PROJECT REPORT ON

AgroSense: Smart Agriculture System

Submitted to

Department of Computer Applications in

partial fulfillment for the award of the degree of

BACHELOR OF COMPUTER APPLICATIONS with AI and DS

Batch (2023-2026)

Submitted by

Name of the student: Diya Munshi

Enrollment Number: GE-23214001

Under the Guidance of

Ms. Aakriti Singh
(Assistant Professor)

GRAPHIC ERA DEEMED TO BE UNIVERSITY DEHRADUN

November -2025

Graphic Era
Deemed to be University

CANDIDATE'S DECLARATION

I hereby certify that the work presented in this project report entitled "**AgroSense: Smart Agriculture System"** in partial fulfilment of the requirements for the award of the degree of **Bachelor of Computer Applications with Artificial Intelligence and Data Science** is a bonafide work carried out by us during the period of July 2025 to December 2025 under the supervision of **Ms. Aakriti Singh** Department of Computer Application, Graphic Era Deemed to be University, Dehradun, India.

This work has not been submitted elsewhere for the award of a degree/diploma/certificate.

Name and Signature of Candidate

This is to certify that the above mentioned statement in the candidate's declaration is correct to the best of my knowledge.

Date: _____ **Name and Signature of Guide**

Signature of Supervisor **Signature of External Examiner**

HOD

CERTIFICATE OF ORIGINALITY

This is to certify that the project report entitled "**AgroSense: Smart Agriculture System"** is submitted to **Graphic Era University, Dehradun** in partial fulfillment of the requirement for the award of the degree of **BACHELOR OF COMPUTER APPLICATIONS (BCA) AI/DS** , is an authentic and original work carried out by **Ms. Diya Munshi** with enrolment number **GE-23214001** under my supervision and guidance.

The matter embodied in this project is genuine work done by the student and has not been submitted whether to this University or to any other University / Institute for the fulfillment of the requirements of any course of study.

Signature of the Student:

Date: ……..

Signature of the Guide

Date: …….

Name and Details of the Student:

Name: Diya Munshi

Enrolment No.: GE-23214001

Name and Designation of the Guide:

Ms. Aakriti Singh

Assistant Professor

Special Note:

Acknowledgement

I would like to express my sincere gratitude to everyone who has contributed to the success of this project. Without their guidance, support, and encouragement, this work would not have been possible.

First and foremost, we would like to extend our deepest appreciation to our mentor Ms. Aakriti Singh, for their invaluable mentorship, expert advice, and continuous encouragement throughout the development of this project. Their insights and suggestions helped shape the project in its current form.

I would also like to thank the Department of Computer Applications at Graphic Era Deemed to be University, Dehradun, India for providing us with the necessary resources and a conducive learning environment to complete this project. Special thanks to all my professors and staff members who have supported us throughout the course of my studies.

A heartfelt thank you to family and friends for their understanding, patience, and unwavering support. Their belief in our abilities provided us with the motivation to complete this project.

Lastly, I would like to acknowledge the valuable contributions from the open-source community, including the libraries and tools we used in developing the system, which significantly enhanced the development process.

Thanks to everyone who has been part of this journey.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Overview

Agriculture is one of the most essential sectors of the world's economy, and efficient water management plays a major role in ensuring crop productivity. Traditional irrigation practices rely heavily on manual decision-making and fixed schedules, often leading to under-irrigation, over-irrigation, or unnecessary water wastage.

AgroSense is a smart agriculture assistance system designed to provide **intelligent, data-driven irrigation recommendations** using machine learning. The system analyzes key environmental parameters such as **temperature, soil moisture, atmospheric pressure, and altitude**, and processes them using a trained Logistic Regression model to predict whether irrigation is required.

The system also integrates **real-time weather data** through the OpenWeather API to improve prediction accuracy and reduce water usage during rainfall conditions. Additionally, AgroSense provides an optional **plant disease detection module**, allowing farmers to upload leaf images and receive instant diagnosis using a deep learning model.

AgroSense combines machine learning, computer vision, and a modern user interface to deliver a **smart, efficient, and user-friendly irrigation support tool** for farmers, researchers, and agricultural students.

1.2 Motivation

Water scarcity and inefficient irrigation practices are major challenges in modern agriculture. Farmers frequently depend on guesswork for irrigation, as measuring soil and environmental conditions manually is time-consuming and often inaccurate. Over-irrigation leads to **water waste, nutrient loss, and soil damage**, while under-irrigation results in **poor crop growth and reduced yield**.

This project is motivated by the need to:

- **Optimize water usage** using intelligent predictions.
- Provide **affordable smart agriculture solutions** without requiring expensive IoT hardware.
- Reduce human error in irrigation decision-making.
- Develop a platform that integrates **machine learning + weather data** for accurate recommendations.
- Support farmers with a **simple, clean, and easy-to-use interface**.
- Enable students to explore how artificial intelligence can enhance real-world agricultural practices.

By developing AgroSense, the goal is to demonstrate how data science and AI can solve real agricultural problems and promote sustainable farming.

1.3 Objectives and Scope

Objectives

- To build a **machine learning–based irrigation prediction model** using environmental parameters.
- To provide **real-time weather integration** using APIs to improve decision accuracy.
- To design a **modern, user-friendly web interface** for farmers and agricultural students.
- To allow users to input temperature, soil moisture, pressure, and altitude to receive irrigation advice.
- To enhance decision-making efficiency and promote water conservation.
- To implement a **plant disease detection module** using deep learning (MobileNetV2).
- To provide disease name, symptoms, causes, and treatments for the uploaded plant leaf image.
- To integrate transparent glass-style UI design for better visibility and usability.
- To demonstrate the use of Flask backend with HTML, CSS, and JavaScript frontend.

Scope of the Project

The scope of AgroSense includes:

✔ ML model for irrigation prediction
✔ Weather data integration
✔ Web-based interface (HTML, CSS, JS, Flask)
✔ Disease detection system (optional but included in the project)
✔ Static backgrounds + mobile-friendly UI
✔ Deployment-ready backend

AgroSense is designed as a **software-based smart agriculture assistant**, suitable for demonstrations, academic projects, and conceptual farm decision support.

1.4 Hardware and Software Requirements

For the system to function as expected, a few technical requirements are necessary:

Hardware Requirements

Since AgroSense is a software-based system, the hardware requirements are minimal:

- A laptop or computer with at least:
    - **Processor:** Intel or Apple Silicon (recommended)
    - **RAM:** 8 GB (for running TensorFlow)
    - **Storage:** 2–3 GB free space
- Optional (for future extension):
    - Soil moisture sensor
    - Temperature/humidity sensor
    - Raspberry Pi or NodeMCU

Software Requirements

- **Programming Language:** Python 3.10+
- **Backend Framework:** Flask
- **Frontend:** HTML, CSS, JavaScript

- **Libraries & Tools:**
    - Scikit-learn
    - TensorFlow / Keras
    - NumPy
    - Joblib
    - Pillow
    - Requests
    - Python-dotenv
- **Other Tools:**
    - VS Code / PyCharm
    - Postman (for API testing)
    - Browser (Chrome/Safari/Firefox)
- **Operating System Compatibility:**
    - macOS
    - Windows

This chapter introduced the idea behind my AgroSense Smart agriculture System, including the motivation for building a water-efficient solution and the goals I aimed to achieve. I explained how the system uses machine learning to predict whether irrigation is needed based on environmental factors such as temperature, pressure, altitude, and soil moisture. The chapter also described how weather data and a clean web interface help make the system more reliable and easy to use. In the next chapters, I will explain the research behind the project, how the system was designed and implemented, the challenges faced during development, and the final outcomes of the AgroSense application.

Chapter 2

System Analysis & Requirements Specifications

2.1 System Architecture

The AgroSense Smart Irrigation System follows a **three-tier architecture** consisting of:

1. User Interface Layer (Frontend)

- Built using **HTML, CSS, and JavaScript**.
- Provides an easy-to-use interface for:
    - Entering environmental parameters (temperature, pressure, altitude, soil moisture)
    - Uploading plant leaf images
    - Viewing irrigation predictions
    - Viewing disease detection results
- Background images and glass-morphism effects provide a clean, modern UI.

2. Application Layer (Flask Backend)

Handles the main logic of the system:

- Processes user input
- Loads machine learning and deep learning models
- Communicates with external APIs
- Generates irrigation recommendations
- Performs plant disease classification
- Returns responses as JSON or rendered HTML

3. Machine Learning Layer

Includes two models:

Irrigation Prediction Model (Logistic Regression)

- Input: Temperature, soil moisture, pressure, altitude

- Output: “Irrigation required” or “Not required”
- Supports weather-aware advice for accurate decisions.

Plant Disease Detection Model (MobileNetV2)

- Input: Leaf image
- Output: Disease name, confidence, cause, symptoms & treatment
- Uses transfer learning for high accuracy.

4. External API Layer

- Uses **OpenWeather API** to fetch:
    - Temperature
    - Rain forecast
    - Pressure
    - Humidity
- This strengthens irrigation recommendations.

5. Data Storage Layer

- Stores trained ML models:
    - irrigation_model.pkl
    - plant_disease_model.h5
- No database is required in the current system.

Overall Flow

User → Frontend → Flask API → ML Model → Result → Frontend UI

Weather API → Flask → Enhances Recommendation

2.2 Data Flow Diagrams (DFD)

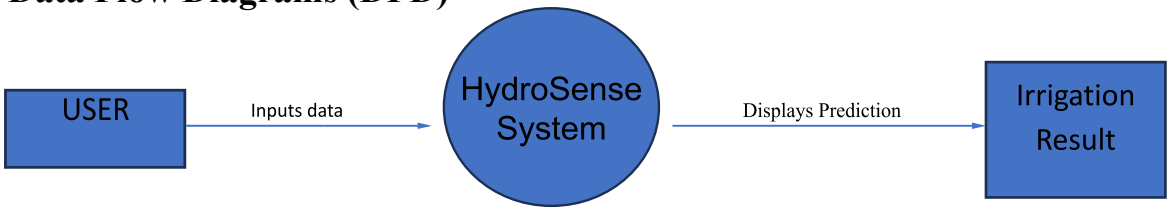

Figure 2.2.1: Level 0 DFD

- The user enters input.
- The system processes the data and predicts the irrigation requirement.
- The prediction result is shown to the user.

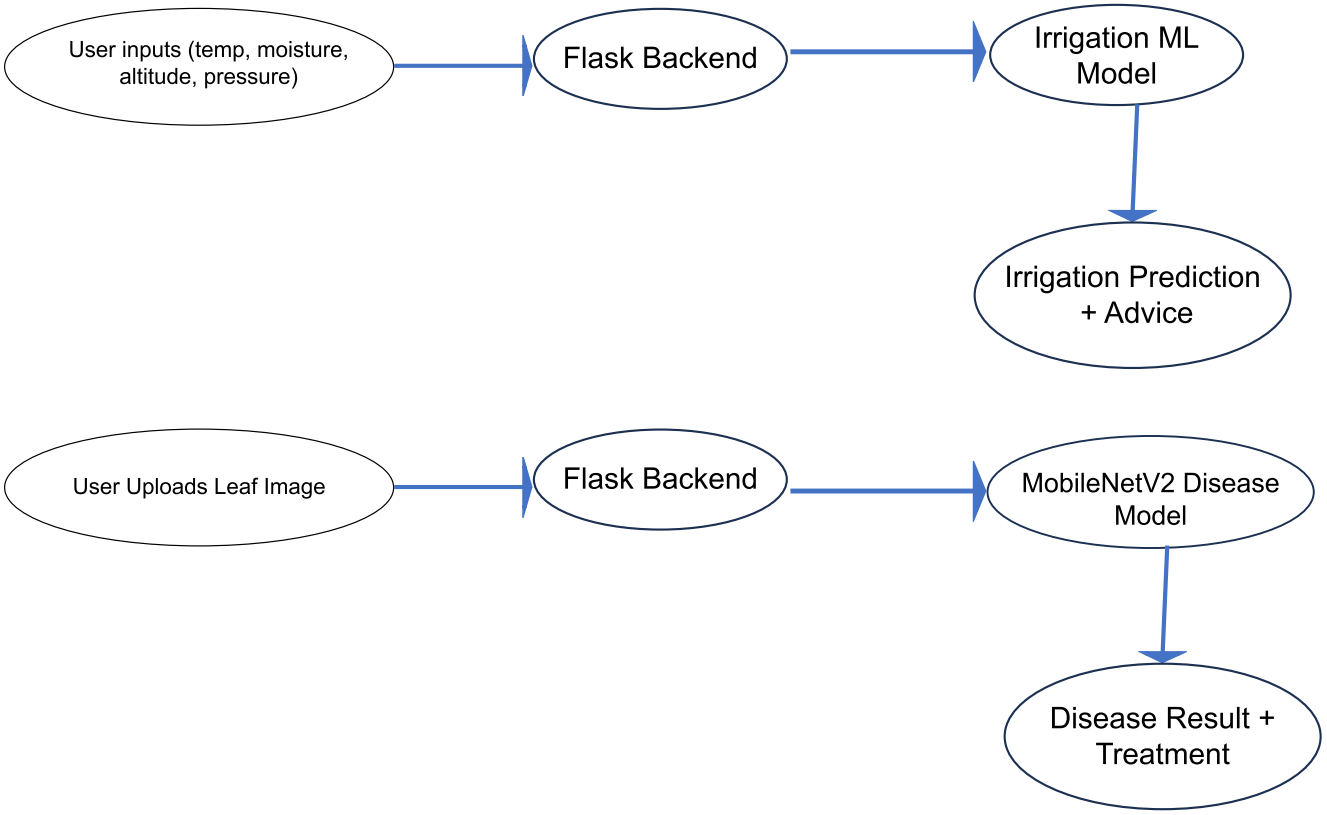

Figure 2.2.2: Level 1 DFD - Detailed Process Flow

This shows how data flows from the front-end to the machine learning model, gets processed and returns the result.

2.3 Use Case Diagram

The Use Case Diagram helps us visualize how different actors interact with the application. It shows the various tasks or actions that can be performed by the actors.

Actors

- User (Farmer / Student) – interacts with the system
- Weather API – provides live weather data
- ML Models – process predictions

Use Case List

1. Enter environmental parameters
2. Request irrigation prediction
3. Upload plant leaf image
4. Request disease detection
5. View recommended irrigation advice
6. View disease symptoms & treatment
7. Fetch weather data

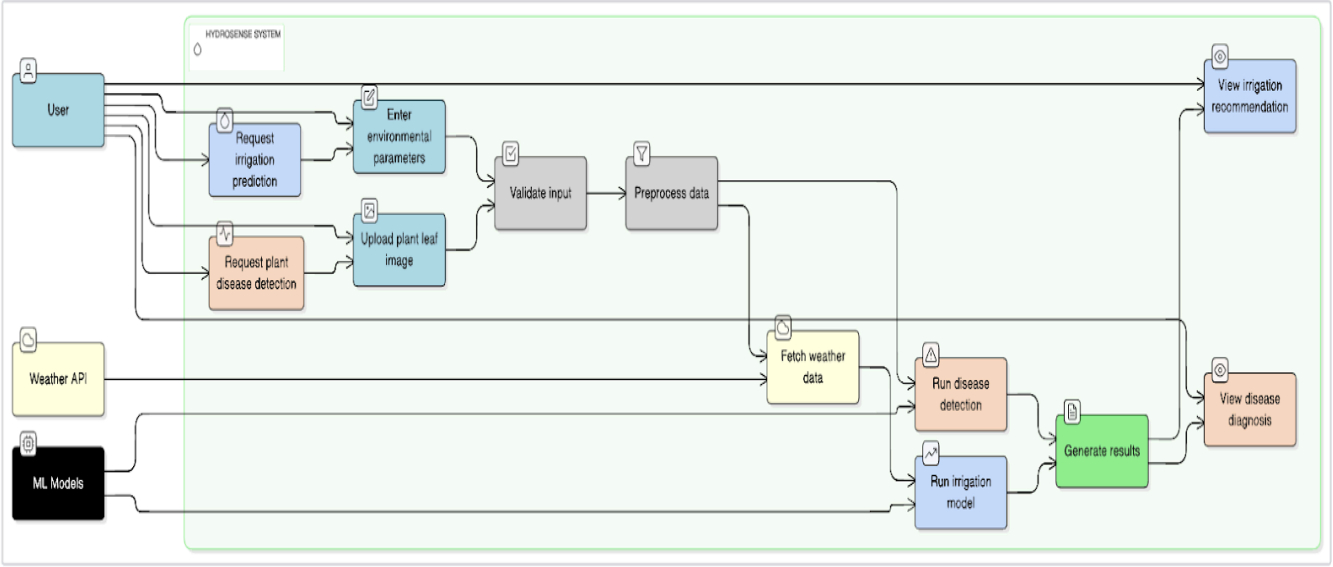

Figure 2.3: Use Case Diagram Showing Interactions

The use case diagram shows how the user interacts with the AgroSense Smart Irrigation System. In this project, there is one primary actor – the **User**, who can be a farmer, agriculture student, or anyone wanting to know whether a crop needs irrigation.

Actors:

User: The person who provides environmental details to the system. The user can also upload a plant leaf image (optional feature) and view the predictions generated by the ML models.

Main Use Cases:

1. Enter Environmental Data

The user enters important crop-related environmental parameters such as:

- Temperature
- Soil Moisture
- Atmospheric Pressure
- Altitude

These values are required to check whether irrigation is needed.

2. Submit Data for Prediction

After entering the values, the user clicks a button to send the data to the backend.
The system then processes the inputs and prepares them for model prediction.

3. Receive Irrigation Recommendation

The backend uses a trained **Logistic Regression model** to analyze the input data and returns a clear suggestion like:

- *"Irrigation required"*
- *"No irrigation needed"*
- Along with an explanation or advice based on weather and crop conditions.

The result is displayed directly on the webpage.

4. Upload Plant Leaf Image (Optional Feature)

The user can upload an image of a plant leaf if they want to check whether the plant has a disease.

5. Receive Plant Disease Diagnosis

The system processes the image using a **MobileNetV2 deep learning model** and returns:

- Disease name
- Confidence level
- Cause
- Symptoms
- Suggested treatment

This helps the user understand the health of the crop.

In this chapter, I explained the system architecture and how the different components of AgroSense work together. The Data Flow Diagrams (DFD) show how information moves

between the user, the backend, and the machine learning models. The Use Case Diagram helps visualize how the user interacts with the system by entering environmental data, uploading images, and receiving predictions. Understanding these diagrams and analysis steps was important before starting development, as it helped me clearly define the roles, inputs, processes, and outputs of the system. With this foundation, I was able to plan the development of AgroSense more effectively and build a smarter and more user-friendly irrigation decision support system.

Chapter 3

System Design

3.1 System Design Overview

The System Design phase describes how the AgroSense Smart Agriculture System is structured internally and how each component works together to achieve the overall functionality. The design focuses on transforming user inputs into meaningful irrigation recommendations through a series of logical steps involving data preprocessing, model prediction, and information visualization.

AgroSense uses a modular design consisting of:

1. User Interface Layer (Frontend)

- Designed using HTML, CSS, and JavaScript.
- Provides fields for entering temperature, pressure, altitude, and soil moisture.
- Allows users to upload plant images.
- Displays prediction results clearly.

2. Application Layer (Flask Backend)

- Acts as the bridge between the UI and machine learning models.
- Handles:
    - API requests
    - Input validation
    - Preprocessing
    - Model prediction
    - Weather API communication

3. Machine Learning Layer

Includes two major components:

a. Irrigation Prediction Model

- Logistic Regression algorithm
- Determines if irrigation is required
- Uses normalized environmental data

b. Plant Disease Detection Model

- MobileNetV2 (Transfer Learning)
- Processes image input
- Predicts disease class and treatment

4. Weather API Integration

- Retrieves live temperature, humidity, pressure, and rain forecast.
- Enhances system accuracy by preventing irrigation during rainfall.

5. Output Layer

- Displays:
    - Irrigation advice
    - Weather-adjusted guidance
    - Disease diagnosis and treatment

This structured design ensures that AgroSense is scalable, easy to maintain, and capable of producing accurate results.

3.2 Flowchart

To better understand how AgroSense processes data, we can visualize the entire workflow using a flowchart. The system begins when the user enters environmental values or uploads a plant image. After basic validation, the backend fetches weather information (if needed), preprocesses the input, and passes it to the appropriate machine learning model. The irrigation model predicts whether watering is required, while the disease model identifies plant diseases from images. Finally, the system returns a clear recommendation or diagnosis, which is displayed to the user. This flowchart outlines each of these steps from start to finish.

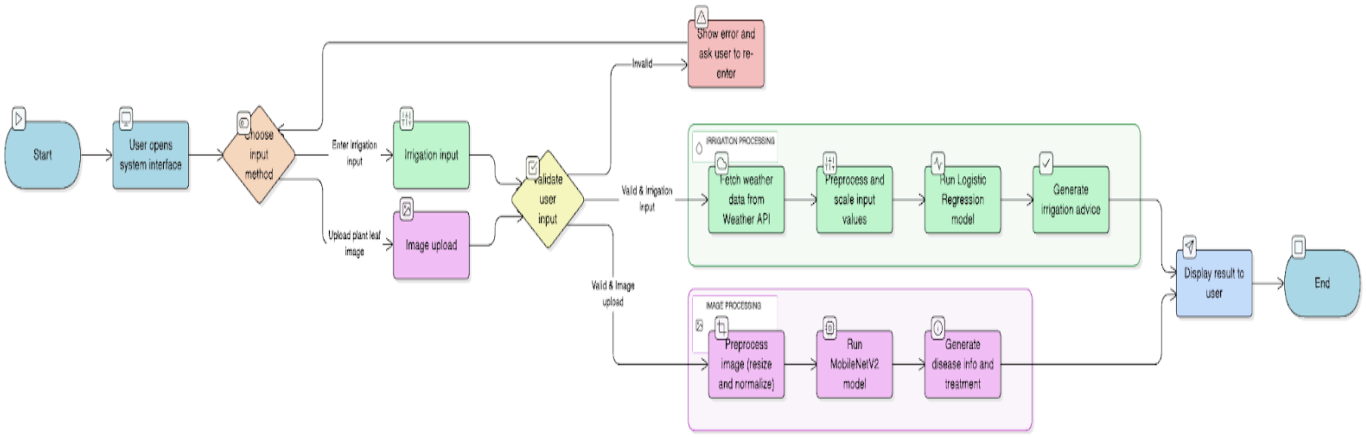

Figure 3.1: Flowchart of the system design.

3.3 Algorithm

Here is the algorithm used in AgroSense for Irrigation Prediction:

Irrigation Prediction Algorithm

Step 1: Start the system.
Step 2: Accept user inputs:

- Temperature
- Soil moisture
- Pressure
- Altitude

Step 3: Validate all inputs.
If any value is missing → show an error.

Step 4: Fetch real-time weather information using the OpenWeather API.
Extract:

- Temperature
- Pressure
- Rain forecast
- Humidity

Step 5: Preprocess input data:

- Convert values to numerical format
- Normalize using StandardScaler

Step 6: Feed the preprocessed data into the Logistic Regression model.

Step 7: Receive model output:

- 0 or 1 → Irrigation Required
- 2 or 3 → Irrigation Not Needed

Step 8: Generate advice:

- If rainfall expected → suggest skipping irrigation
- If high temperature → recommend watering
- If high soil moisture → irrigation not needed

Step 9: Send the final recommendation to the frontend.

Step 10: Display advice to the user.

Step 11: End the process.

Plant Disease Detection Algorithm

Step 1: User uploads a leaf image.
Step 2: Convert image to RGB and resize to 224×224.
Step 3: Normalize pixel data.
Step 4: Pass the image through MobileNetV2 model.

Step 5: Get the predicted disease class.

Step 6: Fetch:

- Cause
- Symptoms
- Treatment

Step 7: Display the result.

Chapter 4

Project Management

4.1 Project Planning and Scheduling

Effective planning and scheduling were essential for developing the AgroSense Smart Agriculture System. The project was divided into multiple stages to ensure smooth development, reduce errors, and maintain a clear workflow. Each phase focused on a specific goal, helping to manage time and resources efficiently.

1. Requirement Analysis (Week 1)

- Understanding the problem of water wastage in agriculture
- Researching existing irrigation systems
- Identifying input parameters (temperature, soil moisture, pressure, altitude)
- Deciding to add optional plant disease detection

2. System Design (Week 2)

- Designing architecture diagrams
- Creating DFDs and use case diagrams
- Planning the communication flow between frontend, backend, and ML models

3. Dataset Preparation & Model Training (Week 3–4)

- Collecting environmental and plant disease datasets
- Preprocessing data
- Training Logistic Regression for irrigation prediction
- Training MobileNetV2 for plant disease detection
- Saving models for deployment

4. Backend Development (Week 5)

- Building API endpoints using Flask
- Integrating ML models

- Adding weather API functionality
- Handling input validation and error responses

5. Frontend Development (Week 6)

- Creating UI using HTML, CSS, JavaScript
- Designing irrigation and disease detection pages
- Implementing dynamic result display

6. Integration & Testing (Week 7)

- Connecting frontend with backend
- Testing model predictions
- Checking weather API responses
- Fixing UI and logic bugs

7. Documentation & Final Presentation (Week 8)

- Preparing report, diagrams, and screenshots
- Creating the final demonstration flow
- Writing conclusion and future scope

4.2 Risk Management

Risk management helps identify possible challenges during development and defines strategies to handle them. In AgroSense, several technical and operational risks were considered to ensure smooth project completion.

Key Risks and Mitigation Strategies

1. Model Accuracy Risk

Risk: The ML model may give inaccurate irrigation predictions.
Mitigation:

- Use proper preprocessing and scaling
- Test with multiple sample values

- Fine-tune Logistic Regression and evaluate performance

2. Weather API Failure

Risk: API may be unavailable or slow, affecting predictions.
Mitigation:

- Handle API failure with fallback values
- Display a message: “Weather data unavailable, using manual inputs only.”

3. Incorrect User Input

Risk: Wrong or missing values (e.g., negative temperature).
Mitigation:

- Add frontend and backend validation
- Show clear error messages

4. Model Loading/Deployment Errors

Risk: TensorFlow or joblib model might fail to load on different machines.
Mitigation:

- Fix versions in requirements.txt
- Test the environment using a virtual environment (venv)

5. Performance Issues

Risk: Image prediction may be slow due to MobileNetV2.
Mitigation:

- Compress the model
- Use optimized preprocessing
- Limit image size

6. Data Security Risk

Risk: User input and plant images may not be properly handled.

Mitigation:

- Avoid storing user data unnecessarily
- Validate and sanitize all requests

7. UI/UX Errors

Risk: Users may misinterpret results or fail to understand inputs.
Mitigation:

- Keep UI simple and responsive
- Provide tooltips or placeholder hints

Chapter 5

Input Design

5.1 Input Screens/Forms

Input design focuses on how users provide data to the system. AgroSense includes two main input areas: *Irrigation Prediction Input* and *Plant Disease Detection Input*. Both are designed to be simple, user-friendly, and easy to understand.

1. Irrigation Prediction Input Form

This form allows the user to enter environmental values required by the ML model. It contains the following fields:

- **Temperature (°C)** — Numeric input
- **Soil Moisture (%)** — Numeric input
- **Atmospheric Pressure (hPa)** — Numeric input
- **Altitude (meters)** — Numeric input

Additional features in the form:

- **Fetch Weather Button** – Automatically gets temperature and pressure using the Weather API.
- **Submit Button** – Sends the values to the backend to generate irrigation advice.

The form is designed with:

- Clear labels
- Placeholder values
- Validation to avoid empty or incorrect inputs
- A clean UI so users understand exactly what to enter

2. Plant Disease Detection Input Screen

This screen allows the user to upload a plant leaf image for disease detection.

Key input components:

- **Choose Image Button** — Opens file picker
- **Preview Window** — Displays the selected leaf image
- **Analyze Button** — Sends the image to the backend

The form accepts:

- JPG / PNG images
- Images automatically resized and processed by the backend

This input screen is especially useful for identifying tomato, potato, and pepper leaf diseases using deep learning.

3. Error Handling Inputs

The system validates input before processing:

- Alerts for missing values
- Errors for invalid numbers
- Prompts if no image is selected

This ensures that the ML models receive clean, meaningful data.

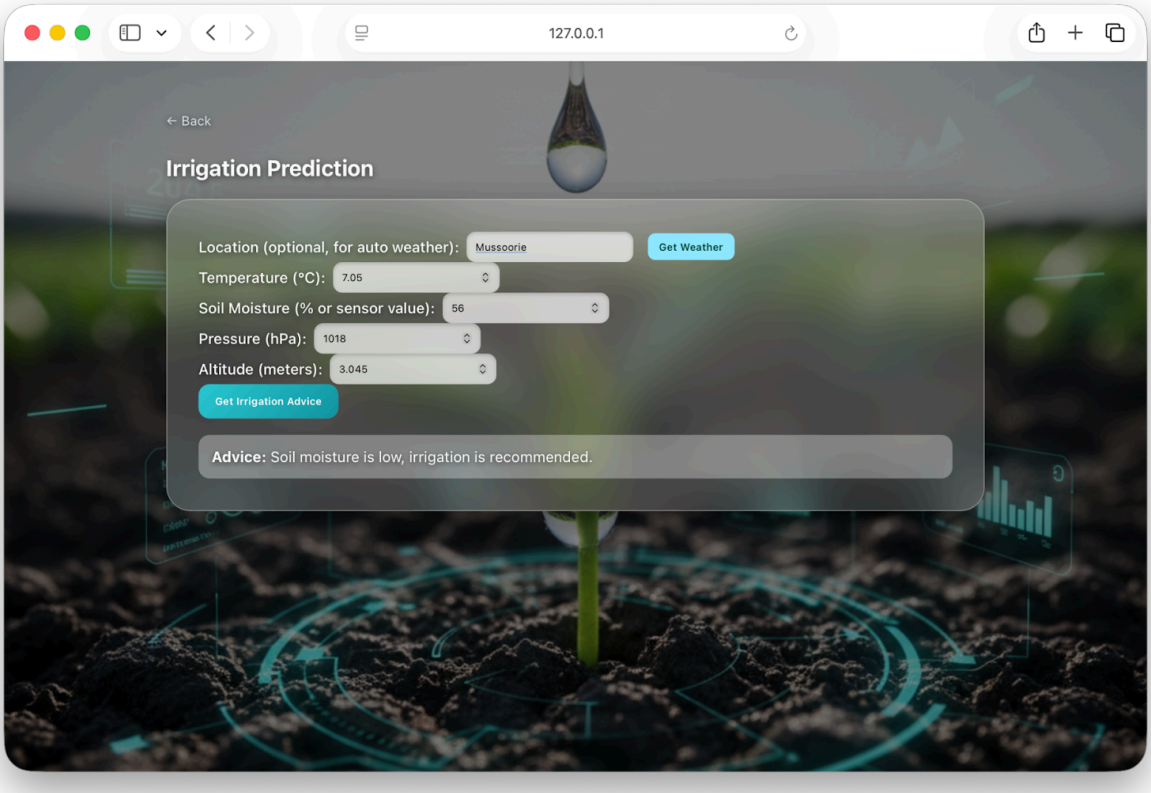

Figure 5.1: Example of an Input Screen

5.2 Input Flow

The input flow describes how data moves from the user to the backend system, and how it is prepared for prediction.

Below is the simplified flow:

Input Flow for Irrigation Prediction

1. **User Opens the Irrigation Page**
   The user sees the form with four fields and a weather fetch option.
2. **User Enters Environmental Data**
   Inputs like temperature, soil moisture, pressure, and altitude are typed into the form.

3. **Frontend Validation**

   The browser checks if fields are empty or incorrect.

   If invalid → show a warning.

4. **Send Input to Backend**

   When the user clicks "Predict", the frontend sends the data as JSON to:

   /predict/irrigation

5. **Backend Preprocessing**
   - Convert values to numeric
   - Scale inputs using StandardScaler
   - Prepare data shape for the ML model
6. **Model Prediction**

   Logistic Regression analyzes the scaled values and predicts irrigation requirement.

7. **Backend Generates Advice**
   - Combines prediction result
   - Adds weather-based tips (e.g., rain expected → irrigation not needed)
8. **Send Response to UI**

   The backend sends JSON response to the frontend.

9. **Display Result to User**

   The webpage shows the irrigation recommendation and explanation.

Input Flow for Disease Detection

1. **User Uploads Leaf Image**

   The file is selected through the upload button.

2. **Image Preview**

   The user sees a preview to confirm the correct file.

3. **Send Image to Backend**

   The image is uploaded via POST request to:

   /predict/plant

4. **Backend Image Preprocessing**
   - Convert to RGB
   - Resize to 224×224
   - Normalize pixels

5. **Model Prediction**

   MobileNetV2 classifies the disease.

6. **Generate Detailed Output**

   Provides:

   - Disease name
   - Confidence
   - Causes
   - Symptoms
   - Treatment

7. **Display Result on UI**

AgroSense uses clean, easy-to-understand forms that allow users to input important environmental data or upload leaf images. The input flow ensures that all data is validated, preprocessed, and sent correctly to the backend models, leading to accurate irrigation recommendations and disease diagnoses.

Chapter 6

Output Design

Output Design focuses on how the system presents results to the user after processing inputs. Good output design helps users clearly understand irrigation recommendations or plant disease diagnoses without confusion. AgroSense is designed to provide simple, meaningful, and visually clean results through its web interface.

6.1 Output Screens

AgroSense generates two major outputs:

1. Irrigation Recommendation Output Screen

After the user enters temperature, soil moisture, pressure, and altitude values, the system displays:

- **Irrigation Decision:**
  "Irrigation Required" or *"No Irrigation Needed"*
- **Explanation/Advice**
  The system gives short and clear advice based on:
  - Soil moisture
  - Weather conditions (rain forecast, temperature)
  - Atmospheric pressure

    Example output:

    "Soil moisture is low and temperature is high. Irrigation is recommended."
- **Weather Information (if fetched)**
  - Current temperature
  - Pressure
  - Rain indication
  - Estimated altitude

The layout uses:

- Highlighted result box
- Clear text
- Calm colors that match the UI theme

This ensures the user quickly understands the irrigation decision.

2. Plant Disease Detection Output Screen

When a user uploads a leaf image, the output screen shows:

- **Predicted Disease Name**
  Example: *"Tomato Early Blight"*
- **Confidence Percentage**
  Example: *"Confidence: 92%"*
- **Cause**
  Example: *"Caused by the fungus Alternaria solani."*
- **Symptoms**
  Example: *"Brown circular spots with concentric rings on lower leaves."*
- **Treatment Suggestions**
  Example: *"Use mancozeb-based fungicides and remove infected leaves."*
- **Image Preview**
  The uploaded image is displayed so the user knows the correct file was processed.

All outputs appear in a soft, glass-effect card with white text, making it readable over any background.

3. Error Handling Output

If the user enters wrong values or uploads an invalid image, the system displays:

- "Invalid Input. Please enter correct numeric values."
- "Please upload a valid image file."
- "Weather data unavailable, using manual values only."

These messages help the user correct their steps without technical confusion.

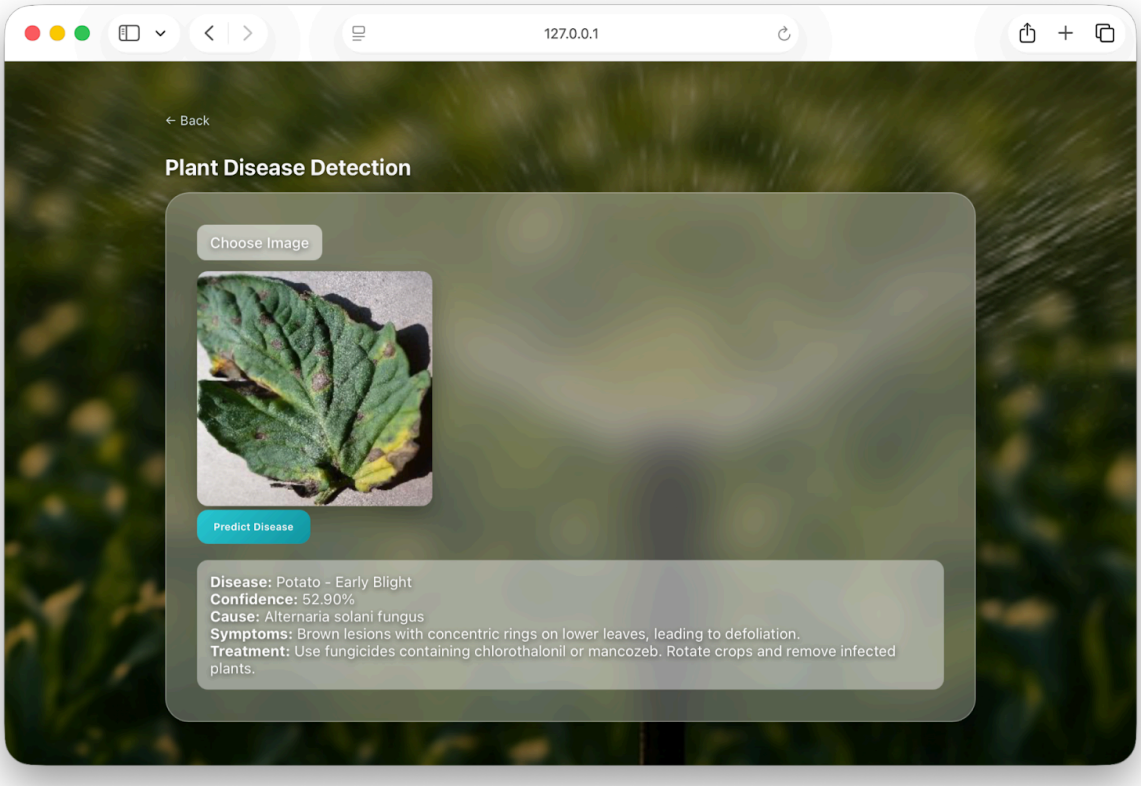

Figure 6.1.1: Example of Output Screen

6.2 Output Flow

This section describes how outputs are generated after backend processing.

Output Flow for Irrigation Prediction

1. User submits environmental input.
2. Backend validates input → If invalid, send error output.
3. Weather API (optional) fetches temperature/pressure.
4. ML Model (Logistic Regression) predicts irrigation requirement.
5. Backend prepares output
    - Irrigation decision
    - Reasoning
    - Weather-based advice

6. Frontend receives JSON response.
7. Formatted result displayed inside a clean result card.
8. User sees a clear recommendation with proper instruction.

Output Flow for Disease Detection

1. User uploads an image.
2. Backend preprocesses image.
3. MobileNetV2 model runs prediction.
4. System generates complete diagnosis
    - Disease name
    - Confidence
    - Cause, symptoms, treatment
5. Frontend displays the output in a structured card.
6. User views diagnosis and suggested actions

The output design of AgroSense focuses on clarity, simplicity, and readability. Both irrigation and disease detection outputs are formatted neatly so users can easily understand the results. Meaningful insights—such as advice, explanations, and confidence levels—help the user take informed decisions about watering and plant health.

Chapter 7

System Testing, Implementation & Maintenance

7.1 Testing Strategy

Testing ensures that AgroSense functions correctly, gives accurate predictions, and provides smooth interaction between the user, backend, and machine learning models. To verify system reliability, different types of testing were performed.

1. Unit Testing

Performed on individual components such as:

- Input validation functions
- Weather API handler
- Image preprocessing pipeline
- ML model loading functions

Example test cases:

- Check if temperature input is numeric
- Ensure image is resized correctly to 224×224
- Verify scaler transforms values without errors

2. Functional Testing

Ensures that every feature performs its expected task.

What was tested:

- Irrigation prediction API (/predict/irrigation)
- Plant disease detection API (/predict/plant)
- Weather API integration (/check_weather)
- Submission and result display in UI

All functions worked correctly with proper responses.

3. Integration Testing

Tests how the full system works when the frontend, backend, ML models, and APIs interact.

Examples:

- Input sent from UI → Flask backend → ML model → Output returns to UI
- Weather API response integrated with irrigation logic

This confirmed smooth data flow across all modules.

4. Model Testing

Ensures the ML models behave correctly.

- Logistic Regression accuracy checked using test dataset
- MobileNetV2 evaluated with sample leaf images
- Incorrect inputs tested to ensure robust error handling

5. User Interface (UI) Testing

Verifies usability and design consistency.

- Form alignment
- Responsiveness on mobile
- Color contrast and text visibility
- Result card readability

6. Error Handling Testing

Tested how the system responds to invalid conditions:

- Missing inputs
- Weather API failure
- Wrong file format (for image uploads)
- Backend exceptions

The system displayed clear and helpful error messages in every case.

7.2 Implementation Strategy

The implementation of AgroSense was planned in small, manageable stages to reduce complexity and ensure successful deployment.

1. Model Development

- Train Logistic Regression for irrigation prediction
- Train MobileNetV2 for plant disease classification
- Save models in pkl and .h5 formats

2. Backend Setup (Flask)

- Create Flask app
- Define routes:
    - /predict/irrigation
    - /predict/plant
    - /check_weather
- Load ML models and scaler
- Implement preprocessing steps
- Add JSON responses for frontend

3. Frontend Development

- Build UI using HTML, CSS, JavaScript
- Create irrigation input form
- Add image upload design
- Use fetch() to call Flask APIs
- Display results inside glass-effect cards

4. Integration

- Connect UI to Flask backend
- Test prediction flow end-to-end
- Debug CORS issues and fix paths

5. Deployment (Local Host)

To run AgroSense:

- python3 -m venv venv
- source venv/bin/activate
- pip install -r requirements.txt
- python app.py

The system runs on:
http://localhost:5001/

6. Documentation

Prepared diagrams, screenshots, and user instructions for final submission.

7.3 Maintenance Strategy

Maintenance ensures that AgroSense continues to run smoothly, even after submission or future updates. The maintenance plan includes:

1. Model Update & Retraining

- Update datasets when new environmental or plant disease data becomes available
- Retrain models to improve accuracy
- Replace .pkl and .h5 model files when needed

2. Dependency Management

- Keep requirements.txt updated

- Avoid outdated versions of TensorFlow, Flask, or scikit-learn
- Test environment compatibility before updates

3. Error & Bug Fixing

- Monitor console logs for backend errors
- Fix problems in input validation or API responses
- Update UI if users report confusion

4. Weather API Monitoring

- Replace API key if it expires
- Update API endpoints if OpenWeather changes
- Add fallback mechanisms for outages

5. UI/UX Improvements

- Ensure the UI remains responsive
- Improve layout based on feedback
- Update images, backgrounds, and styling if required

6. Security Maintenance

- Validate all user inputs
- Prevent uploading harmful files
- Avoid storing unnecessary user data

Chapter 8

Summary and Future Scope

8.1 Summary

The AgroSense Smart Agriculture System was developed to provide farmers and users with a reliable decision-support tool for efficient water management. The system uses machine learning algorithms to predict whether irrigation is required based on environmental parameters such as temperature, soil moisture, atmospheric pressure, and altitude. And also allows users to upload plant leaf images for disease detection using a deep learning model.

Throughout the project, the system was designed with a simple and user-friendly interface that collects inputs, processes them through a Flask backend, and displays the results clearly using a responsive web layout. The Logistic Regression model ensures fast irrigation prediction, while the MobileNetV2 deep learning model enhances the solution by identifying common plant diseases with high accuracy.

The system architecture, diagrams, flowcharts, and testing strategies ensured that all components worked together smoothly. Overall, AgroSense demonstrates how artificial intelligence can improve agricultural practices by making irrigation smarter, reducing water wastage, and helping maintain crop health.

8.2 Future Scope

While AgroSense provides reliable irrigation recommendations and basic disease detection, there are several ways the system can be expanded and improved in the future:

1. IoT Sensor Integration

Integrating real-time sensors such as:

- Soil moisture sensors

- Temperature and humidity sensors
- Water level sensors

This would automate irrigation decisions and make the system even more accurate.

2. Automatic Irrigation Control

Connecting AgroSense to:

- Water pumps
- Sprinkler systems
- Drip irrigation controllers

This would allow the system to automatically turn irrigation ON/OFF without human intervention.

3. Mobile Application

Developing an Android/iOS app would make the system more accessible and easier to use in the field.

4. More Plant Disease Classes

Adding more crops and disease categories to the deep learning model would make the system useful for a wider range of farmers.

5. GPS-Based Weather & Soil Data

Automatically detecting the user's location and fetching:

- Local weather
- Soil data
- Rainfall history

This would remove the need for manual input.

6. Cloud Deployment

Hosting the system on cloud platforms like AWS, GCP, or Azure would:

- Enable remote access
- Support multiple users
- Store historical irrigation and disease data for analytics

7. AI-Based Crop Recommendations

Adding features such as:

- Crop selection advice
- Fertilizer recommendations
- Yield prediction

This would convert AgroSense into a complete smart farming assistant.

References/Bibliography

[1]. TensorFlow. *MobileNetV2 Architecture Documentation*. Available at: https://www.tensorflow.org/api_docs

[2]. Scikit-learn Developers. *Logistic Regression — Scikit-learn Documentation*. Available at: https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

[3]. Kaggle. *PlantVillage Dataset for Plant Disease Classification*. Available at: https://www.kaggle.com/datasets/emmarex/plantdisease

[4]. OpenWeather Ltd. *OpenWeather API — Current Weather Data*. Available at: https://openweathermap.org/api

[5]. Chollet, F. (2017). *Deep Learning with Python*. Manning Publications.

[6]. Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow*. O'Reilly Media.

[7]. Flask Documentation Team. *Flask: Web Application Framework for Python*. Available at: https://flask.palletsprojects.com/

[8]. Parker, J. (2021). *Smart Irrigation Using Machine Learning: A Review of ML Techniques in Agriculture*. International Journal of Agricultural Technology, 17(3), 109–118.

Appendices

Appendix A: Code

```
def check_weather():
    location = request.args.get("location", "default_location")
    url = f"http://api.openweathermap.org/data/2.5/weather?q={location}&appid={WEATHER_API_KEY}&units=metric"
    response = requests.get(url)
    if response.status_code == 200:
        weather_data = response.json()
        rain_forecast = "rain" in weather_data["weather"][0]["description"].lower()
        altitude = weather_data["coord"]["lat"] * 0.1
        return jsonify({
            "rain_expected": rain_forecast,
            "temperature": weather_data["main"]["temp"],
            "pressure": weather_data["main"]["pressure"],
            "altitude": altitude
        })
    return jsonify({"error": "Could not fetch weather data"}), 400


@app.route("/predict/irrigation", methods=["POST"])
def predict_irrigation():
    try:
        data = request.json
```

```
    required_fields = ["temperature", "soil_moisture", "pressure", "altitude"]
    if not all(k in data for k in required_fields):
        return jsonify({"error": "Missing required fields"}), 400

    # Convert input values
    temp = float(data["temperature"])
    soil = float(data["soil_moisture"])
    pressure = float(data["pressure"])
    altitude = float(data["altitude"])
    X = np.array([[temp, pressure, altitude, soil]])
    X_scaled = scaler.transform(X)

    prediction = irrigation_model.predict(X_scaled)[0]

    if prediction in [0, 1]:
        if temp > 35:
            advice = "High temperature detected, irrigation is strongly recommended to prevent heat stress."
        elif pressure < 1000:
            advice = "Low atmospheric pressure detected, indicating possible weather changes. Irrigation is advised."
        else:
            advice = "Soil moisture is low, irrigation is recommended."

    elif prediction in [2, 3]:
```

```
            if temp < 15:
                advice = "Low temperature detected, moisture evaporation is slow. No irrigation needed."
            elif pressure > 1020:
                advice = "High atmospheric pressure detected, reducing evaporation. No irrigation required."
            else:
                advice = "Soil is sufficiently wet, irrigation is not required."
        else:
            advice = "Unexpected model output. Please verify input data."

        return jsonify({"prediction": advice})
    except Exception as e:
        return jsonify({"error": str(e)}), 500
```