring the soil from underground. By having the box on the ground, GeoVerde becomes easier to manually install.

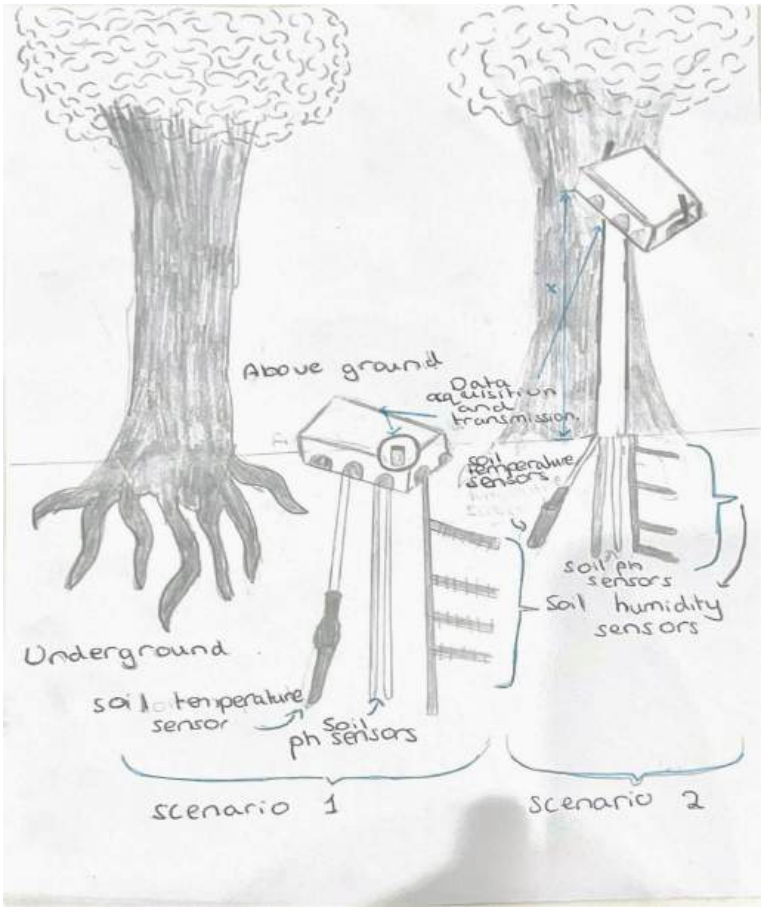


Figure 4.1: GeoVerde Installation Scenario in a Green Space

Figure 4.2 below illustrates the CAD model of the drone we are developing, equipped with a high-resolution camera to capture detailed images of the plant leaves.

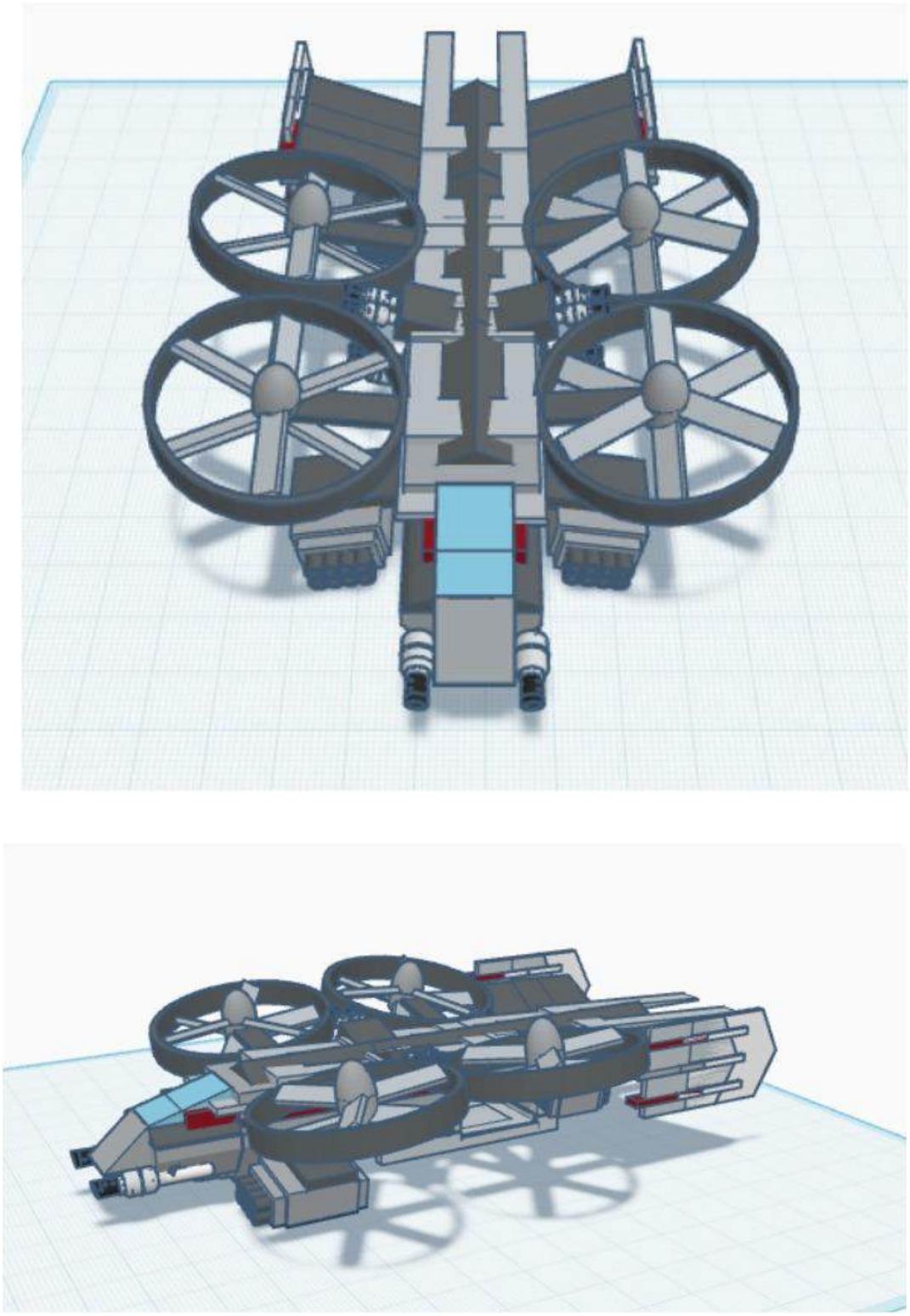


Figure 4.2: CAD model of a drone

4.2. Inspiration

The development of GeoVerde was inspired by the growing need for permanent agriculture in the midst of climate change, soil falls and increasing global demand for food. Farmers and agricultural researchers have faced challenges in providing health data for long -term and accurate soil. Existing methods often fail to give real -time insight, which causes disabled resource allocation. As we observe these intervals, we have purposefully created a solution that benefits from the condition-of-art technology to improve soil management practices, increase the crop yield and reduce the environmental impact

4.3. Research

Our innovation produces and improves many existing technologies, including:

IoT-based soil surveillance: Use smart sensors to continuously track soil parameters.

Drone imaging for accurate agriculture: to detect plant diseases and to assess crop health, to use high -resolution and multispectral cameras.

Machine learning application in agriculture: Development of a future model based on historical and real-time data to adapt soil and plant health management.

Cloud Computing and GIS mapping: Storage and analysis Vishal

4.4. Impacts & Justification

The GeoVerde project deals with an important issue in modern agriculture (Think with Niche, 2024): skilled, lack of real -time monitoring of soil health and plant disease.

Farmers face several challenges in dealing with soil conditions and preventing crop diseases, reducing the dividend, increasing costs and preventing environmental discharge. Current solutions are often inadequate and depend on manual inspection or wrong sensors.

GeoVerde aims to bring revolution into soil health control by integrating IoT-based soil data collection and AI-operated plant disease diagnosis, which can enable farmers to make date-driven decisions for adapting productivity and stability. Preliminary research suggests that the crop yield can increase by 30%, which reduces the requirement for chemical treatment by imbalance of the soil's nutrients and early detection of plants. In

addition, permanent soil management practices can improve the soil's health and reduce environmental effects.

By implementing this innovative solution, GeoVerde can significantly improve agricultural efficiency, reduce operating costs and contribute to more durable agricultural practices. The potential market for such technology is spacious, as it can be used globally in small scale and large farms, providing an average environment and economic benefits.

4.5. Creative process

The GeoVerde project began with recognition of increasing challenges, which farmers face to maintain healthy soil and prevent plant diseases. Through extensive research, the team identified high pain points, such as monitoring from time to time and the difficulty of monitoring the limited accuracy of traditional disease identification methods.

1. Concept Development

GeoVerde was envisioned as a revolutionary IoT and AI system that could aid modern farming by conducting real-time soil analysis and timely plant disease diagnosis. The first line of focus was on temperature monitoring combined with AI-powered disease diagnosis through phone camera images with the future hope of employing drone captured images for extensive monitoring.

2. Research & Planning

The team conducted broad research on IoT systems for soil moisture monitoring, AI plant disease diagnosis, and UAV imaging. Main objectives included:

- Determining the best temperature sensors that could later allow clip-on sensors for other soil health indicators.
- Determining the capabilities of existing AI models that could recognise diseases of plants from photographs taken by mobile phones.
- Assessing the feasibility of the incorporation of drones for primary scalability purposes.
- Finding AI models with the capability of identifying plants’ diseases through images taken with cell phones.
- Determining the possibility of integrating drones for future scalability.

3. Prototype Development

The first prototype was planned with a temperature sensor module which was able to gather and transmit data for processing in near real-time. This design was created with expansion in mind for future implementation of moisture, pH, and nutrient sensors. At the same time, an AI model was created and trained with planted disease images focused on early diagnosis through images taken with a phone camera.

4. Testing and Refinement

The prototype was tested in the field for accuracy and functionality. Important changes included:

- Calibrating the sensors for accurate temperature measurement.
- Improving artificial intelligence detection by increasing the accuracy of image processing.
- Streamlining data transmission capabilities and augmenting storage for real-time monitoring.

5. Performance Evaluation

The team reviewed the accuracy in measuring temperature and identifying plant diseases alongside the controlled testing cycles and sought to achieve what was required from the prototype. These findings were essential in framing further refinement which expected greater integration from the sensors and higher accuracy from the artificial intelligence.

6. Future Development and Integration

The wider perspective comprises:

- Creation of more sensors for the determination of soil moisture, pH, and nutrient content.
- Full automation of disease detection using drones with high-definition cameras.
- Optimisation and enhancement of the system's usability for practical farming operations is needed to improve accessibility to the system by farmers.

5. GeoVerde: Approach & Implementation

5.1. Design Process

GeoVerde adopts a methodical approach that commences with the distribution of soil sensor nodes across a field. These sensors monitor the level of moisture, temperature, pH, nutrients, and transmit them wirelessly to a central hub. The hub stores this information locally or on the cloud after processing it. Different AI available parameters are used to analyse this information to form recommendations on how to improve the soil health such as changing irrigation and fertiliser application methods. For farmers to make informed decisions quickly, the data is simplified and presented in an easy to understand dashboard. The system also enables sustainable farming practices by providing round the clock monitoring to ensure timely interventions and alterations are made optimally.

5.2. Technological Framework

The technological framework of our GeoVerde is illustrated in Figure 5.1 below, which consists the following blocks:

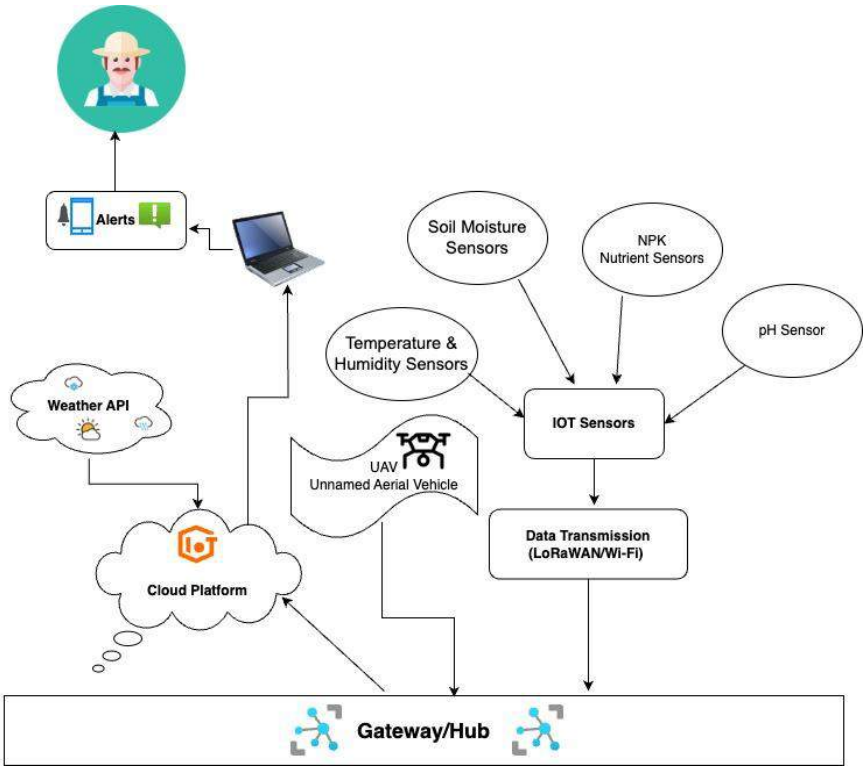


Figure 5.1. Technological Framework of GeoVerde

1. **IoT Sensors**

These sensors are placed strategically across the field to constantly assess essential parameters such as moisture, temperature, pH, and nutrients. Collected data is later transmitted to the central hub for processing.

2. **Central Hub (Data Processing Unit)**

This unit acts as a central location for receiving and recording raw sensor data and processing them locally or in the cloud. The received data is processed for verification and checked for anomalies. IP filtering and segmentation are done so the data is ready for AI analysis.

3. **AI Analysis & Recommendation System**

AI analyses the provided data and identifies trends of available soil and recommends the usage of proper irrigation techniques, fertiliser application, and other practices for achieving desired results.

4. **Cloud Storage & Data Access**

Helps farmers and agricultural professionals greatly by providing instant data access.

5. **Dashboard For Farmers (User Interface)**

Changes complex principles of AI into a visual-easy representation which can be interpreted without much of a hassle and presents information on soil health trends, recommended actions, and alerts enabling the required attention at the right time. APIs are used as an intermediate between frontend and backend to transmit the data to the user

6. **Sustainable Farming Feedback Loop**

Helps with every farming task at all levels, providing real-time data to adjust and improve agricultural methods and also promotes practices that are more sustainable by lessening the wastage of resources.

5.3. Challenges

GeoVerde encounters problems like poor sensor data, battery and connectivity in remote locations. These problems can be fixed by calibrating the sensors, making an energy efficient design, and having reliable communication using LoRaWAN technology or Wifi to store the data in the Cloud. Initially, we struggled with creating the AI- based plant disease detection as it was inaccurate . However, we fixed this by adding more data and treating missing and outlier values. Moreover, farmers may not be comfortable with this idea as they are not accustomed to it. However, by providing a simple interface and training they can slowly get used to this format. Budget limitations may also be problems, but this can be solved with a modular approach and potential design partners for support.

Technology

Hardware Components:

Soil Moisture Sensors: Maxims WUE by preventing over- or under-watering.

Nutrient Sensors: Increase NUE by monitoring nitrogen, phosphorus, and potassium (NPK) levels for proper fertilisation.

Temperature & Humidity Sensors: Monitor environmental parameters affecting soil health.

LoRa Modules: Support low-power, long-range wireless data transmission.

Software & Data Analytics:

Machine Learning Models: Predict soil trends and recommend sustainable irrigation and fertilisation.

Cloud-Based Dashboard: Provides farmers actionable insights, real-time alerts, and sustainability reports.

Automated UAV Monitoring: Use drones for aerial photogrammetry and multispectral analysis to optimise pesticide use efficiency (PUE).

Water Conservation: WUE is enhanced by smart irrigation methods to overcome water scarcity concerns.

Soil Regeneration: Controlled fertilisation maintains soil profile and prevents degradation.

Lower Carbon Footprint: Reduced application of chemical inputs minimises carbon levels and enhances sustainability.

Less Pollution: Reduced chemical runoff avoids contaminating surrounding water bodies and ecosystems.

There are small, low-power devices embedded in the soil, called IoT sensors, that measure moisture, temperature, pH, and nutrient levels. The Central Hub, also known as a Gateway Device collects data from multiple sensor nodes and utilises wireless communication to transmit data. It processes and transmits data to the cloud or a local database. This can also be solar-powered for remote applications. The data collected can be seen using a web-based or mobile application. This application can display real-time soil health metrics, trends, and alerts and the user receives AI-driven insights for optimising soil conditions.

5.4. Prototypes

In this project, we develop two prototypes: **1) Plant Disease Detection (PDD)** and **2) Temperature Monitoring (TM)**. The PDD prototype utilises ML to identify plant diseases based on images captured by a phone camera. The TM prototype employs a temperature sensor integrated with an Arduino board to monitor environmental temperature, triggering alerts when it falls within a specific range or exceeds a predefined threshold.

5.4.1. Plant Disease Detection (PDD) Prototype

For the PDD prototype, we first develop a training model to classify plant images into three categories: healthy, rusted, and powdery. The training image dataset can be downloaded on https://www.kaggle.com/datasets/rashikrahmanpritom/plant-disease-recognition-dataset

The Python code for model training, written in Jupyter Notebook, is provided in Appendix A.

Once trained, the model is loaded into a Python script running in Visual Studio (VS) Code (see Appendix B). It is deployed as a Flask app for demonstrating the PDD prototype, as illustrated in Figure 5.2 below.

Figure 5.2. Plant Disease Detection Flask App

The step-by-step instructions for running the PDD prototype are as follows:

Step 1: Download all files into the same working directory.

Step 2: Download the training image dataset on Kaggle.

Step 3: Launch Jupyter Notebook from Anaconda.

Step 4: Open file "PlantDiseaseModelTraining_V2.ipynb" (see Appendix A)

Step 5: Run all cells in this notebook. The trained model will be saved as "plant disease.keras" in the same directory.

Step 6: Launch Visual Studio Code.

Step 7: Open file "camera.py" (see Appendix B)

Step 8: Run the script until the trained model is successfully loaded.

Step 9: Launch the Flask app on local server at http://127.0.0.1:5000/

Step 10: Select an image from the "Test" folder for prediction.

5.4.2. Temperature Monitoring (TM) Prototype

For the TM prototype, we first design a circuit using Tinkercad (https://www.tinkercad.com/), as shown in Figure 5.3 below. The required hardware components include:

- Arduino Uno board (x1)
- Breadboard with jumper wires (x1)
- LM35 temperature sensor (x1)
- Buzzer (x1)
- LED (x1)
- 220Ω resistor (x1)

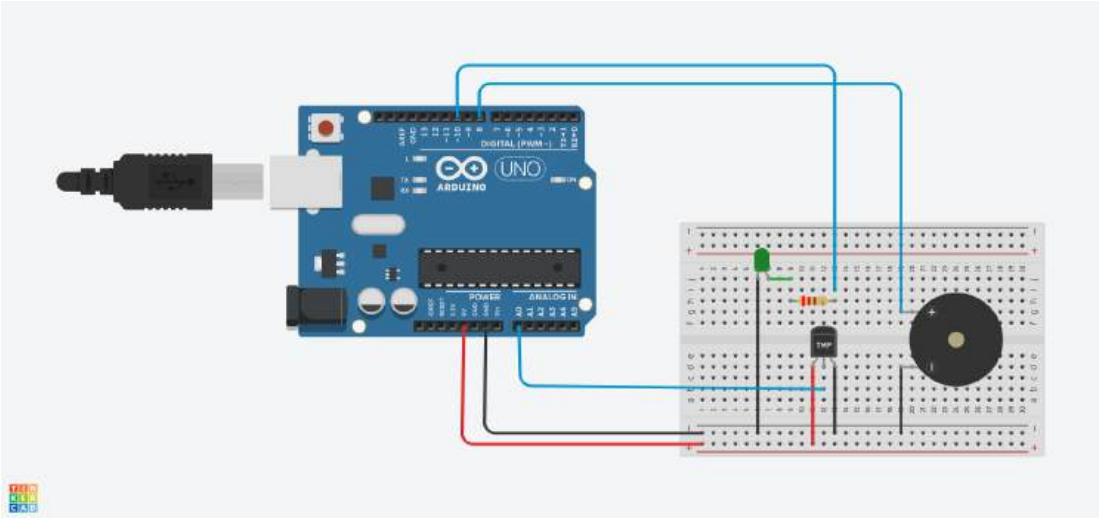


Figure 5.3. Tinkercad Design for Temperature Monitoring Prototype

This circuit is then assembled with physical components on the Arduino board as in Figure 5.4. The Python script to run our TM prototype is written in VS Code (see Appendix C).

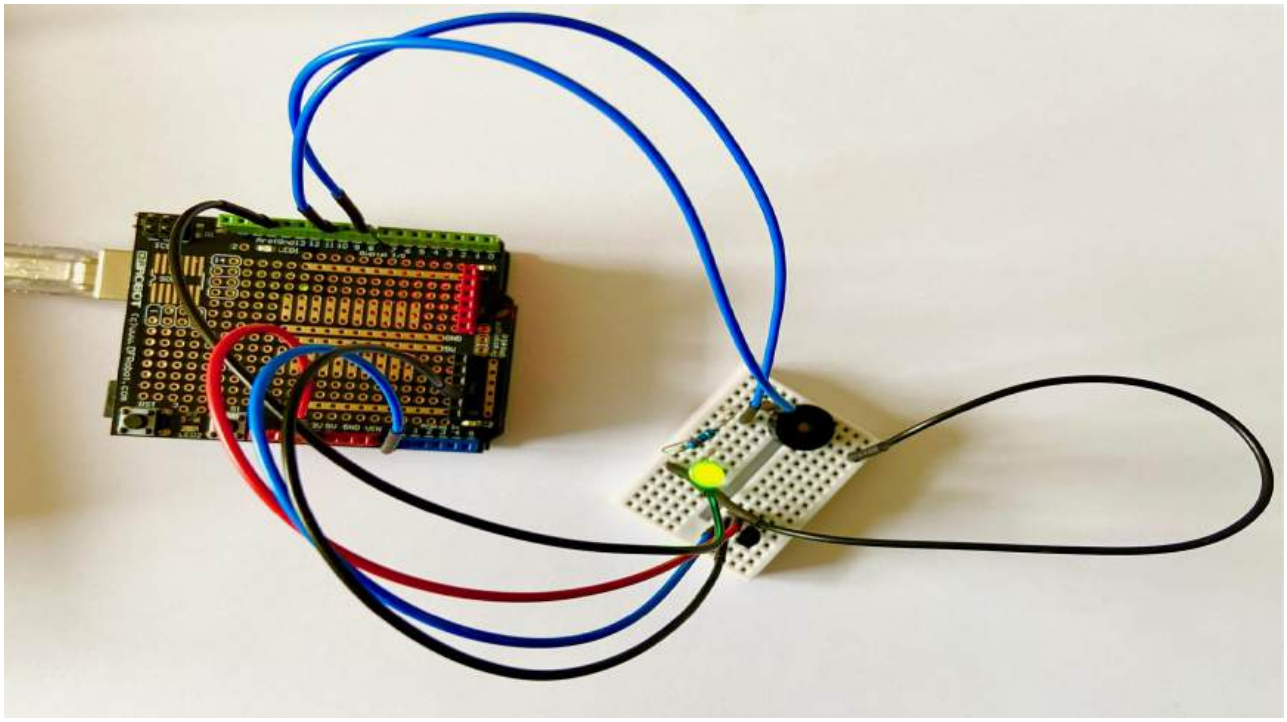


Figure 5.4. Hardware Circuit Design of Temperature Monitoring Prototype

Below are step-by-step instructions for running the TM prototype:

- Step 1: Set up the hardware as shown in Figure 5.3 above.
- Step 2: Launch Arduino IDE.
- Step 3: Open the StandardFirmata sketch by navigating to: File -> Examples -> Firmata -> StandardFirmata
- Step 4: Verify, compile and upload the code to the Arduino board.
- Step 5: Launch Visual Studio Code.
- Step 6: Open file "temperaturesensor.py" (see Appendix C)
- Step 7: Adjust the PORT parameter according to your system configuration.
- Step 8: Set different values of temperature threshold, minimum and maximum temperature.
- Step 9: Save and run the script.

6. Impact & Sustainability Assessment

Arduino-based soil surveillance systems with high-resolution cameras equipped on UAV to detect the plant disease can provide significant benefits in three key dimensions: environmental, economic and social stability. In this section, the impacts of our GeoVerde project are firstly analysed, followed by a discussion of possible challenges and mitigation techniques. Finally, sustainability metrics are outlined to validate the effectiveness of our proposed approach.

6.1. Impact Analysis

1. Environmental Impact

Custom use: Continuous monitoring of soil condition enables accurate cultivation, reduces excessive use of water, fertilisers and pesticides.

Early disease detection: UAV helps detect plant disorders, prevents broad outbreaks and reduces the need for chemical treatment.

Protection of biodiversity: By reducing pesticides, the project supports favorable pests and microorganisms, and promotes biodiversity in agricultural ecosystems.

Reduction in carbon footprints: More efficient use of agricultural entrances reduces energy consumption and reduces unnecessary chemical applications and low emissions.

2. Economic impact

Cost savings: The prosecutors help reduce the costs of agricultural technology, Arduino-based soil sensors, water, fertilsers and pesticides.

Increase in crop dividend: Detection of initial disease improves crop health, leading to better returns and high economic returns.

Cheap technology: Arduino-based systems are affordable, which also makes them available to small farmers with limited resources.

Data-driven decisions: The data collected from these technologies enables farmers to make informed decisions, optimise operations and reduce crop losses.

3. Social influence

Empire farmers: Land and plant health data in real time gives small holders the right to make better decisions and increase productivity.

Education and skills development: Adoption of Agritech and IoT solutions increase digital literacy, especially in rural areas.

Food security: Healthy crops and high dividends contribute to more stable food supply, reduce waste and ensure food security.

Sustainable agricultural practices: By encouraging environmentally friendly agricultural techniques, the project helps to reduce the earth's decline in the long term.

6.2. Possible Challenges & Mitigation Techniques

Technology adoption obstacles:

Some farmers may additionally struggle to use new techniques.

Solution: Provide education to manual farmers through the adoption system and set up local help networks.

Data Management and Privacy challenge:

There can be problems about security and privacy for agricultural statistics.

Solution: Use open assets, peasant -managed systems to make certain secure and obvious data management.

Original installation cost:

Although technology is cheap, some farmers may face challenges with advance investment.

Solution: To reduce financial obstacles, offer grants or find out the social -driven model.

6.3. Sustainability Metrics for GeoVerde

In order to assess the stability effect of the project, we can focus on the most important metrics related to resource efficiency and environmental protection.

1. Water Use Efficiency (WUE) - Arduino-based Soil Monitoring

The WUE, measured in kilograms/litre, is defined as the ratio of the crop yield (in kilograms) to water consumption (in litres) and can be calculated as follows:

WUE = Crop Yield (kg) / Water Used (litre)

A higher WUE indicates a reduced water consumption, which can be achieved through soil moisture monitoring and precise irrigation.

Example: Suppose traditional irrigation consumes 7,000 litres/hectare per cycle, whereas the Arduino-based soil monitoring can reduce water usage by 1,400 litres/hectare per cycle. As a result, the WUE increases by 20% while maintaining the same crop yield.

2. Pesticide Use Reduction (PUR) - UAV detection of plant diseases

The PUR is defined as the reduction percentage of pesticides achieved by the early detection of plant illness, which can be computed by

PUR (%) = (Baseline Pesticide − Optimised Pesticide) / Baseline Pesticide x 100

Example: Suppose a farm uses 5 kilograms/hectare of pesticides per season. By employing UAV technology for early plant disease detection, pesticide usage can be reduced by 3 kilograms per hectare. As a result, the PUR is 40%.

6.4. Conclusion

Water conservation: Arduino-based soil surveillance increases water use efficiency, which causes large-scale water to save water.

Low chemical use: UNA-based plant detection reduces the use of pesticides, improves soil health and promotes biodiversity.

Improved Crop Health: Through soil monitoring and early detection of diseases in plants, the system averts crop loss and ensures improved quality and yield of crops.

Low-Cost Farming: Intelligent monitoring systems assist in reducing the input cost in the form of water, fertilisers, and pesticides, thus making precision farming less expensive for farmers.

Scalability and Future Potential: Guaranteed future growth through scalable and modular system design with the integration of UAVs for large-area surveillance and enhanced AI for enabling more precise disease diagnosis.

Overall, this project illustrates the potential of emerging technologies in rendering agriculture an efficient, sustainable, and green endeavor.

7. Future Development

One potential advancement for GeoVerde is the ability to detect and differentiate between dewy or frosty leaves and those with powdery surfaces. To achieve this, we would need to test and train the AI to recognise specific leaf conditions. This would involve using image classification with deep learning techniques, specifically Convolutional Neural Networks (CNNs), to train the model effectively. Additionally, conducting stress tests on the AI system will ensure its robustness and precision, allowing it to accurately distinguish between subtle variations in leaf conditions, such as the differences between frost and powder. This targeted training will enhance the accuracy of the diagnostics provided by GeoVerde, ensuring it is highly specialised and reliable for real-world applications.

By placing more soil sensors, more accurate and precise data could be collected for GeoVerde . GeoVerde could increase the granularity of the real-time soil moisture, pH, temperature, and other essential parameters if more sensors are added. Improved irrigation control, enhanced fertilisation, and faster identification of soil dysfunctions are just some of the changes that might result from this growth. Moreover, more locally relevant analysis could be provided with the increased number of sensors leading to reduced resource wastage and improved farming decisions directed towards increasing crop yields and sustainability.

Phase	Tasks and goals	Estimated Duration
Phase 1: Develop TM Prototype	Build & prototype soil data-collection IoT sensors	2-3 months
Phase 2: Develop PDD Prototype	Combine drone with an AI for Plant disease diagnosis	2-3 months
Phase 3: AI Model Training & Testing	Develop an AI to differentiate frost from powdery mildew	2-3 months
Phase 4: Field Trials & Pilot Launch	Adopt for limited application to small scale family farms for usability trials	2-3 months
Phase 5: Market Expansion & Full Release	Refine system for broad application in large scale family farms	4-6 months

8. Conclusion

GeoVerde proposes a low-cost, scalable, and sustainable solution to the problems confronting modern agriculture. Through the integration of advanced sensor networks, AI-driven insights, and real time data capture, It can also provide farmers with actionable intelligence to optimise soil health. GeoVerde can revolutionise precision agriculture, facilitate higher efficiency, reduce wastage and environmental sustainability.

References

Javaid, M., Haleem, A., Khan, I. H., & Suman, R. (2024) 'Unmanned aerial vehicles (UAVs): an adoptable technology for precise agriculture', *Discover Artificial Intelligence*, 4(1), p. 66. Available at: [https://link.springer.com/article/10.1007/s43926-024-00066-5]

Pamuklu, T., Syed, A., Kennedy, W. S., & Erol-Kantarci, M. (2023) 'Heterogeneous GNN-RL Based Task Offloading for UAV-aided Smart Agriculture', *arXiv preprint arXiv:2305.02112*. Available at: [https://arxiv.org/abs/2305.02112]

Rejeb, A., Abdollahi, A., Rejeb, K., & Treiblmaier, H. (2022) 'Drones in agriculture: A review and bibliometric analysis', *Computers and Electronics in Agriculture, Volume 198.* Available at: [https://www.sciencedirect.com/science/article/pii/S0168169922003349]

Shenoy, H., & Chatterjee, S. (2024) 'Precision Agriculture Drones: An AgriTech Trend in 2024'. Available at: [https://www.greyb.com/blog/precision-agriculture-drones/]

Zhang, S., Zhang, C., Yang, C., & Liu, B. (2024) 'Editorial: Artificial intelligence and Internet of Things for smart agriculture', *Frontiers in Plant Science*, 15, p. 1494279. Available at: [https://www.frontiersin.org/articles/10.3389/fpls.2024.1494279/full]

Think with Niche (2024) 'Top AI Innovations Transforming Agriculture in 2024: The Future of Farming Is Here'. Available at: [https://www.thinkwithniche.com/blogs/details/top-ai-innovations-transforming-agriculture-in-2024-the-future-of-farming-is-here]

Li, T., Shu, J., Chen, Q., Abrar, M. M., & Raiti, J. (2024) 'Threshold-Based Automated Pest Detection System for Sustainable Agriculture', *arXiv preprint arXiv:2410.19813*. Available at: [https://arxiv.org/abs/2410.19813]

Appendices

Appendix A - Python Code for PDD Model Training in Jupyter Notebook

```
# Dataset source for this code:
https://www.kaggle.com/datasets/rashikrahmanpritom/plant-disease-recognition-dataset

import os # os module for working with file system

# Define a function to count the number of files in a given folder

def total_files(folder_path):

   num_files = len([f for f in os.listdir(folder_path) if
os.path.isfile(os.path.join(folder_path, f))])

   return num_files

# Define paths for training images

train_files_healthy = "Dataset/Train/Train/Healthy"

train_files_powdery = "Dataset/Train/Train/Powdery"

train_files_rust = "Dataset/Train/Train/Rust"

# Define paths for test images

test_files_healthy = "Dataset/Test/Test/Healthy"

test_files_powdery = "Dataset/Test/Test/Powdery"

test_files_rust = "Dataset/Test/Test/Rust"

# Define paths for validation images

valid_files_healthy = "Dataset/Validation/Validation/Healthy"

valid_files_powdery = "Dataset/Validation/Validation/Powdery"

valid_files_rust = "Dataset/Validation/Validation/Rust"
```

```
from PIL import Image # Python Imaging Library (PIL) module to handle image processing
import IPython.display as display # IPython's display module to show images in Jupyter Notebook
# Display an example healthy plant image used for training
image_path = 'Dataset/Train/Train/Healthy/8bf87605d2b3a323.jpg'
with open(image_path, 'rb') as f:
   display.display(display.Image(data=f.read(), width=500))
```

```
# Display an example rust plant image used for training
image_path = 'Dataset/Train/Train/Rust/8abc65c20f33e4da.jpg'
```

```
with open(image_path, 'rb') as f:

   display.display(display.Image(data=f.read(), width=500))
```

```
```

```
# Display an example powdery plant image used for training

image_path = 'Dataset/Train/Train/Powdery/8de3e321b1f96c03.jpg'

with open(image_path, 'rb') as f:

   display.display(display.Image(data=f.read(), width=500))
```

```
```

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator # 
ImageDataGenerator class for image augmentation

# Create an image data generator for training data with various augmentations

train_datagen = ImageDataGenerator(rescale=1./255, shear_range=0.2, zoom_range=0.2, 
horizontal_flip=True)

# Create an image data generator for testing data

test_datagen = ImageDataGenerator(rescale=1./255)
```

```
   2025-03-13 14:26:53.565815: I tensorflow/core/platform/cpu_feature_guard.cc:210] 
This TensorFlow binary is optimized to use available CPU instructions in 
performance-critical operations.

   To enable the following instructions: AVX2 FMA, in other operations, rebuild 
TensorFlow with the appropriate compiler flags.

# Create a generator for loading training images
```

```
train_generator = train_datagen.flow_from_directory('Dataset/Train/Train',
                                                    target_size=(225, 225),
                                                    batch_size=32,
                                                    class_mode='categorical')
# Create a generator for loading validation images
validation_generator =
test_datagen.flow_from_directory('Dataset/Validation/Validation',
                                                        target_size=(225, 225),
                                                        batch_size=32,
                                                        class_mode='categorical')
```

```
   Found 1322 images belonging to 3 classes.
   Found 60 images belonging to 3 classes.
```

```
# Import Sequential model from Keras with necessary layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
# Define Sequential model with different layers
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(225, 225, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(3, activation='softmax'))
```

```
/opt/anaconda3/lib/python3.12/site-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

    super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

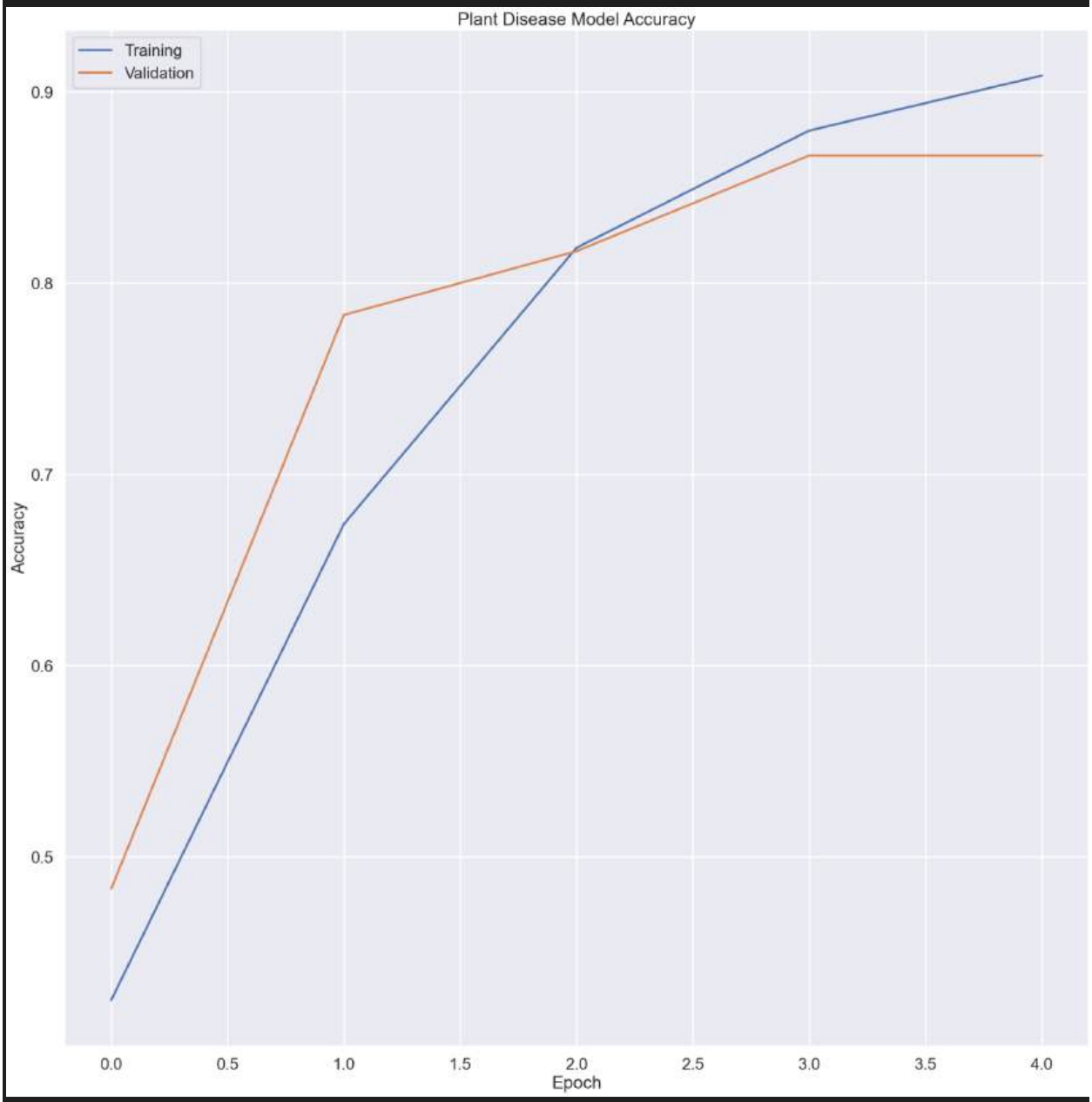
```
# Compile the model with the Adam optimizer, categorical cross-entropy loss and accuracy as the evaluation metric

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
# Train the model using the training data and validate it using the validation data

history = model.fit(train_generator,

                    batch_size=16,

                    epochs=10,

                    validation_data=validation_generator,

                    validation_batch_size=16

                    )
```

```
/opt/anaconda3/lib/python3.12/site-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_multiprocessing`, `max_queue_size`. Do not pass these arguments to `fit()`, as they will be ignored.

    self._warn_if_super_not_called()



  Epoch 1/10

  [1m42/42[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m182s[0m 4s/step - accuracy: 0.4229 - loss: 4.7968 - val_accuracy: 0.7667 - val_loss: 0.6366

  Epoch 2/10

  [1m42/42[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m170s[0m 4s/step - accuracy: 0.7874 - loss: 0.5246 - val_accuracy: 0.8167 - val_loss: 0.5872
```

```
   Epoch 3/10

   [1m42/42[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m171s[0m 4s/step -
accuracy: 0.8622 - loss: 0.3637 - val_accuracy: 0.7333 - val_loss: 0.5711

   Epoch 4/10

   [1m42/42[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m172s[0m 4s/step -
accuracy: 0.8875 - loss: 0.3333 - val_accuracy: 0.8333 - val_loss: 0.4692

   Epoch 5/10

   [1m42/42[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m170s[0m 4s/step -
accuracy: 0.9354 - loss: 0.1998 - val_accuracy: 0.8667 - val_loss: 0.3179

   Epoch 6/10

   [1m42/42[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m170s[0m 4s/step -
accuracy: 0.9318 - loss: 0.2032 - val_accuracy: 0.8333 - val_loss: 0.5193

   Epoch 7/10

   [1m42/42[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m170s[0m 4s/step -
accuracy: 0.9077 - loss: 0.2551 - val_accuracy: 0.8833 - val_loss: 0.3309

   Epoch 8/10

   [1m42/42[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m171s[0m 4s/step -
accuracy: 0.9347 - loss: 0.2010 - val_accuracy: 0.8667 - val_loss: 0.4344

   Epoch 9/10

   [1m42/42[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m170s[0m 4s/step -
accuracy: 0.9444 - loss: 0.1754 - val_accuracy: 0.9000 - val_loss: 0.2780

   Epoch 10/10

   [1m42/42[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m170s[0m 4s/step -
accuracy: 0.9563 - loss: 0.1504 - val_accuracy: 0.8667 - val_loss: 0.3822



from matplotlib import pyplot as plt # pyplot module for plotting

from matplotlib.pyplot import figure # figure function for setting the figure size

import seaborn as sns # seaborn module for plot styling



```

```
sns.set_theme() # default seaborn theme for plots

sns.set_context("poster") # for larger fonts and elements

figure(figsize=(25, 25), dpi=100) # figure size: 25x25 inches, resolution: 100 dpi

# Plot the accuracy of training and validation data across epochs

plt.plot(history.history['accuracy']) # Plot the training accuracy

plt.plot(history.history['val_accuracy']) # Plot the validation accuracy

plt.title('Plant Disease Model Accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['Training', 'Validation'], loc='upper left')

plt.show()
```




```
# Save the trained model

model.save("plantdisease.keras")

```

```
from tensorflow.keras.preprocessing.image import load_img, img_to_array # Keras
functions for loading and converting images

import numpy as np # NumPy module for array manipulations

# Define a function to preprocess an image for model input

def preprocess_image(image_path, target_size=(225, 225)):

   img = load_img(image_path, target_size=target_size)

   x = img_to_array(img)

   x = x.astype('float32') / 255.

   x = np.expand_dims(x, axis=0)

   return x

# Display an example rust plant image used for testing

image_path = 'Dataset/Test/Test/Rust/89cb83b03f3c60cf.jpg'

with open(image_path, 'rb') as f:

   display.display(display.Image(data=f.read(), width=500))
```

```
# Preprocess the image

x = preprocess_image(image_path)

# Use the trained model to make predictions on the preprocessed image

predictions = model.predict(x)

predictions[0]
```

```
   [1m1/1[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 87ms/step

   array([7.4444663e-19, 4.6984855e-17, 1.0000000e+00], dtype=float32)
```

```
# Retrieve the class indices from the training generator

labels = train_generator.class_indices

labels = {v: k for k, v in labels.items()} # invert the class indices to map to class
labels
```

```
labels
```

```
   {0: 'Healthy', 1: 'Powdery', 2: 'Rust'}
```

```
# Find the index of the highest probability in the prediction result
predicted_label = labels[np.argmax(predictions)]
print(predicted_label) # print the predicted class label
```

```
   Rust
```

```
# Display an example powdery plant image used for testing
image_path = 'Dataset/Test/Test/Powdery/9f08c72693f34da4.jpg'
with open(image_path, 'rb') as f:
   display.display(display.Image(data=f.read(), width=500))
```

```
x = preprocess_image(image_path)

predictions = model.predict(x)

predictions[0]

predicted_label = labels[np.argmax(predictions)]

print(predicted_label)
```

```
   [1m1/1[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 31ms/step
```

```
   Powdery
```

```
# Display an example healthy plant image used for testing

image_path = 'Dataset/Test/Test/Healthy/8ebc94969f1252d9.jpg'

with open(image_path, 'rb') as f:

   display.display(display.Image(data=f.read(), width=500))
```

```

![jpeg](output_17_0.jpg)

```
x = preprocess_image(image_path)

predictions = model.predict(x)

predictions[0]

predicted_label = labels[np.argmax(predictions)]

print(predicted_label)

```

```
   [1m1/1[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 32ms/step

   Healthy
```
```

Appendix B - Python Code for PDD App in VS Code

```
# Import libraries

import os # OS for file path operations

import tensorflow as tf # TensorFlow for deep learning model operations

import numpy as np # NumPy for numerical computations

from tensorflow.keras.preprocessing import image # Keras for image preprocessing

from PIL import Image # Python Imaging Library (PIL) for handling image files

from keras.models import load_model # for loading pre-trained Keras models

from flask import Flask, request, render_template # Flask web for handling requests
and rendering templates

from werkzeug.utils import secure_filename # Utility for securing filenames before
saving

from tensorflow.keras.preprocessing.image import load_img, img_to_array # for loading
and processing images



# Initialise Flask application

app = Flask(__name__)



# Load the pre-trained plant disease detection model

model =load_model('plantdisease.keras')

print('Model loaded. Check http://127.0.0.1:5000/')



# Define class labels corresponding to the model's output indices

labels = {0: 'Healthy', 1: 'Powdery', 2: 'Rust'}



# Function to preprocess image and make prediction

def getResult(image_path):

   img = load_img(image_path, target_size=(225,225)) # load and resize image to match
the model input

   x = img_to_array(img) # convert the image to a NumPy array

   x = x.astype('float32') / 255. # normalise pixel values to the range [0, 1]

   x = np.expand_dims(x, axis=0) # expand dimensions to match the expected input shape
for the model
```

```
    predictions = model.predict(x)[0] # make predictions using the trained model
    return predictions # return the predicted probabilities


# Route for the home page (renders the index.html in 'templates' folder)
@app.route('/', methods=['GET'])
def index():
    return render_template('index.html')


# Route for handling image uploads and making predictions
@app.route('/predict', methods=['GET', 'POST'])
def upload():
    if request.method == 'POST':
        f = request.files['file'] # get the uploaded file
        basepath = os.path.dirname(__file__) # get the base directory of the code
        file_path = os.path.join(basepath, 'uploads', secure_filename(f.filename)) #
create a secure file path
        f.save(file_path) # save the uploaded file
        predictions=getResult(file_path) # get model predictions for the uploaded image
        predicted_label = labels[np.argmax(predictions)] # determine the class label
with the highest probability
        return str(predicted_label) # return the predicted label
    return None # if the request method is not 'POST'


# Run the Flask app when the code is executed
if __name__ == '__main__':
    app.run(debug=True)
```

Appendix C - Python Code for TM Prototype in VS Code

```
# Import libraries
from pyfirmata2 import Arduino, util # Arduino and utility modules from the pyFirmata2
library
import time # time module for adding delays in the code

# Setup parameters
PORT = '/dev/cu.usbmodem144101' # adjust according to your system
TEMP_SENSOR_PIN = 0 # analog pin for temperature sensor LM35
BUZZER_PIN = 8 # digital pin for buzzer
LED_PIN = 10 # digital pin for LED
TEMP_THRESHOLD = 23 # temperature threshold for buzzer alarm (set low for testing)
TEMP_RANGE_MIN = 18 # min temperature for LED
TEMP_RANGE_MAX = 30 # max temperature for LED
SERIAL_UPDATE_INTERVAL = 1 # time interval for serial updates (1 second)
# Define buzzer delay pattern (like Morse code)
BUZZER_PATTERN = [0.1, 0.3, 0.2, 0.1, 0.5] # delay intervals in seconds

# Initialise the Arduino board
board = Arduino(PORT)

# Start an iterator thread to keep the board running
it = util.Iterator(board)
it.start()

# Enable analog input for the temperature sensor
board.analog[TEMP_SENSOR_PIN].enable_reporting()

# Global variable to store the latest temperature reading
temperature = None

# Function to update temperature when LM35 sensor value changes
def handle_analog_read(value):
   global temperature
   if value is not None:
       voltage = value * 5.0 # convert the sensor value to voltage (0-5V range)
       temperature = voltage * 100.0 # convert the voltage to temperature in Celsius
(10mV per °C)

# Attach the callback function to the LM35 sensor
board.analog[TEMP_SENSOR_PIN].register_callback(handle_analog_read)
```

```
# Start reading analog values
board.analog[TEMP_SENSOR_PIN].enable_reporting()

# Timer to control serial output frequency
last_serial_time = time.time()
# Function to make the buzzer beep using a predefined pattern of time delays
def buzzer_pattern():
    for delay in BUZZER_PATTERN:
        board.digital[BUZZER_PIN].write(1) # turn buzzer ON
        time.sleep(delay) # wait for ON duration
        board.digital[BUZZER_PIN].write(0) # turn buzzer OFF
        time.sleep(0.1) # short pause between beeps

# Function to turn on the LED when temperature is within the defined range
def handle_led(temperature):
    if TEMP_RANGE_MIN <= temperature <= TEMP_RANGE_MAX:
        board.digital[LED_PIN].write(1) # turn on LED
        print("Temperature within range! LED ON.")
    else:
        board.digital[LED_PIN].write(0) # turn off LED
        print("Temperature out of range. LED OFF.")
try:
    while True:
        if temperature is not None:
            # Check if temperature exceeds threshold
            if temperature > TEMP_THRESHOLD:
                print("High Temperature!")
                buzzer_pattern() # activate buzzer
            else:
                board.digital[BUZZER_PIN].write(0) # turn off buzzer
            # Handle LED based on temperature range
            handle_led(temperature)
            # Print temperature every 1 second
            if time.time() - last_serial_time > SERIAL_UPDATE_INTERVAL:
                print(f"Temperature: {temperature:.2f}C")
                last_serial_time = time.time()
        time.sleep(0.1) # small delay to prevent excessive CPU usage
except KeyboardInterrupt:
    print("Exiting...")
    board.exit()
```