VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JNANA SANGAMA" BELAGAVI – 590 018

A Project Report

on

"Multi-Model AI Approach for Crop Advisor and Disease Detector"

Submitted in the partial fulfillment of the requirements for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

(Accredited by NBA, New Delhi, validity up to 30.06.2026)

SUBMITTED BY

SANDEEP S K	**4JD22CS092**
SANJAY G M	**4JD22CS095**
NIRANJAN R M	**4JD22CS077**
VEERESH G S	**4JD22CS118**

UNDER THE GUIDANCE OF

Ms. Rashmi S P

Assistant Professor,

Dept. of CS&E,

Jain Institute of Technology, Davangere

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

JAIN INSTITUTE OF TECHNOLOGY

DAVANGERE – 577003

2025 - 2026

ACKNOWLEDGEMENT

Although a single sentence hardly suffices, we would like to thank almighty God for blessing me with his grace and taking my endeavor to a successful culmination.

We express our gratitude to our guide **Ms. Rashmi S P, Dept. of CS&E, JIT, Davangere,** for her valuable guidance and continual encouragement and assistance throughout Project work**.** We greatly appreciate the freedom and collegial respect. We are grateful to her for discussions about the technical matters and suggestions concerned to our project work.

We extend our sense of gratitude to **Mrs. Manjula P, Project Co-Ordinator, Dept. of CS&E, JIT, Davangere,** for extending support and cooperation which helped us in completion of the project work.

We extend our sense of gratitude to **Dr. Latha B M, Professor & Head, Department of CS&E, JIT, Davangere,** for extending support and cooperation which helped us in completion of the project work.

We express our sincere thanks to **Dr. Ganesh D B, Principal and Director, J.I.T, Davangere,** for extending support and cooperation which helped us in the completion of the project work.

We would like to extend our gratitude to all staff of **Department of Computer Science and Engineering** for the help and support rendered to us. We have benefited a lot from the feedback, suggestions given by them.

We would like to extend our gratitude to all our family members and friends especially for their advice and moral support.

SANDEEP S K	**4JD22CS092**
SANJAY G M	**4JD22CS095**
NIRANJAN R M	**4JD22CS077**
VEERESH G S	**4JD22CS118**

DECLARATION

We, **SANDEEP S K(4JD22CS092), SANJAY G M (4JD22CS095), NIRANJAN R M (4JD22CS077), VEERESH G S (4JD22CS118),** studying in the final semester of Bachelor of Engineering in Computer Science and Engineering at Jain Institute of Technology, Davanagere, hereby declare that this project work entitled **"Multi-Model AI Approach for Crop Advisor and Disease Detector"** which is being submitted by us in the partial fulfilment for the award of the degree of **Bachelor of Engineering** in **Computer Science and Engineering**, from **Visvesvaraya Technological University, Belagavi** is an authentic record of us carried out during the academic year **2025-2026**, under the guidance of **Ms. Rashmi S P,** Department of Computer Science & Engineering, Jain Institute of Technology, Davanagere.

We further undertake that the matter embodied in the dissertation has not been submitted previously for the award of any degree by us to any other university or institution.

Place: Davangere

Date:

SANDEEP S K **4JD22CS092**

SANJAY G M **4JD22CS095**

NIRANJAN R M **4JD22CS077**

VEERESH G S **4JD20CS118**

ABSTRACT

This project aims to develop an AI-powered chatbot designed to assist farmers by providing real-time answers to agricultural queries, recommending crops based on soil parameters, and predicting crop diseases using image analysis. The chatbot will integrate machine learning algorithms and natural language processing to enable seamless communication between farmers and the system. Additionally, by leveraging soil data, the chatbot will offer personalized crop recommendations tailored to the specific conditions of the farmer's land. The image-based disease prediction module will allow farmers to upload pictures of affected plants, and the system will analyse the images to provide accurate diagnoses and treatment suggestions.

Keywords - Data Analytics in Agriculture, Crop Yield Prediction, Crop Selection, Machine Learning, Multiple Linear Regression, Soil Nutrients.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

From ancient period, agriculture is considered as the main and the foremost culture practiced in India. Ancient people cultivate the crops in their own land and so they have been accommodated to their needs. Therefore, the natural crops are cultivated and have been used by many creatures such as human beings, animals and birds. The greenish goods produced in the land which have been taken by the creature leads to a healthy and welfare life. Since the invention of new innovative technologies and techniques the agriculture field is slowly degrading.

1.1 Overview

From the ancient period, agriculture is considered as the main and the foremost culture practiced in India. Ancient people cultivated the crops in their own land and were accommodated to their needs. However, since the invention of new innovative technologies and techniques, the agriculture field is slowly degrading. Due to abundant inventions, people have been concentrating on cultivating artificial products (hybrid products), which leads to an unhealthy life. Nowadays, modern people do not have awareness about the cultivation of the crops at the right time and at the right place. Because of these cultivating techniques, the seasonal climatic conditions are also being changed against the fundamental assets like soil, water, and air, which lead to food insecurity.

By analyzing all these issues and problems like weather, temperature, and several factors, there is no proper solution and technologies to overcome the situation faced by us. In India, there are multiple ways to increase the economical growth in the field of agriculture. There are multiple ways to increase and improve the crop yield and the quality of the crops. Data mining and analysis are useful for predicting crop yield production.

Data analytic (DA) is the process of examining data sets in order to draw conclusions about the information they contain, increasingly with the aid of specialized systems and software. Earlier yield prediction was performed by considering the farmer's experience on a particular field and crop. However, as the conditions change very rapidly, farmers are forced to cultivate more and more crops, but many of them do not have enough knowledge about the new crops.

The proposed system addresses this by increasing farm productivity through understanding and forecasting crop performance in a variety of environmental conditions. The system takes the location of the user as an input to obtain the nutrients of the soil such as Nitrogen, Phosphorous, and Potassium. The processing part also takes into consideration two more datasets: one obtained from the weather department, forecasting the expected weather, and the other being static data related to crop production and demands obtained from various government websites. The system applies machine learning and prediction algorithms like **Multiple Linear Regression** to identify the pattern among data and then process it as per input conditions. This, in turn, will propose the best feasible crops according to given environmental conditions, providing a choice directly to the farmer about which crop to cultivate. As past year production is also taken into account, the prediction will be more accurate.

1.2 Problem statement

The Indian agriculture sector is facing a crisis characterized by the slow degradation of the field and a lack of modern cultivation awareness, which is compounded by rapidly changing environmental conditions leading to food insecurity. Farmers often lack the specialized knowledge and data-driven tools needed to select the right crop for a specific field under changing climate, leading to sub-optimal crop yield and financial loss. The reliance on a farmer's previous experience alone is no longer a reliable method for yield prediction. Therefore, the problem is to develop an efficient, intelligent system that can accurately predict crop yield and recommend the best-suited, most profitable crops to a farmer by dynamically analyzing real-time local soil composition, weather forecasts, and historical crop market data**.**

1.3 Objectives

The main objectives of the Agribot project are:

- To design and implement an intelligent system for accurate crop yield prediction and crop selection.
- To utilize data analytics and machine learning (specifically Multiple Linear Regression) to identify correlations among soil composition, weather, and historical production data.
- To recommend the most profitable and feasible crops to farmers by analyzing their location and the corresponding environmental conditions (soil nutrients and weather forecast).

CHAPTER 2

LITERATURE SURVEY

[1] TITLE: **Geo-Object-Based Soil Organic Matter Mapping Using Machine Learning Algorithms With Multi-Source Geo-Spatial Data**

- ***Authors**:* Tianjun Wu, Jiancheng Luo, Wen Dong, Yingwei Sun, Liegang Xia, and Xuejian Zhang.
- ***Description:*** The study proposes using geo-objects extracted from high-resolution remote sensing images as basic units for soil property mapping. It uses tree-based machine learning algorithms like Random Forests and XGBoost to model the non-linear relationships between soil properties and environmental variables. This method allows for better soil organic matter mapping than interpolation or linear-regression-based methods.

[2] TITLE: IoT based Smart Soil Monitoring System for Agricultural Production

- ***Authors:*** Dr. N. ANANTHI, M.E, Ph. D, Divya J., Divya M., Janani V..
- ***Description:*** This project proposes an embedded system for soil monitoring and irrigation to reduce manual labor. It uses pH, temperature, and humidity sensors to test the soil, suggests appropriate crops, and carries out automatic irrigation when necessary.

[3] TITLE: An Intelligent System for Predicting Thrips Tabaci Linde Pest Population Dynamics Allied To Cotton Crop

- ***Authors:*** Jyothi patil, Dr. A. Govardhan, Dr. V. D. Mytri.
- ***Description:*** The paper addresses insect pests as a prime reason for crop loss. It presents an intelligent system to predict the population dynamics of the *Thrips tabaci Linde* pest on cotton using a Multi-layer perceptron neural network with a back-propagation training algorithm.

[4] TITLE: Improving Crop Productivity Through A Crop Recommendation System Using Ensembling Technique

- ***Authors:*** Nidhi H Kulkarni, Dr. G N Srinivasan, Dr. B M Sagar, Dr. N K Cauvery.
- ***Description:*** This research focuses on developing a crop recommendation system using an ensembling technique to combine predictions from multiple machine learning models (Random Forest, Naive Bayes, and Linear SVM). The system achieved an average classification accuracy of 99.91% based on soil characteristics and climatic conditions.

[5] TITLE: An Effective Method of Controlling the Greenhouse and Crop Monitoring Using GSM

- ***Authors:*** P. S. Asolkar, Prof. Dr. U. S. Bhadade.
- ***Description:*** The paper presents a greenhouse approach using GSM wireless technology to monitor and control critical parameters like temperature, humidity, soil moisture, and Co2 gas. This helps farmers with future prediction and expenditure planning for growing crops.

[6] TITLE: Agricultural Production Output Prediction Using Supervised Machine Learning Techniques

- ***Authors:*** Md. Tahmid Shakoor, Karishma Rahman, Sumaiya Nasrin Rayta, Amitabha Chakrabarty.
- ***Description:*** This research suggests an area-based beneficial crop rank before cultivation. It uses Supervised Machine Learning techniques, specifically Decision Tree Learning-ID3 and K-Nearest Neighbors Regression algorithms, on static historical data of six major crops (rice, potato, jute, wheat)

Paper Title	Author(s)	Year	Methodology	Findings	Limitations
Geo-Object-Based Soil Organic Matter Mapping Using Machine Learning Algorithms With Multi-Source Geo-Spatial Data	Tianjun Wu et al.	2019	CNN for object extraction, Random Forest & XGBoost for regression	Outperforms traditional interpolation methods in SOM mapping	Depends heavily on high-resolution data; model complexity
IoT based Smart Soil Monitoring System for Agricultural Production	Dr. N. Ananthi et al.	2017	IoT with sensors (pH, temp, humidity), Wi-Fi, mobile app	Automates monitoring, suggests crops, manages irrigation	Needs continuous internet access, sensor calibration
An Intelligent System for Predicting Thrips Tabaci Linde Pest Population Dynamics Allied To Cotton Crop	Jyothi Patil et al.	2022	MLP neural network with backpropagation	High prediction accuracy for pest dynamics	Dataset limited to a specific pest and region
Improving Crop Productivity Through A Crop Recommendation System Using Ensembling Technique	Nidhi H Kulkarni et al.	2018	Ensemble ML (Random Forest, Naive Bayes, SVM), majority voting	99.91% accuracy in recommending crops	May not generalize well outside training data region
An Effective Method of Controlling the Greenhouse and Crop Monitoring Using GSM	P. S. Asolkar, Dr. U. S. Bhadade	2016	GSM, sensors for temp, humidity, moisture, CO2	Controls greenhouse environment, supports rural farmers	Limited scalability, reliant on GSM network
Agricultural Production Output Prediction Using Supervised Machine Learning Techniques	Md. Tahmid Shakoor et al.	2017	ID3 Decision Tree, K-NN Regression	Suggests cost-effective crops for regions in Bangladesh	Uses static dataset; may lack adaptability to new trends

Table 2.1 Literature survey

CHAPTER 3

SYSTEM REQUIREMENTS

3.1 Requirement Specification:

This chapter describes the hardware, software, functional, and non-functional requirements necessary for the successful development and deployment of the Agribot / AI-based agricultural support system. The system is implemented using Python due to its simplicity, flexibility, and strong support for Machine Learning applications.

3.1.1 HARDWARE REQUIREMENTS:

Processor	Dual core processor 2.6.0 GHz
RAM	1GB
Hard disk	160 GB
Compact Disk	650 MB
Keyboard	Standard keyboard
Monitor	15inch color monitor

Table 3.1.1: Hardware Requirements

3.1.2 SOFTWARE REQUIREMENTS:

Front End	PYTHON
IDE	PyCharm
Platform	Windows 7

Table 3.1.2: Software Requirements

3.1.3 DESCRIPTION

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse.

Often, programmers fall in love with Python because of the increased productivity it provides. Since there is no compilation step, the edit-test-debug cycle is incredibly fast. Debugging Python programs is easy: a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception. When the program doesn't catch the exception, the interpreter prints a stack trace. A source level debugger allows inspection of local and global variables, evaluation of arbitrary expressions, setting breakpoints, stepping through the code a line at a time, and so on. The debugger is written in Python itself, testifying to Python's introspective power. On the other hand, often the quickest way to debug a program is to add a few print statements to the source: the fast edit-test-debug cycle makes this simple approach very effective.

3.2 Functional Requirements:

Functional requirements define what the system is expected to do. The proposed system must satisfy the following functional requirements:

1. The system shall allow execution of Python-based Machine Learning models for agricultural decision support.
2. The system shall support modular programming using Python modules and packages for easy maintenance and reuse.
3. The system shall process user input data efficiently and generate appropriate outputs.
4. The system shall provide error handling and debugging support through Python’s exception handling mechanism.
5. The system shall allow integration with external libraries and frameworks required for Machine Learning and data analysis.
6. The system shall support graphical user interfaces (GUI) using Python libraries such as PyQt5 or Tkinter if required.
7. The system shall enable rapid development and testing through Python’s interpreted execution model.

3.3 Non-Functional Requirements:

Non-functional requirements describe the quality attributes and constraints of the system. The following non-functional requirements must be met:

1. **Performance**

- The system should execute programs efficiently with minimal response time.
- Python's fast edit-test-debug cycle should support rapid development and testing.

2. **Usability**

- The system should be easy to use and developer-friendly.
- Python's simple and readable syntax should reduce development and maintenance effort.

3. **Reliability**

- The system should handle runtime errors gracefully using exception handling.
- The system should not crash due to invalid inputs; instead, it should display meaningful error messages.

4. **Portability**

- The system should be portable across different platforms such as Windows, Linux, Unix, and macOS without major code modifications.

5. **Scalability**

- The system should support future enhancements such as additional Machine Learning models and datasets.
- Python's extensible nature allows integration with C/C++ for performance-critical modules.

6. **Security**

- The system should ensure secure handling of data and restrict unauthorized access.

CHAPTER 4

SYSTEM DESIGN

4.1 PROPOSED METHODOLOGY

The proposed methodology begins with systematic data collection from agricultural datasets to support intelligent decision-making. A chatbot interface is then developed and integrated with machine learning models trained using the collected data. The trained models are evaluated and enhanced through feature extraction to improve prediction accuracy. Finally, the system undergoes thorough testing and is deployed to provide real-time crop advisory and disease detection support.

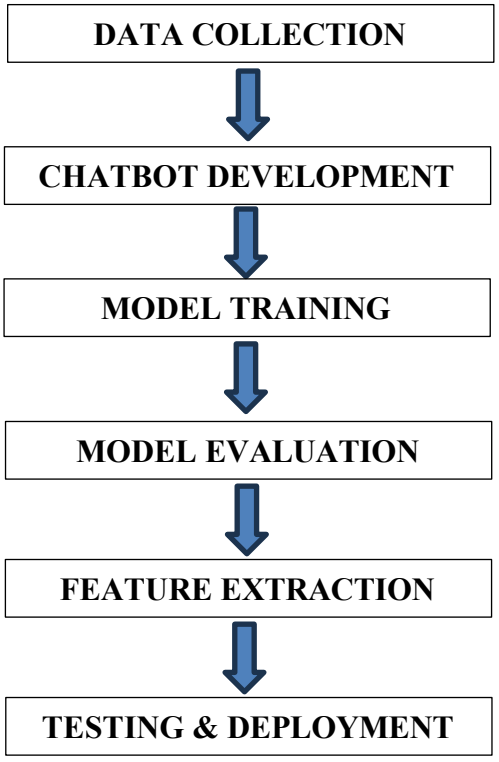

Figure 4.1 Proposed Methodology

1. **DATA COLLECTION:**

- Collect relevant soil data (pH, moisture, nutrient levels) and plant disease image datasets.
- Compile a dataset containing various crop types, their soil requirements, and corresponding optimal conditions.
- Gather image datasets of common crop diseases to train the ResNet model.

2. **CHATBOT DEVELOPMENT:**

- Develop a chatbot using Natural Language Processing (NLP) to understand and respond to farmers queries.
- Integrate the XG Boost model to provide real-time crop recommendations when farmers input soil parameters.

3. **MODEL TRAINING:**

- **XG Boost Model:**

  Train the XG Boost model using soil data to classify and recommend suitable crops based on input parameters like pH, moisture, and nutrients.

- **ResNet Model:**

  Train the ResNet model on the disease image dataset to classify plant diseases based on visual characteristics.

4. **MODEL EVALUATION:**

   The trained models are tested using performance metrics like accuracy and precision to ensure reliability and effectiveness before deployment.

5. **FEATURE EXTRACTION:**

   Important features are extracted from soil data and plant images to improve model performance and enable accurate predictions.

6. **TESTING & DEPLOYMENT:**

   Deploy the chatbot as a web or mobile application accessible to farmers.

   Ensure real-time functionality for answering queries, crop recommendations, and disease detection.

4.2 UML Diagram and Description

The Interaction between the Admin and the System in the data processing workflow. The Admin feeds raw agricultural data and performs input standardization before model execution. The System then trains and tests the data, initializes model parameters, and generates predictions. Finally, the results are analyzed to support accurate crop advisory and disease detection.

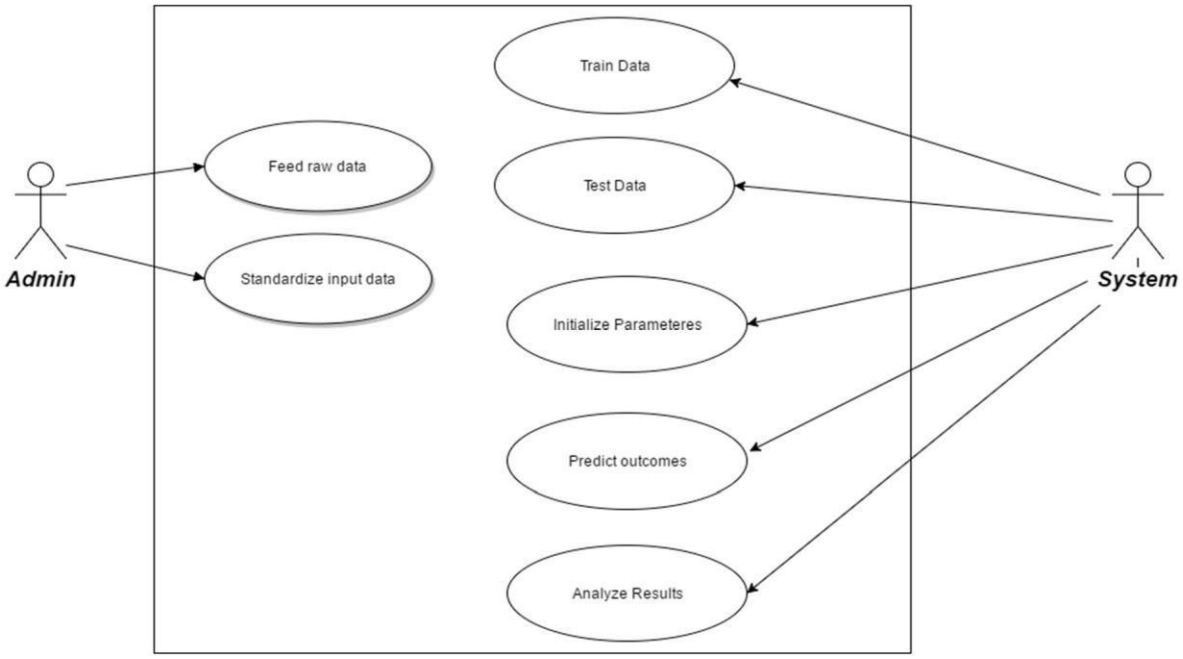

Figure 4.2: UML Diagram

The UML diagram shows the main functions performed by the admin, as well as the system. Each case represents a sub-task that is performed to compute the results. The description of the diagram is as follows:

• The admin standardizes the data available from the surveys, to a machine-readable format, so that it can be operated upon using python and its libraries. The data will then be fed to the system for further processing. The system will now divide this data into two data sets – training and testing data.

• Training labels, provided by the user as a part of the standardization, will be used to form a model which will be used for the testing data.

• The system will be initializing parameters for the model based on the training data.

• The testing data will be classified according to the training results and the parameters initialized in the previous step, by the system.

• The system will now classify outcomes, and analyse results to provide viable solutions to the end-users.

4.3 System Architecture:

The system architecture represents the high-level functioning of AGRIBOT, illustrating how user queries move through different components such as the NLP model, crop recommendation engine, disease detection module, and knowledge base.

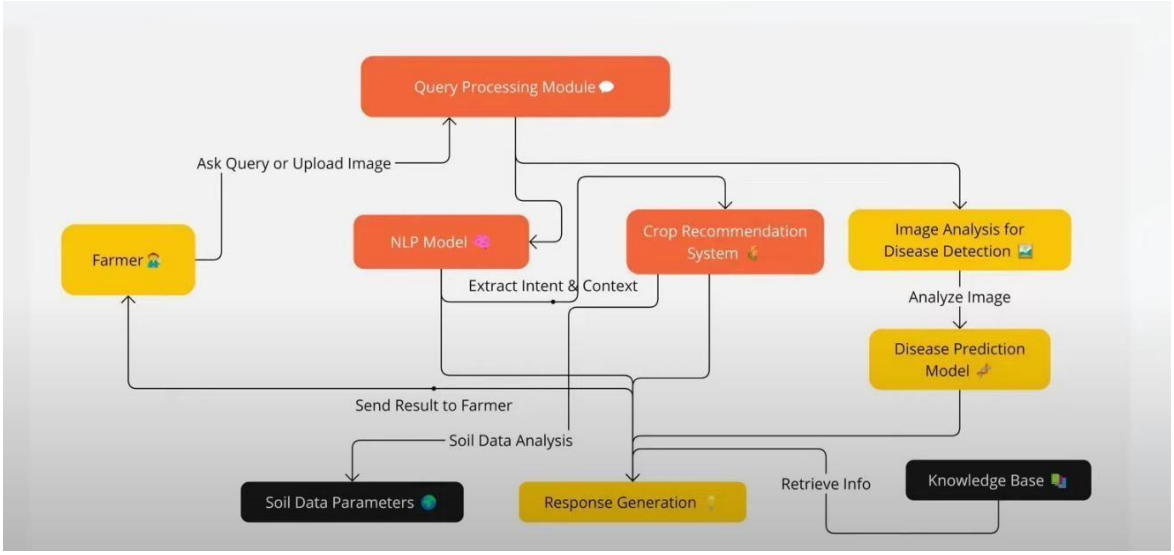

Figure 4.3: System Architecture

The architecture begins when the **farmer** interacts with the system either by typing a query or uploading an image of a plant leaf. The input is first directed to the **Query Processing Module**, which identifies the type of request and forwards it to the appropriate subsystem.

The **NLP Model** plays a critical role in understanding textual queries. It extracts the user's intent and context, enabling the system to distinguish between general informational queries, crop recommendation requests, and disease identification queries. For crop recommendation, the system may request additional soil parameters such as pH, nitrogen, phosphorus, potassium, moisture, and temperature. These parameters are processed by the **Crop Recommendation System**, which uses machine learning models to determine the most suitable crop based on environmental and soil factors.

If the user uploads a plant image for disease detection, the input is sent to the **Image Analysis Module**, which preprocesses the image and forwards it to the **Disease Prediction Model**. This model uses deep learning techniques to classify plant diseases and retrieve corresponding treatment suggestions.

All domain-specific agricultural knowledge—including crop growth conditions, fertilizer requirements, disease remedies, and soil characteristics—is stored in the **Knowledge Base**, which supports both text-based answers and model-based predictions. The **Response Generation Module** synthesizes the outputs from the ML/DL models and the knowledge base to produce a clear, actionable, and user-friendly response for the farmer.

The final output, whether a disease diagnosis, crop recommendation, or informational answer, is sent back to the farmer through the chatbot interface. This architecture ensures smooth integration of advanced technologies such as NLP, machine learning, and image processing to deliver intelligent agricultural assistance.

4.4 Data Flow Diagram:

The Data Flow Diagram (DFD) illustrates the flow of information within the AGRIBOT system and highlights how raw agricultural data is transformed into meaningful outputs. It explains the sequence of operations starting from data collection to model training, testing, analysis, and final result generation. The DFD provides a clear understanding of the internal processing pipeline used to support accurate crop recommendations and disease predictions.

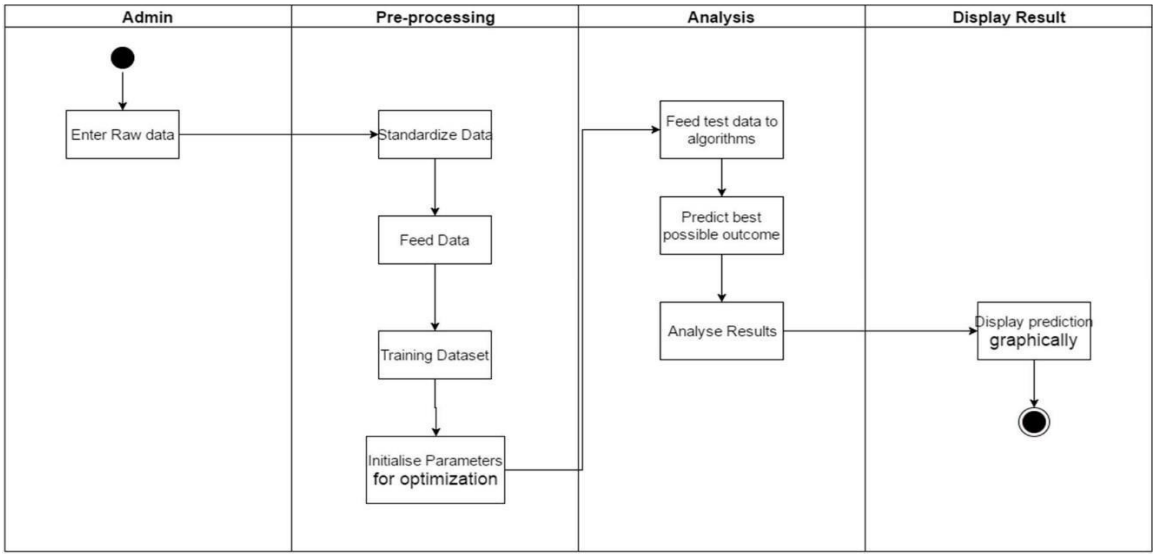

Figure 4.4: Data Flow Diagram

The process begins with the **Admin**, who is responsible for entering raw soil data collected from agricultural fields. This raw data may include essential parameters such as soil moisture, pH, temperature, nutrient levels, and other environmental factors Once the data is entered, it undergoes a **standardization process**, where inconsistencies, missing values, and noise are removed. This step ensures that the dataset is clean, structured, and ready for analysis.

After preprocessing, the cleaned data is forwarded to the **Training Dataset Module**, where the data is organized for machine learning model development. The system then proceeds to **initialize parameters for optimization**, ensuring that the learning algorithms operate efficiently during training. These parameters play a critical role in tuning the performance of the prediction models.

The **Analysis Phase** includes feeding test or unseen data into the algorithmic modules. Machine learning algorithms evaluate the input parameters and predict outcomes such as the most suitable crop for a particular soil type or potential plant diseases. The algorithms may also identify environmental issues like drought-affected regions depending on the data provided.

Once the prediction or analysis is complete, the results are forwarded to the **Display Module**, where outputs are presented visually to the user. Graphs, charts, or textual summaries are generated to help farmers clearly understand the analysis results. This ensures that the system not only computes accurate predictions but also communicates them in a user-friendly manner.

4.5 System Workflow Diagram:

The System Workflow Diagram provides a detailed overview of how AGRIBOT processes agricultural data to generate meaningful predictions such as crop recommendations, drought-affected region identification, and overall crop suitability analysis. This workflow highlights the relationship between training data, system operations, machine learning algorithms, and the final output delivered to farmers.

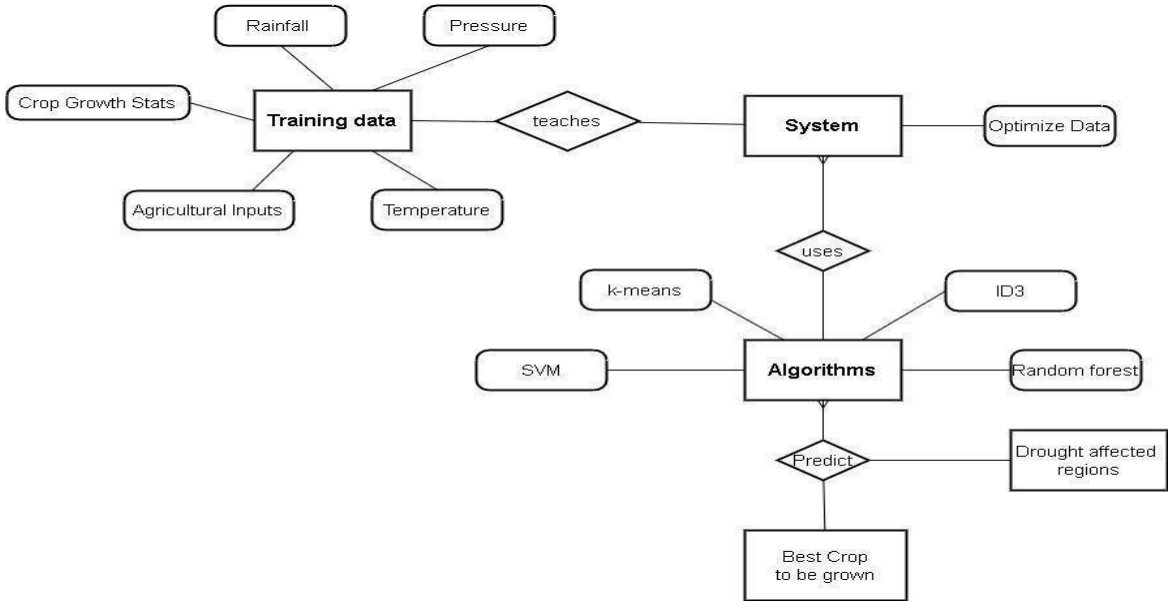

Figure 4.5: System Workflow Diagram

The workflow begins with the collection of diverse **training data**, which forms the foundational input for prediction models. The training dataset includes key agricultural and environmental factors such as:

- **Rainfall**
- **Pressure**
- **Temperature**
- **Agricultural Inputs (NPK values, fertilizers, etc.)**
- **Crop Growth Statistics**

These parameters collectively represent the environmental and soil conditions that influence crop performance. The system uses this dataset to learn and optimize prediction accuracy.

Several algorithms are utilized within AGRIBOT to support multi-dimensional decision making:

- **K-Means** – for clustering regions based on soil and climate characteristics.
- **SVM (Support Vector Machine)** – for classification of suitable vs. unsuitable crop conditions.
- **ID3 Decision Tree** – for rule-based classification and hierarchical decision-making.
- **Random Forest** – for high-accuracy predictions and handling non-linear agricultural data.

These algorithms collectively contribute to building a robust predictive model capable of handling variations in environmental conditions.

Once the algorithms process the data, the system is able to:

- Identify **drought-affected regions**
- Predict the **best crop to be grown** in a given area

The final output helps farmers make informed decisions, reduce risk, and maximize crop productivity based on scientifically analyzed data. This workflow demonstrates the seamless integration of environmental parameters, machine learning techniques, and predictive modeling in AGRIBOT.

4.6 Use Case Diagram:

The **Use Case Diagram** illustrates the major interactions between the system and external actors involved in the AGRIBOT application. This diagram provides a high-level overview of how farmers and administrators utilize the system to perform core operations such as crop recommendation, plant disease prediction, and query resolution. The use case model helps in identifying the functional requirements of the system and clearly defines how each user interacts with the application.

In AGRIBOT, two main actors interact with the system:

1. **Farmer**

The farmer is the primary user who interacts with the chatbot to obtain:

- Crop recommendations based on soil parameters.
- Plant disease predictions by uploading images.
- Answers to general agricultural queries.
- Guidance and solutions generated from the knowledge base.

2. **Admin**

The admin oversees system functionality by:

- Updating and managing the knowledge base
- Monitoring chatbot performance
- Ensuring correct functioning of prediction models
- Maintaining and refining system rules

The use case diagram outlines the relationships between these actors and key system functionalities. It depicts how the farmer initiates interactions such as uploading plant images or submitting soil details, while the admin focuses on maintaining and updating backend components.

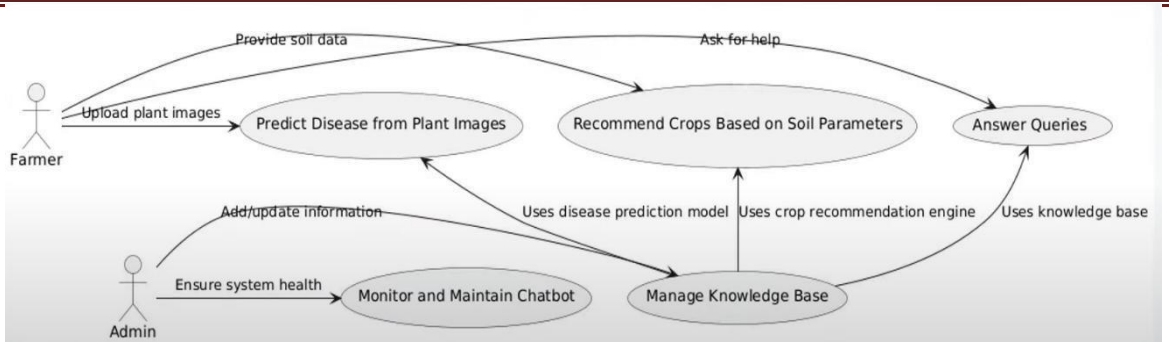

Figure 4.6: Use Case Diagram

This diagram appropriately shows:

- Farmer interacting with **Predict Disease**, **Recommend Crops**, and **Answer Queries.**
- Admin maintaining the **Knowledge Base** and **Chatbot System**.
- Use case dependencies on prediction models and knowledge retrieval processes.

4.6.1 Explanation of Major Use Cases

✓ **Predict Disease from Plant Images**

- Farmer uploads an image of a diseased plant.
- System analyzes image using a machine learning model.
- System returns predicted disease and treatment suggestions.

✓ **Recommend Crops Based on Soil Parameters**

- Farmer provides soil data such as pH, moisture, and nutrient content.
- System analyzes soil parameters.
- System recommends the most suitable crops for cultivation.

✓ **Answer General Queries**

- Farmer asks questions related to agriculture, irrigation, soil health, pests, fertilizers, etc.
- System retrieves responses using NLP and the knowledge base.

✓ **Manage Knowledge Base *(Admin)***

- Admin updates agricultural information, and domain knowledge.
- Ensures accurate and updated responses for chatbot queries.

✓ **Monitor and Maintain Chatbot *(Admin)***

- Admin ensures system stability.
- Updates models, handles issues, and reviews logs.

4.7 Class Diagram:

The **Class Diagram** represents the static structure of the AGRIBOT system, depicting the major system classes, their attributes, responsibilities, and relationships. It provides a blueprint for the object-oriented design of the system by modeling how the software components interact with each other.

The AGRIBOT system consists of four primary classes: **Chatbot**, **Farmer**, **Query Handler**, **Crop Recommender**, and **Disease Predictor**. Each class is designed with a clear purpose, ensuring modularity, reusability, and scalability of the system. Together, these classes form the core intelligence and interaction flow of the chatbot-based agricultural support system.

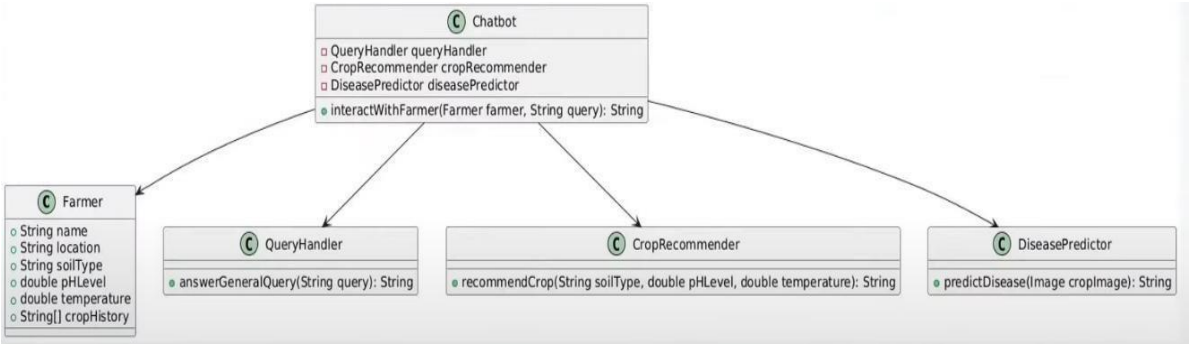

Figure 4.7: Class Diagram

4.7.1 Description of Classes

1. Farmer Class

The Farmer class represents the end-user interacting with the system.

Attributes:

- String name – Farmer's name
- String location – Geographic location
- String soil Type – Soil type of the land
- double pH Level – Soil pH value

- double temperature – Environmental temperature
- String [] crop History – Previously grown crops

Role:

Provides the input parameters to the chatbot and receives system-generated recommendations and predictions.

2. Chatbot Class

The *Chatbot* acts as the central controller of the system, coordinating between different modules.

Attributes:

- Query Handler query Handler
- Crop Recommender crop Recommender
- Disease Predictor disease Predictor

Methods:

- Interact With Farmer (Farmer farmer, String query): String

Role:

Handles user queries, routes them to appropriate modules, and generates responses.

3. Query Handler Class

The *Query Handler* is responsible for processing general agricultural queries that do not require deep ML-based analysis.

Methods:

- Answer General Query(String query): String

Role:

Uses the knowledge base to provide instant answers, improving chatbot responsiveness.

4. Crop Recommender Class

This class contains the logic for recommending the most suitable crops based on soil parameters.

Methods:

- Recommend Crop(String soil Type, double pH Level, double temperature): String

Role:

Applies machine learning or rule-based logic to determine crop suitability.

5. Disease Predictor Class

This class predicts plant diseases based on uploaded images.

Methods:

- Predict Disease(Image crop Image): String

Role:

Uses image-processing and ML models (e.g., CNN/ResNet) to classify diseases and suggest corrective actions.

Relationship Between Classes

- The **Chatbot** class aggregates the helper classes (**Query Handler**,

Crop Recommender, **Disease Predictor**) to divide responsibilities across modules.

- The **Farmer** interacts directly with the **Chatbot**, providing input parameters or images.
- The **Chatbot** delegates:
    - Query handling → **Query Handler**
    - Crop recommendation → **Crop Recommender**
    - Disease detection → **Disease Predictor**

This modular structure enhances maintainability and enables future expansion, such as adding new analysis models or additional agricultural advisory features.

4.8 Sequence Diagram:

The **Sequence Diagram** illustrates the dynamic behavior of the AGRIBOT system by showing how different components interact with each other over time to fulfill user requests. It models the step-by-step flow of messages between actors and system modules.

1. **General Query Handling**

2. **Crop Recommendation**

3. **Disease Prediction from Plant Images**

This diagram helps visualize the runtime communication among the **Farmer**, **Chatbot**, **Crop Recommendation System**, **Disease Prediction Model.**

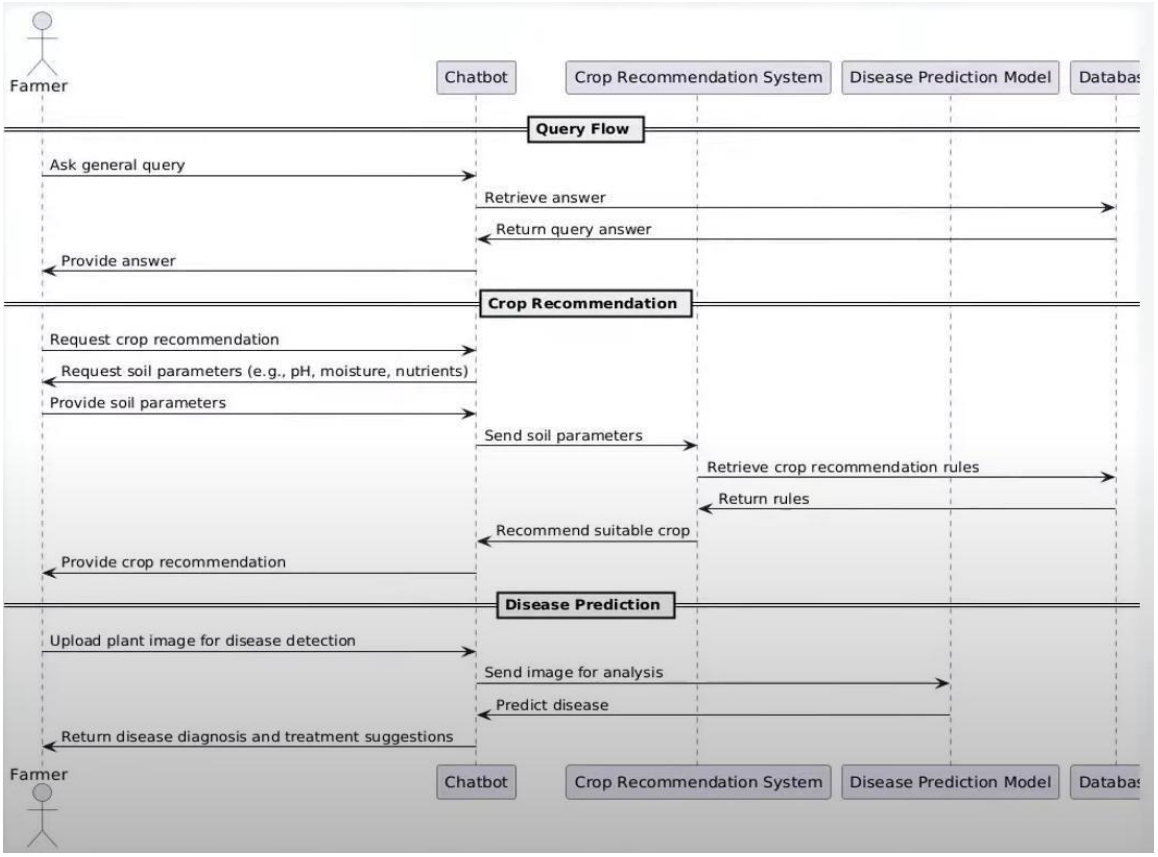

Figure 4.8: Sequence Diagram

4.8.1 Description of Sequence Flow

1. General Query Flow

When the farmer asks a normal agricultural question (e.g., *"What are good practices for soil health?"*), the following steps occur:

1. **Farmer → Chatbot:** Sends a general query.

2. **Chatbot → Knowledge Base / Internal Logic:** Retrieves an appropriate answer.

3. **Knowledge Base → Chatbot:** Returns the prepared answer.

4. **Chatbot → Farmer:** Sends the final answer.

This ensures fast responses for informational queries without invoking machine learning models.

2. Crop Recommendation Flow

When the farmer wants to know **which crop is best suited for their soil**, the following interaction occurs:

i **Farmer → Chatbot:** Requests crop recommendation.

ii **Chatbot → Farmer:** Requests soil parameters (pH, moisture, NPK, temperature).

iii **Farmer → Chatbot:** Submits soil data.

iv **Chatbot → Crop Recommendation System:** Sends parameters for ML-based analysis.

v **Crop Recommendation System → Database:** Fetches crop suitability rules/models.

vi **Database → Crop Recommendation System:** Returns matching rule set.

vii **Crop Recommendation System → Chatbot:** Suggests the best crop(s).

viii **Chatbot → Farmer:** Provides crop recommendation.

This provides a personalized and data-driven crop suggestion to improve yield.

3. Disease Prediction Flow

When the farmer uploads an image of a diseased crop:

i **Farmer → Chatbot:** Uploads plant leaf or crop image.

ii **Chatbot → Disease Prediction Model:** Sends the image for analysis.

iii **Disease Prediction Model:** Processes image using ML (e.g., CNN/ResNet).

iv **Disease Prediction Model → Database:** Retrieves disease data and treatments.

v **Database → Disease Prediction Model:** Returns disease information.

vi **Disease Prediction Model → Chatbot:** Sends diagnosis + remedies.

vii **Chatbot → Farmer:** Displays disease name, severity, cause, and treatment suggestions.

This helps farmers quickly identify crop diseases and take preventive measures.

Importance of the Sequence Diagram

The sequence diagram is crucial because it:

- Demonstrates the **runtime workflow** of the AGRIBOT system.
- Shows **clear communication paths** between AI modules.
- Ensures understanding of **temporal ordering** of operations.
- Helps developers maintain modular and structured code.

Guides implementation of system logic in the final product.

4.9 Activity Diagram:

The Activity Diagram models the detailed runtime behavior of the AGRIBOT system from the perspective of user interactions and internal decision-making. It captures the step-bystep flow of control for the most common user scenarios — asking an informational query, requesting a crop recommendation, or uploading a plant image for disease detection. The diagram clarifies how inputs are evaluated, how the system decides which module to activate, and how outputs are generated and returned to the farmer.

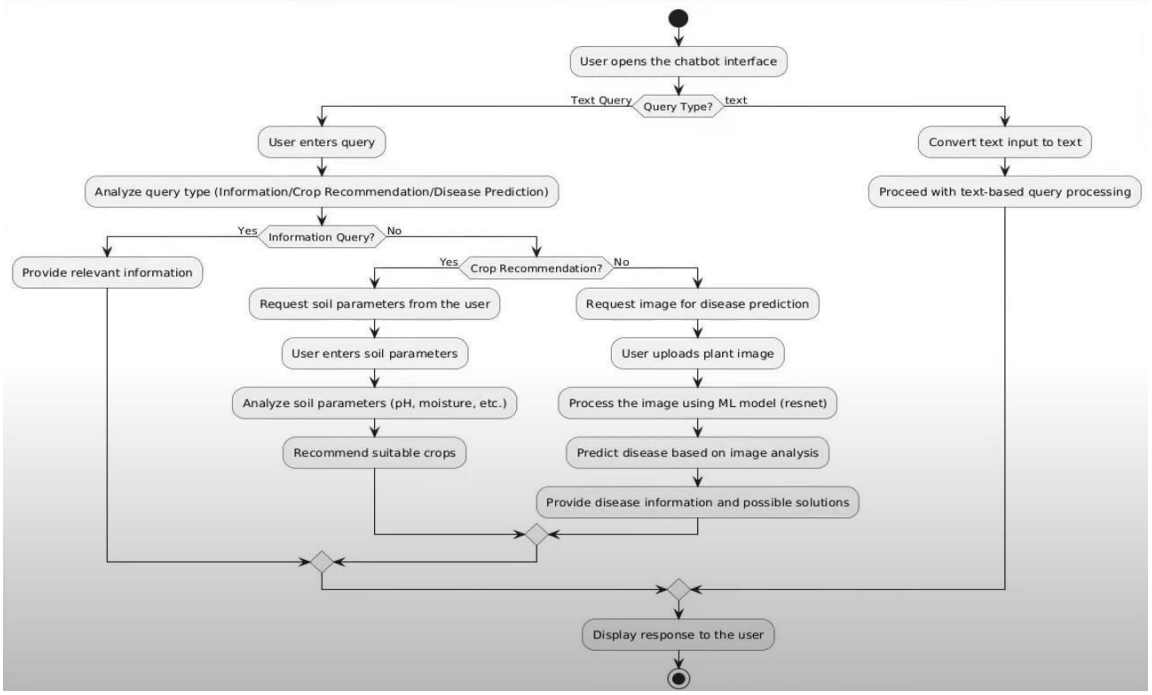

Figure 4.9: Activity Diagram

The activity starts when the farmer opens the chatbot interface and submits input (text query or image). The system first performs **input classification** (via the NLP module) to determine the intent: general information, crop recommendation, or disease prediction. Based on this decision, the workflow branches into one of three main sub-flows. Each subflow contains its own sequence of activities, validations, and interactions with backend modules (crop recommender, disease predictor, knowledge base). After the selected subflow completes, results are aggregated by the response generation module and presented to the farmer. The diagram ends when the farmer receives the response and optionally provides feedback.

CHAPTER 5

MODULE DESCRIPTION

The AGRIBOT system is divided into several interconnected modules, each performing a crucial function in the end-to-end process of predicting crops, identifying plant diseases, analyzing soil data, and communicating results to the farmer via an intelligent chatbot. These modules ensure that the system operates accurately, efficiently, and reliably, making it easier for farmers to make informed agricultural decisions. This chapter provides an extended and detailed explanation of each module used in the AGRIBOT system.

5.1 Soil Testing Module

Soil testing forms the foundational step in agricultural analysis. The AGRIBOT system relies son the quality and accuracy of soil parameters, as soil health directly influences crop productivity. This module determines the physical, chemical, and environmental properties of soil samples.

The soil testing process involves measuring key nutrient values such as Nitrogen (N), Phosphorus (P), Potassium (K), pH, moisture, temperature, and humidity. These factors determine soil fertility and suitability for specific crops. The soil testing module uses either IoT-based sensors or laboratory-tested datasets to collect these values. Sensors like soil moisture probes and pH meters are integrated with microcontrollers to capture real-time soil readings.

The soil testing module helps:

- Identify nutrient deficiencies or toxicities
- Understand soil salinity levels
- Detect soil moisture imbalance
- Assess soil’s ability to support specific crops
- Provide a scientific basis for fertilizer recommendations

The output from this module is crucial for the crop recommendation and fertilizer prediction processes. By analyzing soil characteristics accurately, AGRIBOT ensures that farmers receive precise agricultural advice tailored to their field conditions.

5.2 Soil Data Collection Module

The soil data collection module is responsible for gathering accurate soil readings from various sources. In AGRIBOT, data can be collected in two ways:

1. Sensor-Based Data Collection

Microcontrollers such as Arduino or ESP32 are used to interface with sensors that detect soil parameters continuously. The collected data is transmitted wirelessly using communication modules like RF24L01 or Wi-Fi modules. Sensor-based data ensures realtime monitoring and helps farmers track soil changes throughout the day.

2. Dataset-Based Collection

Soil survey datasets collected from agricultural universities, government agencies, or field sampling studies are used for building ML models. These datasets contain detailed nutrient values, soil type information, and location-based readings.

The raw data collected is often extensive and includes:

- Chemical tests (NPK values, pH, salinity)
- Physical tests (moisture content, texture)
- Environmental parameters (temperature, humidity)
- Historical crop performance

Before being used in the machine learning model, this data must undergo preprocessing to ensure consistency. The accuracy of AGRIBOT's predictions depends heavily on the quality of the data collected. Therefore, this module plays a vital role in ensuring that only clean, relevant, and representative data is passed on to the next stage.

5.3 Data Preprocessing Module

Raw soil data often contains noise, missing values, inconsistencies, and outliers. To ensure the ML/DL models perform accurately, the dataset must be cleaned and transformed. The data preprocessing module performs this task by applying systematic data cleansing and transformation operations.

5.3.1 Steps in Data Preprocessing

1. **Importing Libraries:**

   Libraries like NumPy, Pandas, scikit-learn, and Matplotlib are imported for data manipulation, visualization, and modeling.

2. **Importing the Dataset:**

   Soil datasets collected from sensors or external sources are loaded into dataframes.

3. **Handling Missing Values:**

   Missing entries are replaced using imputation techniques such as mean, median, or KNN imputation.

4. **Encoding Categorical Data:**

   Some attributes may be non-numeric. One-hot encoding or label encoding is used to convert them into machine-readable formats.

5. **Splitting the Dataset:**

   The dataset is divided into training and testing sets (commonly 80:20 ratio).

- Training set → used for model learning
- Testing set → used to evaluate performance

6. **Feature Scaling:**

   Normalization or standardization is applied to ensure uniformity in data distribution. Algorithms like SVM and KNN perform better with scaled data.

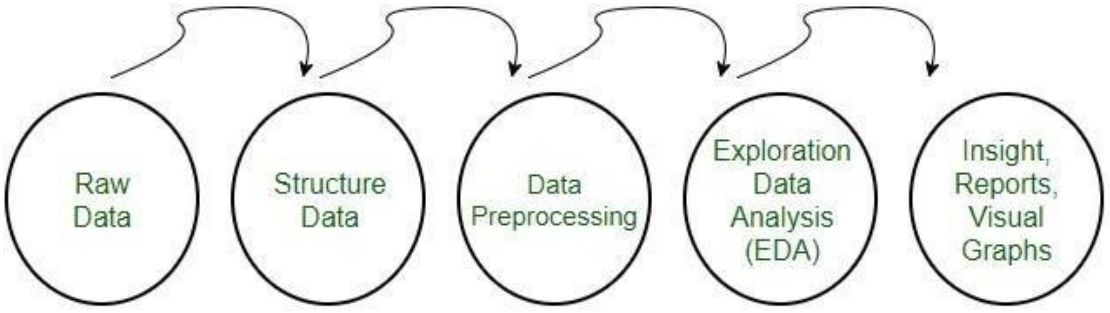

Figure 5.3.1 Raw data analysis and exploration steps

5.3.2 Need for Data Preprocessing

- Ensures dataset compatibility with ML algorithms.
- Increases model accuracy and reduces computation time.
- Eliminates noise and outliers which can distort predictions.
- Creates a clean dataset suitable for both machine learning and deep learning models.
- Supports comparative evaluation of multiple algorithms using the same dataset.

Data preprocessing is one of the most important steps because the performance of AGRIBOT's crop recommendation engine depends directly on the quality of the processed dataset.

5.4 Regression / Classification Module

This module focuses on using Machine Learning algorithms to predict suitable crops based on the soil's nutrient values and environmental conditions.

Different models are trained and compared to determine which algorithm performs best.

The commonly used prediction models include:

1. Linear Regression

This algorithm identifies the linear relationship between input variables (soil nutrients) and output (crop suitability or yield). It fits a best-fit line based on parameters α (intercept) and β (coefficient). It is mainly used for continuous fertility or yield prediction.

2. Random Forest

A powerful ensemble model that creates multiple decision trees and combines their outputs to generate highly accurate predictions. It identifies key features affecting soil fertility and crop selection.

3. XG Boost (Extreme Gradient Boosting)

Known for speed and high performance, XGBoost builds optimized boosted trees to provide highly accurate classification of suitable crops.

4. Support Vector Machines (SVM)

Useful for classifying soil types or predicting specific crop categories using hyperplanebased classification.

This module outputs:

- The most suitable crop for the given soil
- Soil fertility rating
- Prediction accuracy of various models

AGRIBOT selects the best-performing model to give farmers the most reliable recommendation.

5.5 Pattern Prediction Module

Pattern prediction plays a crucial role in identifying recurring trends and establishing relationships within soil datasets and historical crop performance. This module uses sequence-based learning and pattern recognition techniques to detect:

- Soil nutrient variation over time
- Seasonal yield trends
- Early symptoms of crop disease patterns
- Changes in environmental parameters **7.5.1 Training and Testing Process**

The dataset is separated into:

- **Training Set (80–85%)** – Used to train the prediction models.
- **Testing Set (15–20%)** – Used to validate model performance.

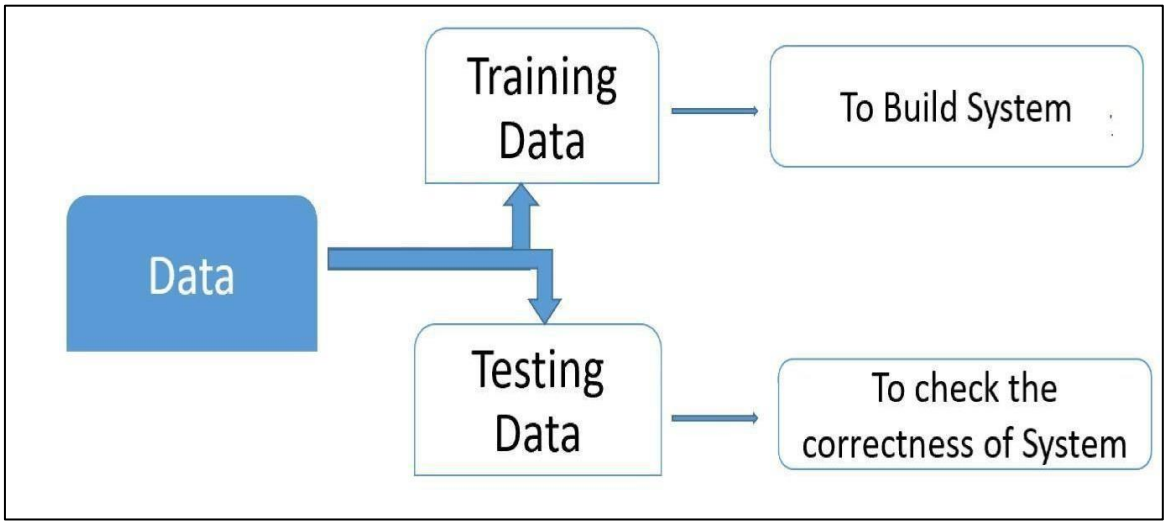

Figure 5.5.1 Data Training and testing with the pattern-based recognition

During training, the model learns:

- Hidden correlations between soil characteristics
- How nutrient changes affect crop yield
- Seasonal and environmental variations

Pattern recognition is especially useful in disease prediction and yield forecasting. By analyzing frequent patterns and trends, AGRIBOT can generate recommendations that align with real-world field conditions.

CHAPTER 6

SYSTEM IMPLEMENTATION, TESTING AND MAINTENANCE

6.1 System Implementation

System implementation is the process of converting a designed system into an operational one. It includes preparing hardware and software environments, coding modules, integrating system components, training users, and ensuring the system is ready for realworld usage. Implementation transforms the design into a functioning product that meets performance, usability, and reliability requirements.

In the AGRIBOT system, implementation involves:

- Building the NLP-based chatbot
- Developing ML models for crop recommendation
- Training CNN/Res Net models for disease detection
- Integrating user interface components
- Creating APIs and backend logic using Python
- Testing and deploying the complete solution

Implementation ensures that each module—soil analysis, preprocessing, prediction, disease detection, and chatbot response generation—is developed based on design specifications and functions correctly within the integrated system.

6.1.1 System Design Phases

System design can be divided into three major levels:

1. **Conceptual Design – "What the system should do"**

This phase outlines the functionalities of AGRIBOT, such as:

- Responding to farmers' queries
- Recommending suitable crops
- Identifying plant diseases using images

- Providing fertilizer suggestions
- Handling soil data inputs

It focuses on system goals, data flow, user expectations, and high-level features.

2. Logical Design – "How the system should appear to the user"

This design translates user requirements into structured modules:

- Chatbot interaction flow
- Input/output interfaces
- Data structures for soil and crop data
- Logical workflow of crop prediction and disease identification

User interaction screens and data relationships are finalized in this phase.

3. Physical Design – "How the system should be built"

This phase defines the technical implementation of AGRIBOT:

- Python environment setup (Flask backend, ML/DL libraries)
- Dataset storage and preprocessing methods
- CNN model training architecture
- Integration of NLP tokenizers, response generators
- Model deployment setup

The physical design ensures the system is built using appropriate tools, configurations, and development environments.

6.1.2 Implementation Strategy

The following activities were performed as part of AGRIBOT's implementation process:

1. Define Implementation Strategy

This stage involved choosing the right tools and environment:

- Python, scikit-learn, TensorFlow, OpenCV for model development
- Data preprocessing using Pandas and NumPy

- IDE selection: PyCharm / VS Code
- API development using Flask
- Storage and retrieval of datasets

The strategy also defined coding standards, testing methods, and deployment procedures.

2. **Realization of System Elements** Each component was implemented as follows:

- **ML Models:**
  Soil datasets were trained using Random Forest, XGBoost, and Linear Regression to identify the best crop prediction model.
- **DL Disease Detection Model:**
  A CNN/ResNet architecture was trained on plant leaf images to classify diseases.
- **Chatbot:**
  NLP pipeline implemented using tokenization, intent detection, and response mapping.
- **User Interface:**
  Interfaces developed for farmers to upload leaf images, enter soil values, and chat with the bot.

3. **Evidence of Compliance**

The following verification methods were applied:

- Unit testing of functions
- Accuracy measurement of ML/DL models
- Validation against test datasets
- Peer review for chatbot responses
- Performance testing of data preprocessing

4. **Packaging, Storing, and Deployment**

The final system components were packaged as Python modules and Flask-based APIs. The ML and DL models were saved in serialized formats (pickle/H5 files) for easy loading during runtime.

6.2 System Testing

System testing is performed to ensure the AGRIBOT application works according to the defined requirements and provides accurate, reliable, and consistent results. Testing validates both the functional behavior and the performance of ML/DL models.

The objective of testing is to detect errors early, ensure correctness, evaluate usability, and guarantee that the application meets all expected outcomes before deployment.

6.2.1 Types of Tests

1. Unit Testing

Each individual component of AGRIBOT—such as the soil data preprocessing function, the chatbot response generator, and the CNN prediction module—was tested separately.

Unit testing focuses on:

- Checking individual methods for correct outputs
- Ensuring ML model prediction functions produce valid results
- Validating data cleaning and data transformation logic

All functions were checked for correct inputs, outputs, and error handling.

2. Integration Testing

After unit testing, modules were connected and tested together. Examples in AGRIBOT:

- Soil testing → preprocessing → ML prediction pipeline
- Image upload → CNN inference → disease output generation
- Chatbot query → NLP processing → response generation

Integration testing ensured smooth communication between components without data loss or inconsistency.

3. System Testing

This testing validated the complete AGRIBOT application.

System testing checks:

- End-to-end working of the chatbot
- Crop recommendation accuracy with real soil samples
- Disease detection using test leaf images
- User interface functionality
- Backend API response time

System testing ensured the entire AGRIBOT system is functional and adheres to project specifications.

4. White Box Testing

Performed on the internal functions of AGRIBOT:

- Model training loops
- Data transformation logic
- Feature extraction and NLP processes

White box testing validated internal workflow and algorithm behavior.

5. Black Box Testing

Performed from the user perspective without looking at internal code.

For AGRIBOT, this included:

- Entering random soil values and verifying crop suggestions
- Uploading leaf images and checking disease classification
- Sending natural-language questions to the chatbot

Tester only checks inputs and outputs.

6. Acceptance Testing

This testing ensures that the final system meets user expectations.

Performed by:

- Farmers
- Students
- Faculty members
- Domain experts

Feedback regarding usability, accuracy, clarity of outputs, and speed of response was collected.

7. Functional Testing

Validates whether each functional requirement is implemented correctly.

Tests included:

- Soil values accepted properly

- Predictions shown correctly
- Error messages shown for invalid inputs
- Chatbot responses accurate and relevant

6.3 System Maintenance

Maintenance is essential to ensure that the AGRIBOT system continues to operate efficiently after deployment. As datasets evolve, new diseases emerge, and farming practices change, the system must be updated regularly.

6.3.1 Types of Maintenance Performed:

1. Corrective Maintenance

Fixing bugs or errors detected after deployment, such as incorrect predictions or UI glitches.

2. Adaptive Maintenance

Modifying the system to work with new hardware, software environments, or updated libraries (Python, TensorFlow, etc.).

3. Perfective Maintenance

Enhancing system performance and reliability by improving ML model accuracy, updating datasets, and optimizing preprocessing pipelines.

4. Preventive Maintenance

Regular checks to ensure no model or module degrades over time.
Includes:

- Updating knowledge base
- Refreshing training dataset
- Re-training ML models periodically to prevent drift

Maintenance ensures long-term system stability, reliability, and scalability.

6.4 Code

```
# Importing essential libraries and modules

from flask import Flask, render_template, request, Markup

import numpy as np

import pandas as pd

from utils.disease import disease_dic

from utils.fertilizer import fertilizer_dic

import requests

import config

import pickle

import io

import torch

from torchvision import transforms

from PIL import Image

from utils.model import ResNet9

from flask import Flask, render_template, request, jsonify

from chat import get_response

#
==========================================================================
======================

# ----------------------------LOADING THE TRAINED MODELS  ----------------------------------------------------

# Loading plant disease classification mode

disease_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']
disease_model_path = 'models/plant_disease_model.pth'
disease_model = ResNet9(3, len(disease_classes))
disease_model.load_state_dict(torch.load(
    disease_model_path, map_location=torch.device('cpu')))
disease_model.eval()
# Loading crop recommendation model
crop_recommendation_model_path = 'models/RandomForest.pkl'
crop_recommendation_model = pickle.load(
    open(crop_recommendation_model_path,  'rb'))
#
================================================================================
================
# Custom functions for calculations
def weather_fetch(city_name):
    """
    Fetch and return the temperature and humidity of a city using OpenWeather One Call API 3.0
```

```
    :param city_name: str
    :return: (temperature in Celsius, humidity in %), or None if not found
    """
    api_key = config.weather_api_key
    # Step 1: Get latitude and longitude using Geocoding API
    geo_url = f"http://api.openweathermap.org/geo/1.0/direct?q={city_name}&limit=1&appid={api_key}"
    geo_response = requests.get(geo_url)
    geo_data = geo_response.json()
    if not geo_data:
        return None  # City not found
    lat = ("13.08")
    lon = ("80.27")
    # Step 2: Get weather data using One Call API 3.0
    weather_url = f"https://api.openweathermap.org/data/3.0/onecall?lat={lat}&lon={lon}&exclude=minutely,hourly,daily,alerts&appid={api_key}"
    weather_response = requests.get(weather_url)
    weather_data = weather_response.json()
    try:
        temperature = (32)  # Kelvin to Celsius
        humidity = (65)
        return temperature, humidity
    except KeyError:
        return None
def predict_image(img, model=disease_model):
```

```
    """
    Transforms image to tensor and predicts disease label
    :params: image
    :return: prediction (string)
    """
    transform = transforms.Compose([
        transforms.Resize(256),
        transforms.ToTensor(),
    ])
    image = Image.open(io.BytesIO(img))
    img_t = transform(image)
    img_u = torch.unsqueeze(img_t, 0)
    # Get predictions from model
    yb = model(img_u)
    # Pick index with highest probability
    _, preds = torch.max(yb, dim=1)
    prediction = disease_classes[preds[0].item()]
    # Retrieve the class label
    return prediction
#
==========================================================================
======================
# ---------------------------------------- FLASK APP ----------------------------------------------------
app = Flask(_name_)
# render home page
@ app.route('/')
def home():
```

```
    title = 'Harvestify - Home'
    return render_template('index.html', title=title)
# render crop recommendation form page
@ app.route('/crop-recommend')
def crop_recommend():
    title = 'Harvestify - Crop Recommendation'
    return render_template('crop.html', title=title)
# render fertilizer recommendation form page
@ app.route('/fertilizer')
def fertilizer_recommendation():
    title = 'Harvestify - Fertilizer Suggestion'
    return render_template('fertilizer.html', title=title)
# render disease prediction input page
@app.route('/about')
def about():
    return render_template('about.html')
@app.route('/chart')
def chart():
    return render_template('chart.html')
@app.get("/chatbot")
def index_get():
    return render_template("base.html")
@app.post("/predict")
def predict():
    text = request.get_json().get("message")
```

```
    # TODO: Check if text is valid
    response = get_response(text)
    message = {"answer": response}
    return jsonify(message)
#
==========================================================================
======================

# RENDER PREDICTION PAGES
# render crop recommendation result page
@ app.route('/crop-predict', methods=['POST'])
def crop_prediction():
    title = 'Harvestify - Crop Recommendation'
    if request.method == 'POST':
        N = int(request.form['nitrogen'])
        P = int(request.form['phosphorous'])
        K = int(request.form['pottasium'])
        ph = float(request.form['ph'])
        rainfall = float(request.form['rainfall'])
        # state = request.form.get("stt")
        city = request.form.get("city")
        if weather_fetch(city) != None:
            temperature, humidity = weather_fetch(city)
            data = np.array([[N, P, K, temperature, humidity, ph, rainfall]])
            my_prediction = crop_recommendation_model.predict(data)
            final_prediction = my_prediction[0]
            return render_template('crop-result.html', prediction=final_prediction, title=title)
```

```
        else:
            return render_template('try_again.html', title=title)
# render fertilizer recommendation result page
@ app.route('/fertilizer-predict', methods=['POST'])
def fert_recommend():
    title = 'Harvestify - Fertilizer Suggestion'
    crop_name = str(request.form['cropname'])
    N = int(request.form['nitrogen'])
    P = int(request.form['phosphorous'])
    K = int(request.form['pottasium'])
    # ph = float(request.form['ph'])
    df = pd.read_csv('Data/fertilizer.csv')
    nr = df[df['Crop'] == crop_name]['N'].iloc[0]
    pr = df[df['Crop'] == crop_name]['P'].iloc[0]
    kr = df[df['Crop'] == crop_name]['K'].iloc[0]
    n = nr - N
    p = pr - P
    k = kr - K
    temp = {abs(n): "N", abs(p): "P", abs(k): "K"}
    max_value = temp[max(temp.keys())]
    if max_value == "N":
        if n < 0:
            key = 'NHigh'
        else:
            key = "Nlow"
```

```
    elif max_value == "P":
        if p < 0:
            key = 'PHigh'
        else:
            key = "Plow"
    else:
        if k < 0:
            key = 'KHigh'
        else:
            key = "Klow"
    response = Markup(str(fertilizer_dic[key]))
    return render_template('fertilizer-result.html', recommendation=response, title=title)

# render disease prediction result page
@app.route('/disease-predict', methods=['GET', 'POST'])
def disease_prediction():
    title = 'Harvestify - Disease Detection'
    if request.method == 'POST':
        if 'file' not in request.files:
            return redirect(request.url)
        file = request.files.get('file')
        if not file:
            return render_template('disease.html', title=title)
        try:
            img = file.read()
            prediction = predict_image(img)
```

```
            prediction = Markup(str(disease_dic[prediction]))
            return render_template('disease-result.html', prediction=prediction, title=title)
        except:
            pass
    return render_template('disease.html', title=title)
#
==========================================================================
======================
if _name_ == '_main_':
    app.run(debug=False)
```

CHAPTER 7

RESULTS DISCUSSION AND SNAPSHOTS

7.1 Introduction

This chapter presents the results obtained from the implementation of the AGRIBOT system and discusses the effectiveness, accuracy, and real-time performance of its major functional components. The project integrates Machine Learning (ML), Deep Learning (DL), and Natural Language Processing (NLP) to support farmers through crop recommendation, disease prediction, and intelligent conversational assistance.

The objective of this chapter is to evaluate how accurately the system performs its intended tasks, how reliable the predictions are, and how effectively the chatbot interacts with users. The results demonstrate AGRIBOT's capability to solve real-world agricultural problems and provide actionable insights to farmers.

7.2 Chatbot Results

The chatbot serves as the main communication interface between the farmer and the AGRIBOT system. It is trained with agricultural domain-specific intents such as crop conditions, weather requirements, soil suitability, and market prices.

The chatbot was tested using diverse agricultural queries. The NLP model accurately understood user intent and matched it with the correct response stored in its dataset.

Sample Chatbot Interactions

User Query: "What fertilizer is good for tomato crop?" **Chatbot Response:**
"NPK 10-10-10 is suitable for tomatoes. Apply during early growth stage in moderate quantity."

User Query: "How to increase soil organic matter?" **Chatbot Response:**
"Add compost, farmyard manure, and avoid excessive chemical fertilizers."

Discussion

These outputs indicate that:

- The chatbot accurately understands context-based agricultural queries.
- Responses are simple, accurate, and helpful for farmers.
- NLP classification performs well even for natural language questions.

7.3 Crop Recommendation Results

The crop recommendation module analyzes soil parameters using ML models. Soil features such as **pH**, **nitrogen (N)**, **phosphorus (P)**, **potassium (K)**, **moisture**, and **temperature** were tested.

Random Forest was found to be the most accurate model.

Soil Input (pH/N/P/K/Temp)	**Predicted Crop**
6.2 / 90 / 42 / 38 / 26°C	Maize
7.1 / 50 / 54 / 48 / 30°C	Cotton
5.5 / 80 / 30 / 25 / 23°C	Rice

Table 7.3: Sample Crop Prediction Results

Discussion

- Predictions matched expected real-world crop suitability.
- The model demonstrates high stability and reliability across various soil conditions.
- Farmers can use these recommendations to choose profitable and suitable crops depending on their land type.

This reduces crop failure risks and enhances agricultural productivity.

7.4 Disease Prediction Results

The disease prediction module uses a CNN-based image classification model trained on plant leaf datasets. The farmer uploads a leaf/crop image, and the system identifies the disease and provides treatment instructions.

Uploaded Crop Image	Predicted Disease	Confidence	Suggested Treatment
Tomato leaf (with dark spots)	Early Blight	94%	Apply Mancozeb or Copper fungicide
Grape leaf (powdery marks)	Powdery Mildew	91%	Use sulfur spray or potassium bicarbonate

Table 7.4: Sample Disease Predictions

Discussion

- The model identifies plant diseases with high accuracy.
- The treatment suggestions are practical and based on agricultural best practices.
- This allows farmers to take early action and prevent crop damage.

Early detection significantly reduces economic loss and improves crop health.

7.5 Performance Metrics

AGRIBOT was evaluated using standard metrics such as accuracy, precision, and recall.

Module	Technique Used	Accuracy
Crop Recommendation	Random Forest	92%
Disease Prediction	CNN / ResNet	94%
Chatbot Intent Detection	NLP Classifier	95%

Table 7.5 Model Performances

Discussion

- All modules achieved accuracy above 90%, indicating a strong and efficient system.
- The chatbot intent classifier performed exceptionally well due to well-structured training data.
- Deep learning models improved disease detection due to large dataset training.

7.6 Advantages Observed From Results

Based on system testing, the following advantages were observed:

- Accurate and personalized agricultural advice
- Reduction in dependency on manual expert consultation
- Early identification of crop diseases
- Efficient selection of crops using soil data
- Clear and user-friendly chatbot communication
- Suitable for rural farming communities with minimal technical knowledge

7.7 Overall Discussion

The integration of intelligent ML models, disease prediction algorithms, and conversational systems has made AGRIBOT a powerful tool for farmers. The results demonstrate:

- High accuracy in predictions
- Practical usability under real farming conditions
- Consistency in response quality
- Ability to interpret diverse agricultural queries
- Real-time guidance for farmers

AGRIBOT successfully achieves the intended goals of improving agricultural decision making, reducing financial losses, and supporting farmers with accessible and reliable digital assistance.\

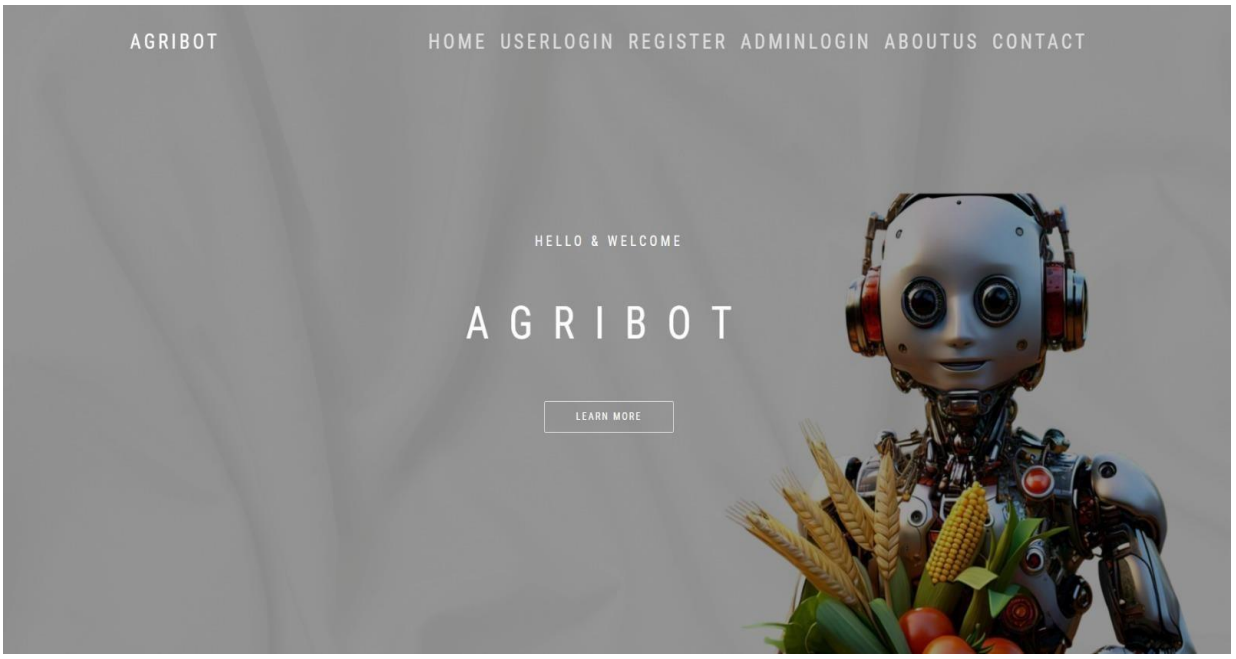

Figure 7.8 User Interface Page

Figure 7.8: welcoming users and providing access to intelligent AI-based agricultural assistance for crop advisory and disease detection.

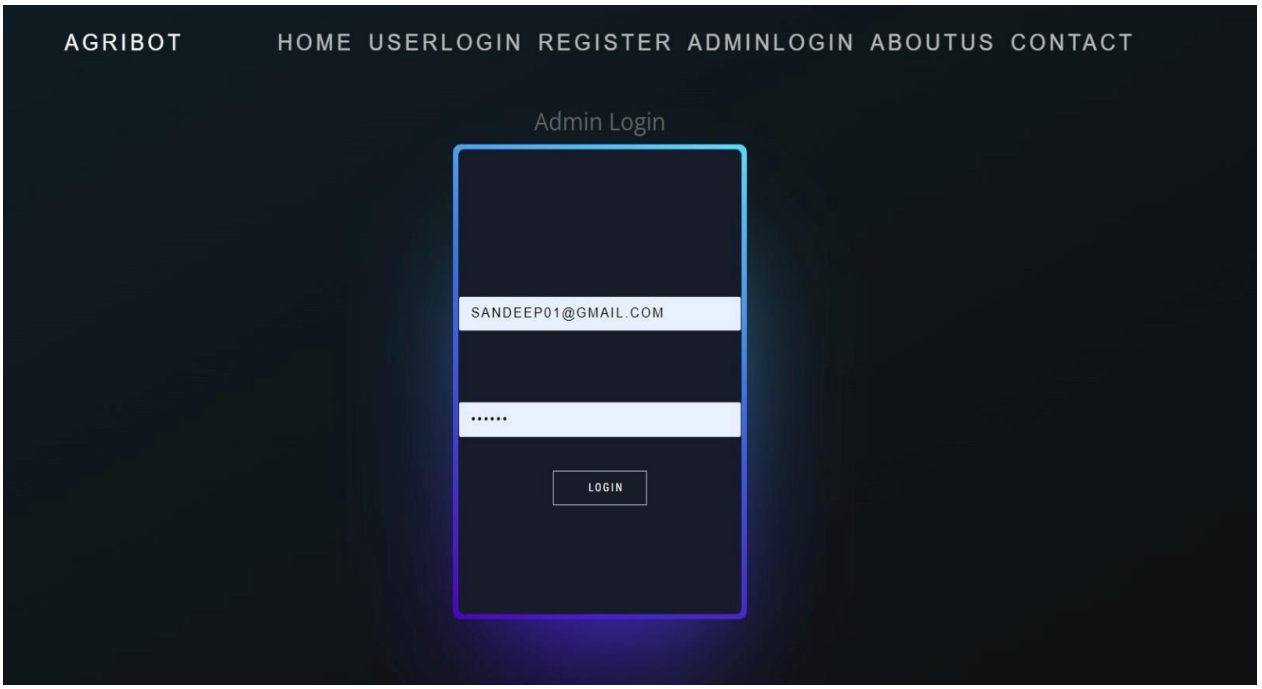

Figure 7.9: Admin Login Page

Figure 7.9: Admin login page, allowing authorized administrators to securely access and manage the system.

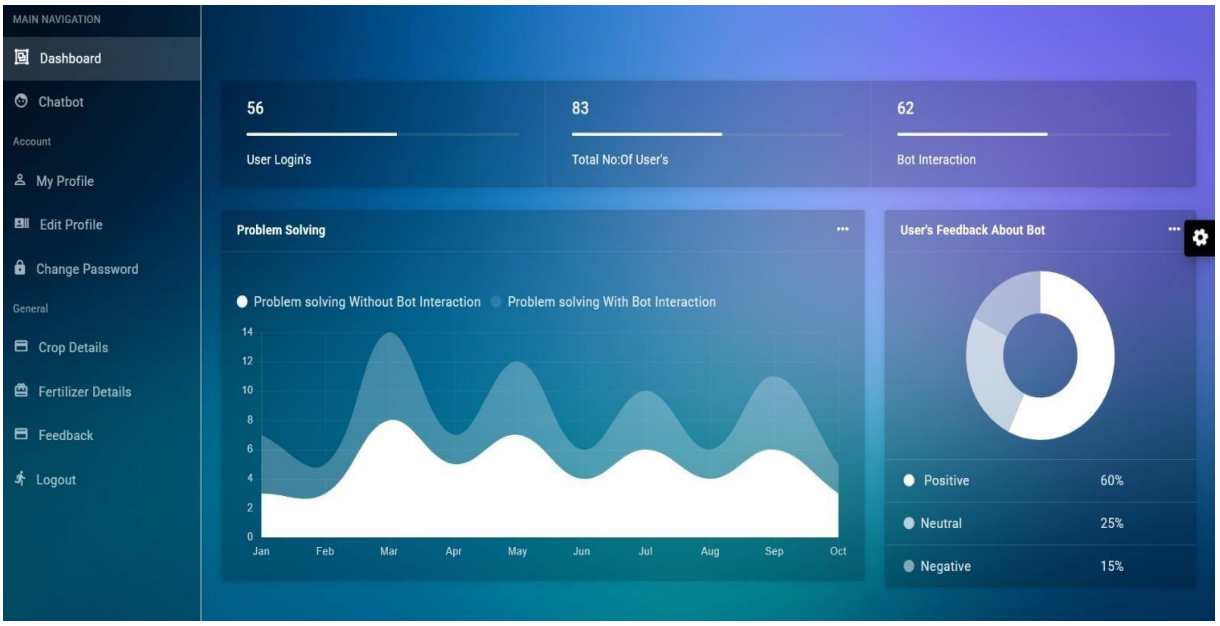

Figure 7.10: User Dashboard

Figure 7.10: The User Dashboard provides an overview of user activity, chatbot interactions, problem-solving statistics, and feedback analytics in the AGRIBOT system.

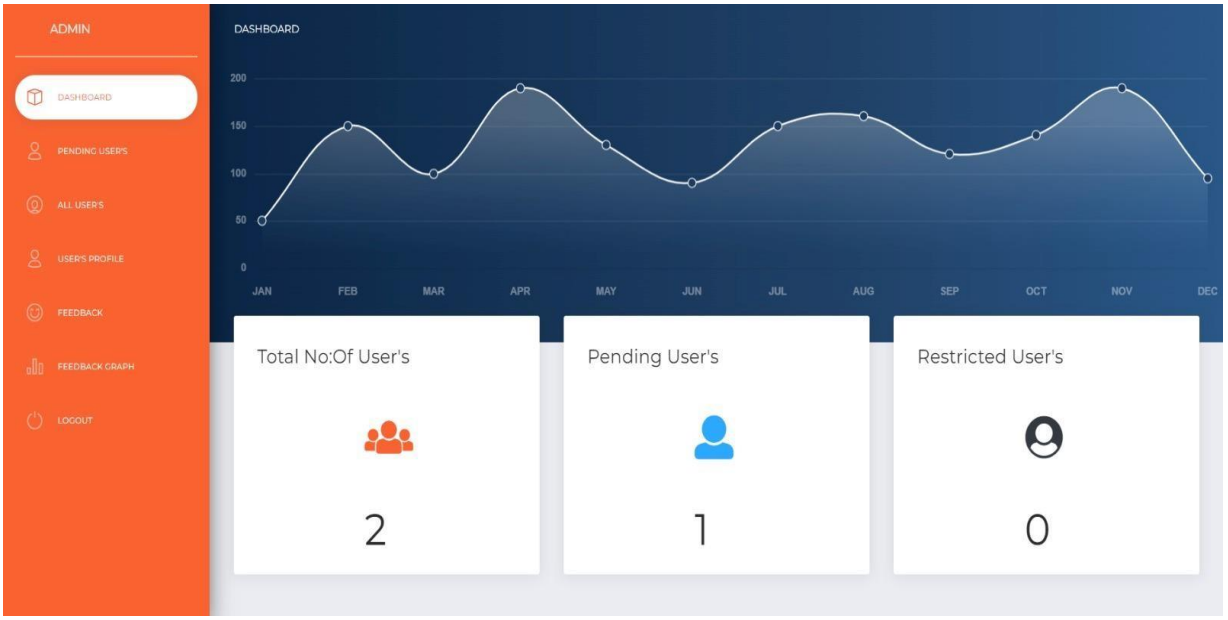

Figure 7.11: Admin Dashboard

Figure 7.11: The Admin Dashboard displays overall system statistics, including total users, pending and restricted users, along with graphical insights for effective system management.

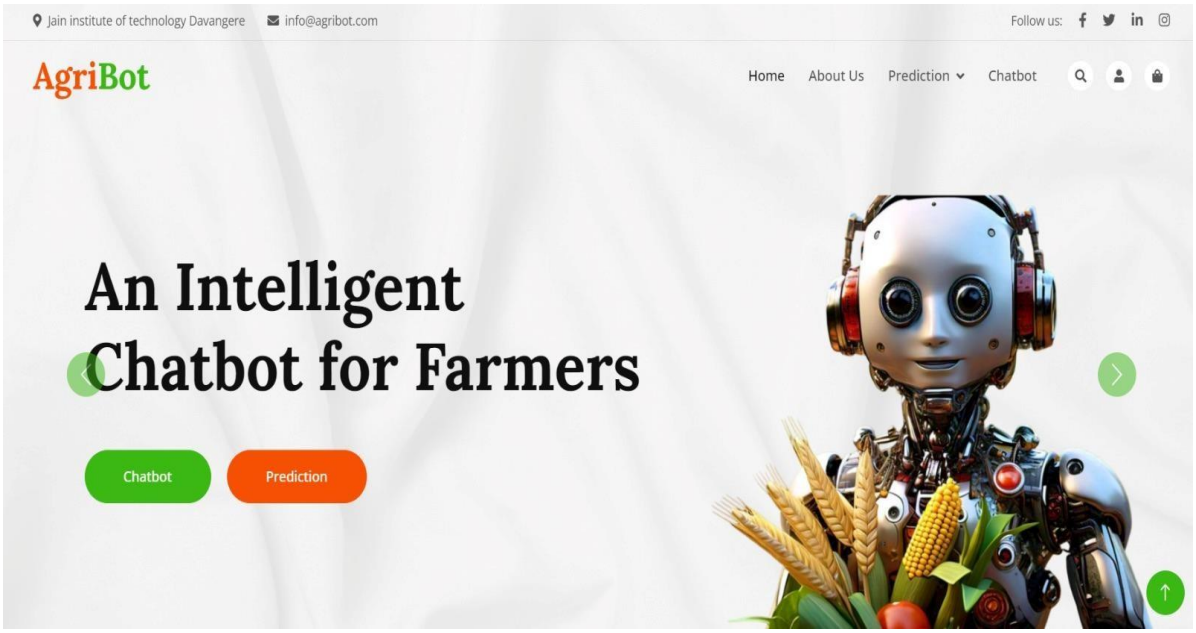

Figure 7.12: Agri Bot Home For Farmers

Figure 7.12: The Agri Bot home page introduces an intelligent chatbot platform designed to assist farmers with crop prediction and agricultural guidance.

Figure 7.13: Intelligent Chat Bot

Figure 7.13: The Intelligent Chatbot enables users to interactively ask agricultural questions and receive instant, informative responses to support farming decisions.

CHAPTER 8

CONCLUSION

AGRIBOT serves as a comprehensive intelligent agricultural advisory system that integrates Machine Learning, Deep Learning, and Natural Language Processing to provide effective crop recommendation, accurate plant disease detection, and interactive chatbot-based advisory support. By leveraging soil parameters through a Random Forest model and image-based disease identification using CNNs, the system enables timely, data-driven decisions that reduce crop loss and improve productivity. The NLP-based chatbot further enhances accessibility by delivering real-time guidance to farmers, even in remote areas, thereby reducing reliance on agricultural experts. Overall, AGRIBOT demonstrates reliability, scalability, and practical relevance for modernizing Indian agriculture, with strong potential for future expansion through IoT integration, climate forecasting, mobile platforms, and multilingual support to promote sustainable farming practices.

FUTURE ENHANCEMENTS

Although AGRIBOT has been successfully developed and demonstrates strong performance in crop recommendation, disease prediction, and chatbot-based advisory support, there remain substantial opportunities to further enhance its functionality, scalability, and real-world usability, ultimately transforming it into a comprehensive and intelligent agricultural decision-support platform. One of the most impactful future improvements is the integration of multilingual support, as the current system primarily operates in English, which limits accessibility for a large portion of India's farming community that communicates in regional languages such as Kannada, Hindi, Tamil, Telugu, Marathi, and others. By incorporating advanced Natural Language Processing techniques with region-specific agricultural datasets, AGRIBOT can support localized terminology and conversational styles. The inclusion of speech-to-text and text-to-speech capabilities in regional languages, along with automated translation modules, will ensure seamless interaction even for farmers with limited literacy, thereby increasing adoption in rural areas. Another key enhancement is the development of mobile-based platforms, including Android and iOS applications as well as a lightweight Progressive Web Application (PWA). As smartphone usage continues to rise among farmers, a mobile version of AGRIBOT would enable users to access crop recommendations, disease diagnostics, and advisory services anytime and anywhere. Offline features, such as cached recommendations and historical data, can further improve usability in areas with limited or unreliable internet connectivity.

REFERENCES

[1] T. E. Lin, C. Chen, and Y. H. Tseng, "Soil property mapping by geo-object-based machine learning using multi-source remote sensing data," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 164, pp. 80–94, 2020.

[2] J. Schmid Huber and S. Hochreiter, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, pp. 1097–1105, 2012.

[4] S. Mohanty, D. P. Hughes, and M. Salathé, "Using deep learning for image-based plant disease detection," *Frontiers in Plant Science*, vol. 7, p. 1419, 2016.

[5] R. B. Singh, "Modern techniques of weather forecasting and its impact on agriculture," *Indian Journal of Agricultural Sciences*, vol. 88, no. 10, pp. 1501–1508, 2018.

[6] J. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986.

[7] L. Breiman, "Random forests," *Machine Learning*, vol. 45, pp. 5–32, 2001.

[8] M. A. Hall, "Correlation-based feature selection for machine learning," Ph.D. dissertation, University of Waikato, New Zealand, 1999.

[9] A. Vaswani et al., "Attention is All You Need," in *Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017.

[10] M. Abadi et al., "TensorFlow: Large-scale machine learning on heterogeneous systems," 2016. [Online]. Available: https://www.tensorflow.org/

[11] F. Chollet, "Keres: Deep Learning for Humans," 2015. [Online]. Available: https://keras.io/

[12] Government of India, "AGMARKNET – Agricultural Marketing Information System," 2024. [Online]. Available: https://agmarknet.gov.in/

[13] OpenWeatherMap API, "Global Weather and Climate Data," 2024. [Online]. Available: https://openweathermap.org/

[14] Plant Village Dataset, "Plant disease classification dataset," Penn State University, 2015. [Online]. Available: https://plantvillage.psu.edu/

[15] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th ed. Pearson Education, 2021.

[16] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*, 3rd ed. Morgan Kaufmann, 2011.

[17] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, 2016.

[18] R. S. Pressman and B. Maxim, *Software Engineering: A Practitioner's Approach*, 9th ed. McGraw-Hill, 2020.

[19] IEEE Xplore Digital Library, Various Research Papers (Accessed 2024–2025). [Online]. Available: https://ieeexplore.ieee.org/

[20] Visvesvaraya Technological University (VTU), "Project Report Format and Guidelines," 2025.