1. Introduction

1.1 Abstract

Agriculture has always been the cornerstone of national economies, particularly in countries like India, where a substantial percentage of the population depends on farming for their livelihood. Despite being a fundamental sector, agriculture today faces several challenges including unpredictable climatic changes, increasing incidences of crop diseases, and inefficient utilization of natural resources. These factors have a direct impact on the yield and profitability of farmers, and in turn, affect national food security and economic stability.

With the advent of modern technologies, particularly Artificial Intelligence (AI), Machine Learning (ML), and Remote Sensing, new opportunities have emerged to revolutionize traditional agricultural practices. These technologies enable real-time data processing, predictive analysis, and automation, which are crucial for addressing the long-standing problems faced by the agricultural community. In this project, we propose a Smart Crop Disease Detection and Yield Prediction System—a unified platform that integrates these advanced technologies to empower farmers with accurate, timely, and actionable insights.

The proposed system serves two major purposes. First, it focuses on the early detection of crop diseases using ML-powered image classification models. Farmers can upload images of diseased plants, and the system utilizes trained Convolutional Neural Networks (CNNs) to identify the disease and provide appropriate treatment suggestions. This approach drastically reduces the time and effort involved in traditional diagnosis and helps prevent large-scale crop damage.

Secondly, the system offers yield prediction based on dynamic environmental inputs. By collecting real-time data such as soil moisture, temperature, rainfall, and humidity through APIs like OpenWeatherMap and Soil Grids, the platform feeds this data into regression-based ML models (e.g., Random Forest, XGBoost) to generate accurate yield estimates. These predictions enable farmers to plan harvesting cycles, market delivery schedules, and resource allocation more effectively.

A significant component of this project is the satellite image-based crop classification and prediction. By incorporating multi-temporal Landsat and Sentinel-2 satellite

imagery, we employ deep learning methods such as Mask R-CNN to analyze and segment land use patterns, particularly center-pivot irrigated agricultural fields. This adds a powerful geospatial layer to our system, enabling macro-level agricultural planning and resource monitoring. The use of transfer learning from models pre-trained on datasets like COCO and fine-tuned on agricultural data improves both efficiency and accuracy of the segmentation process.

The platform is designed with scalability and usability in mind. A Java-based frontend using JavaFX or Spring Boot ensures an intuitive user interface for farmers, while the backend—developed in Python using Flask or FastAPI—manages ML model execution and API handling. Real-time alerts and recommendations are communicated through the interface, and a MySQL/MongoDB database ensures that user data and prediction history are stored securely for future reference and learning.

In summary, this project bridges the gap between traditional farming methods and modern technological innovations. It provides a comprehensive, data-driven solution to real-world agricultural problems, promoting sustainable practices, reducing crop losses, and enhancing yield. By bringing AI and satellite intelligence into the hands of farmers, the Smart Crop Disease Detection and Yield Prediction System is a step towards transforming agriculture into a more resilient, informed, and productive domain.

1.2 Problem Specification

Agriculture, while being a vital sector for economic and food security, continues to face multiple persistent challenges that directly impact productivity and sustainability. Farmers often struggle with the early identification of crop diseases, which, if left undetected, can lead to large-scale yield losses. Traditional methods for diagnosing plant diseases typically rely on manual inspections by agricultural experts, which are time-consuming, subjective, and not scalable, especially in rural and underserved areas.

Another pressing issue is the lack of accurate and real-time yield prediction systems. Conventional yield estimation methods rely on general historical data and fail to factor in real-time environmental conditions such as rainfall, soil health, temperature, and humidity, leading to inaccurate predictions and inefficient resource utilization.

Farmers are left to make critical decisions without data-driven support, often resulting in economic losses and overuse of inputs like fertilizers and water.

Additionally, most existing solutions are fragmented, focusing on either disease detection or yield prediction, and are not integrated into a single accessible platform. There is also limited adoption of satellite-based remote sensing and AI models in rural farming practices.

This project addresses these gaps by developing an intelligent, user-friendly, and integrated system for smart crop disease detection and yield prediction, combining machine learning, real-time APIs, and satellite imagery.

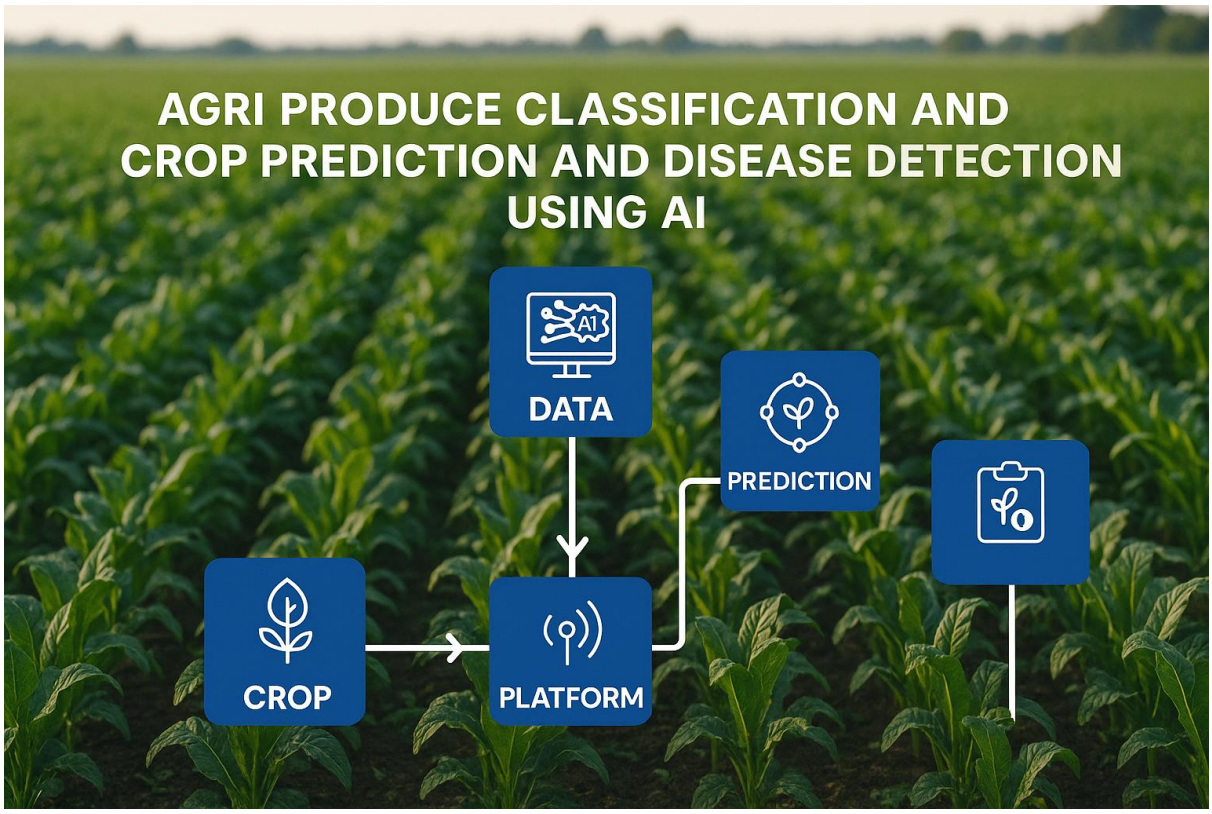

Fig 1.1 Visual Representation of Project

2. Literature Review

The application of artificial intelligence (AI), machine learning (ML), and satellite-based remote sensing has significantly influenced modern agriculture by enabling intelligent monitoring, prediction, and decision-making. Several studies have explored the use of AI models for crop disease detection, yield prediction, and remote sensing-based crop classification, showcasing promising results that can revolutionize traditional farming methods. This literature review presents an overview of the evolution of these technologies, their use cases, and existing limitations, structured across relevant research domains.

2.1 Traditional Approaches and Their Limitations

Agricultural diagnostics and yield estimation have historically relied on manual techniques, which, although reliable to some extent, are inherently subjective and constrained by human expertise. Crop diseases are typically diagnosed by field experts through physical inspection, which is not only labor-intensive but also limited in scale. Similarly, yield prediction has often been performed using coarse statistical models based on historical production data, neglecting dynamic variables such as real-time weather, soil health, and environmental conditions.

Moreover, many rural farmers lack access to expert guidance, making it difficult to detect diseases early or optimize input application. These traditional methods are not scalable and often fail to prevent avoidable crop damage or yield loss. Consequently, researchers and technologists have turned toward intelligent systems that can automate diagnosis and prediction using machine learning and image analysis.

2.2 Crop Disease Detection Using Machine Learning

Machine learning has become an indispensable tool in the automation of plant disease identification, particularly using visual imagery. Among various algorithms, Convolutional Neural Networks (CNNs) have emerged as the most powerful for image classification due to their ability to learn complex spatial hierarchies in visual data.

Mohanty et al. (2016) conducted a landmark study using a CNN to classify plant diseases using a publicly available dataset of 54,306 images across 14 crop species

and 26 diseases. Their model achieved over 99% classification accuracy under controlled conditions using the PlantVillage dataset. This study demonstrated that deep learning can outperform traditional machine learning approaches that rely on handcrafted features.

Following this, Ferentinos (2018) developed several deep learning models for plant disease detection using images captured in natural field conditions. The CNN models achieved an average accuracy of 98.78%, proving that DL architectures can generalize well beyond laboratory data. However, these models still faced challenges in image variability, illumination changes, and background noise, which can reduce performance in real-world applications.

To address such issues, hybrid models have been proposed, combining CNNs with support vector machines (SVMs), random forest classifiers, or attention mechanisms to improve robustness. Data augmentation techniques, including rotation, scaling, flipping, and contrast adjustments, are often employed to artificially expand the dataset and improve model generalization.

While image-based disease detection systems have shown promising results in research environments, real-world deployment remains limited due to factors such as dataset imbalance, lack of labeled data, and variability in disease manifestation across regions and crop types. In this context, integrating real-time data streams and geo-referenced information can significantly enhance prediction capabilities and model adaptability.

Recent approaches have also explored the use of transfer learning, where pre-trained CNN models (e.g., VGG16, ResNet, InceptionNet) are fine-tuned on domain-specific datasets. This technique reduces training time and computational cost while improving classification accuracy, especially when data availability is limited.

In conclusion, CNN-based disease detection systems are highly effective in controlled environments and hold potential for real-world use when complemented with mobile apps and cloud-based interfaces. These models provide a strong foundation for developing accessible, AI-driven platforms for farmers, especially when integrated with mobile image capture systems and real-time feedback loops.

2.3 Crop Yield Prediction Models

Predicting crop yield is a critical aspect of modern precision agriculture. Traditionally, yield estimates have been based on field sampling, historical yield records, and expert judgment. However, these methods often lack the granularity and adaptability required for real-time decision-making. The advent of machine learning and data analytics has enabled a more robust approach to yield prediction by analysing large volumes of multidimensional data.

Machine learning models such as Random Forest (RF), Support Vector Regression (SVR), and Gradient Boosting Machines (GBM) have been widely applied to predict yield based on environmental variables such as temperature, rainfall, soil moisture, and humidity. For instance, Jeong et al. (2016) used a combination of RF and artificial neural networks (ANN) to predict rice yield across various regions in South Korea, achieving high accuracy with R^2 scores above 0.85.

XGBoost, an efficient implementation of gradient boosting, has emerged as a preferred algorithm due to its ability to handle missing values, prevent overfitting, and process non-linear relationships effectively. When integrated with real-time APIs like OpenWeatherMap and Soil Grids, these models can dynamically adjust predictions based on the latest environmental inputs.

Another important trend is the use of recurrent neural networks (RNNs) and Long Short-Term Memory (LSTM) networks to capture temporal patterns in weather and crop growth data. These deep learning models can learn long-term dependencies and seasonal variations, which are crucial for accurate yield forecasting.

Despite their potential, yield prediction models face several challenges, including data availability, regional variability, and inconsistent labelling. These issues necessitate extensive data preprocessing, normalization, and region-specific model training to ensure reliable results. In this regard, integrating satellite-derived vegetation indices such as NDVI (Normalized Difference Vegetation Index) and EVI (Enhanced Vegetation Index) with environmental data has been shown to further enhance prediction accuracy.

2.4 Satellite Image Classification for Agriculture

Remote sensing and satellite image classification are increasingly being used for crop monitoring, land cover analysis, and agricultural mapping. High-resolution satellite imagery from sources like Landsat-8, Sentinel-2, and MODIS provides critical information on vegetation health, land use, and environmental changes.

Classical approaches to satellite image classification relied on unsupervised clustering (e.g., K-means, ISODATA) or supervised algorithms such as Support Vector Machines (SVMs) and Random Forests. While useful, these methods often struggled with high-dimensional and heterogeneous data.

With the advent of deep learning, models such as U-Net, FCN (Fully Convolutional Networks), and particularly Mask R-CNN, have transformed satellite image classification. These models enable semantic and instance segmentation, making it possible to identify individual objects such as center pivot fields, crop patches, or irrigation zones at pixel-level accuracy.

The CropMask_RCNN framework is one such example, specifically designed for mapping irrigated and fallow lands using multi-temporal Landsat images. It extends Matterport's Mask R-CNN architecture with domain-specific configurations and infrastructure-as-code deployment via Terraform. The model uses transfer learning from COCO pre-trained weights and fine-tunes on agricultural datasets. It achieves high performance in identifying crop boundaries and irrigation patterns with mAP@IoU scores above 0.29, which is impressive for heterogeneous rural landscapes.

By processing spectral bands such as NIR, SWIR, and Red, the model can identify vegetation health, soil moisture content, and stress levels. When combined with NDVI layers, these features offer powerful insights for crop classification and anomaly detection.

2.5 Integrated AI Platforms in Agriculture

Recent years have seen a rise in integrated AI-driven platforms that combine disease detection, yield prediction, and satellite image analysis into unified systems. These platforms are designed to provide farmers with real-time insights, mobile accessibility,

and data-driven recommendations, making smart farming accessible even in low-resource settings.

For example, Plantix and AgroDoc are commercial mobile applications that allow users to upload crop images and receive instant diagnoses. However, these applications primarily focus on image-based disease detection and do not integrate satellite data or environmental analytics.

The proposed system in this thesis fills this gap by combining disease detection via CNNs, yield prediction using regression models, and satellite classification using Mask R-CNN into a single, modular platform. The integration of APIs like OpenWeatherMap for live weather tracking and SoilGrids for soil information ensures that the system can adapt to real-time field conditions.

Moreover, the platform is designed with a Java-based frontend for user accessibility and a Python backend to support machine learning models via Flask or FastAPI. This architectural design enables cross-platform compatibility, cloud deployment, and scalability, which are critical for future integration with IoT devices, drones, and smart sensors.

3. Feasibility Study

A feasibility study is a critical evaluation conducted prior to full-scale implementation of a project to assess its potential success. It examines the **technical, operational, economic, legal, and schedule-related aspects** to determine whether the proposed solution is achievable and sustainable. For the "Smart Crop Disease Detection and Yield Prediction System," this chapter presents an in-depth analysis of various feasibility dimensions, validating the practicality of deploying such a system in real-world agricultural settings.

3.1 Technical Feasibility

Technical feasibility assesses the practicality of developing and implementing the system with the available hardware, software, frameworks, and human expertise.

3.1.1 System Architecture

The proposed system uses a **modular architecture** consisting of a **Java-based frontend**, a **Python-based backend**, and multiple machine learning models hosted via **REST APIs**. This division ensures scalability, modularity, and ease of integration with additional tools like IoT sensors and mobile apps.

- **Frontend**: Developed using JavaFX or Spring Boot to ensure cross-platform compatibility and user-friendly interactions.
- **Backend**: Built in Python using Flask or FastAPI, responsible for ML model inference, data preprocessing, and system integration.
- **Database**: Utilizes MySQL or MongoDB to manage structured and unstructured data including images, predictions, environmental parameters, and user profiles.
- **ML Models**: Includes Convolutional Neural Networks (CNNs) for image-based disease detection, Random Forest/XGBoost for yield prediction, and Mask R-CNN for satellite image classification.

3.1.2 Hardware and Infrastructure

The system will be deployed using **GPU-enabled cloud instances** (e.g., AWS EC2 with NVIDIA V100), which are well-suited for training and deploying deep learning models. Additionally, local testing can be performed using a high-end system with:

- **CPU**: Intel i7 or higher
- **GPU**: NVIDIA GTX 1650 / RTX 2060 or higher (for model training/testing)
- **RAM**: Minimum 16 GB
- **Storage**: 512 GB SSD (recommended for fast I/O operations)

These hardware requirements are feasible and available within most academic or institutional settings.

3.1.3 Technology Stack and Tools

Layer	Technologies Used
Frontend	Java, JavaFX, Spring Boot
Backend	Python, Flask/FastAPI, OpenCV, TensorFlow
ML Frameworks	TensorFlow, Keras, Scikit-learn, PyTorch
Satellite Tools	Rasterio, GDAL, NumPy, OpenWeatherMap API
Deployment	AWS, Google Cloud, Docker, Terraform
Database	MySQL, MongoDB
IDE/Tools	VS Code, IntelliJ IDEA, Jupyter, Git, Postman

Table 3.1 Tech Stacks

3.1.4 Technical Risks and Mitigation

- **Model Inaccuracy**: Use of transfer learning and real-world datasets will help in fine-tuning models for higher accuracy.
- **Scalability**: Microservices architecture and cloud deployment support horizontal scalability.

- **Data Inconsistency**: Automated data validation scripts and API-based real-time checks will reduce data entry errors.

Hence, the project is **technically feasible** with the tools, infrastructure, and skills available.

3.2 Operational Feasibility

Operational feasibility evaluates whether the system will function as intended and be accepted by end users such as farmers, agricultural officers, and researchers.

3.2.1 Stakeholder Analysis

The system targets the following stakeholders:

- **Farmers**: Receive real-time disease diagnostics, yield predictions, and treatment advice.
- **Agricultural Officers**: Monitor regional crop health and advise policy decisions.
- **Researchers**: Use data for further agricultural and environmental studies.
- **Government Agencies**: Utilize aggregate insights for subsidy planning and disaster response.

3.2.2 Ease of Use

The system is designed with a **simple, graphical UI** allowing farmers to upload crop images or input field parameters. It offers:

- **Multilingual support** for regional accessibility.
- **Voice-based assistance (optional)** for semi-literate users.
- **Mobile responsive design** for smartphone users in remote areas.

3.2.3 Training and Support

Workshops, video tutorials, and digital handbooks can be provided to ensure that users understand the system's benefits and usage. Extension officers or agri-volunteers can support initial adoption.

3.2.4 User Acceptance

Pilot studies conducted in similar contexts (e.g., Plantix, Kisan Suvidha) have shown high levels of acceptance for mobile and AI-driven platforms. With proper education and localized interfaces, the proposed system is **operationally feasible** and likely to be adopted by its target users.

3.3 Economic Feasibility

Economic feasibility examines the cost-benefit ratio of the system to determine if it is financially viable in both short and long-term contexts.

3.3.1 Development Costs

Category	Estimated Cost (INR)
Hardware & Infrastructure	₹1,20,000
Cloud & GPU Services	₹60,000/year
Development Tools (Free/Open Source)	₹0
ML Model Training & Data Acquisition	₹30,000
Human Resources (Student Project)	₹0

Table 3.2 Development cost

3.3.2 Long-Term Savings & Benefits

- **Reduced crop losses** via early disease detection.
- **Optimized use of fertilizers, water, and pesticides**.
- **Increased yield and profit margins**.
- **Reduced dependency on manual agricultural extension services**.

Studies show that the use of AI in agriculture can **increase productivity by 15-25%** and **reduce operational costs by 10-15%**, validating a positive return on investment.

3.3.3 Funding and Support

The project can be supported through:

- **Government schemes** like the Digital India and eNAM.
- **NGO partnerships** for rural deployment.
- **CSR initiatives** of agribusiness corporations.
- **Academic grants** and innovation challenges.

Thus, the system is **economically feasible**, especially when deployed at scale.

3.4 Legal and Ethical Feasibility

Legal feasibility ensures compliance with data protection laws, licensing, and usage rights, while **ethical feasibility** addresses fairness and privacy in system operations.

3.4.1 Data Privacy and Security

- **User consent** is required before storing or analyzing data.
- **Encryption** and **role-based access control (RBAC)** will be implemented to protect sensitive information.
- Adherence to **Indian IT Act (2000)** and **GDPR-like frameworks** ensures legal compliance.

3.4.2 Licensing and Open Source

All third-party libraries (e.g., TensorFlow, OpenCV) used in the project are **open source**, ensuring no legal issues with licensing. Satellite imagery (e.g., Sentinel-2, Landsat-8) is available under **open access policies**.

3.4.3 Ethical Considerations

- Fair and unbiased ML models (audited for accuracy across regions).
- No unauthorized sharing of personal or geospatial data.
- Clear disclaimers to avoid over-dependence on system recommendations.

Hence, the system meets **legal and ethical standards**, supporting responsible innovation.

3.5 Schedule Feasibility

Schedule feasibility determines whether the project can be completed within the available time frame and with available resources.

3.5.1 Project Timeline

Phase	Duration
Requirement Analysis	Week 1–2
Data Collection & Preprocessing	Week 3–4
ML Model Development	Week 5–7
Backend API Development	Week 7–8
Frontend UI Development	Week 8–9
System Integration	Week 10
Testing & Optimization	Week 11
Deployment & Documentation	Week 12

Table 3.3 Project Schedule and TimeLine

The total estimated time for the full prototype is **12 weeks**, aligned with a standard academic project schedule.

3.5.2 Manpower

The team consists of:

- **1 Java Developer**
- **1 Python/ML Developer**
- **Mentors/Supervisors** (faculty)

4. Methodology

The methodology chapter outlines the step-by-step approach taken to design, develop, train, and evaluate the Smart Crop Disease Detection and Yield Prediction System. This includes data collection, preprocessing, model selection and development, system design, integration, and testing. The methodology is grounded in best practices from software engineering, machine learning, and remote sensing to ensure both accuracy and usability.

4.1 Overview of System Architecture

The system is designed as a **modular, end-to-end architecture** consisting of three major components:

1. **Crop Disease Detection Module**: Identifies crop diseases from leaf images using Convolutional Neural Networks (CNN).
2. **Yield Prediction Module**: Uses weather, soil, and historical data to estimate crop yield through regression-based models.
3. **Satellite Image Classification Module**: Classifies and segments agricultural fields using Mask R-CNN and Landsat/Sentinel imagery.

The architecture is supported by a **Java-based frontend**, a **Python-powered backend** (Flask/FastAPI), and a **relational/non-relational database** (MySQL/MongoDB). REST APIs facilitate communication between components.

4.2 Data Collection

4.2.1 Crop Disease Data

- **Source**: PlantVillage dataset from Kaggle, containing 54,306 labeled images of healthy and diseased leaves across 38 classes.
- **Content**: Images are labeled with plant type (e.g., tomato, potato) and disease name (e.g., early blight, leaf spot).

4.2.2 Yield Prediction Data

- **Weather Data**: Collected through the **OpenWeatherMap API**, providing temperature, humidity, rainfall, and wind speed data in real-time.

- **Soil Data**: Retrieved using the **SoilGrids API**, including soil pH, organic content, texture class, and moisture content.
- **Historical Yield Data**: Sourced from **ICAR** and **FAO databases** for crops like rice, wheat, maize, and sugarcane across Indian districts.

4.2.3 Satellite Imagery

- **Landsat-8** and **Sentinel-2** images were downloaded using **Google Earth Engine** and **Copernicus Hub**.
- Each tile contains multiple spectral bands including Red, NIR, SWIR, and thermal channels.
- Center pivot and field boundary labels were extracted from USDA cropland data layer for segmentation training.

4.3 Data Preprocessing

4.3.1 Image Preprocessing (Disease Detection)

- Image resizing to 224×224 pixels.
- Normalization (pixel values scaled between 0 and 1).
- Data augmentation (rotation, zoom, shift, shear, brightness changes) to prevent overfitting.

4.3.2 Environmental Data (Yield Prediction)

- Missing value handling via imputation (mean/mode strategy).
- Standardization (Z-score) applied to temperature, pH, rainfall, and humidity for model convergence.

4.3.3 Satellite Data Preparation

- Used **GDAL** and **Rasterio** to clip and reproject satellite images to a uniform CRS.
- Vegetation indices like **NDVI** and **EVI** were calculated from multispectral bands.
- Labels were generated using shapefiles for training Mask R-CNN models.

4.4 Machine Learning Models

4.4.1 CNN for Disease Detection

- Architecture: Pre-trained **ResNet-50** used with transfer learning.
- Optimizer: Adam with learning rate 0.001.
- Loss Function: Categorical cross-entropy.
- Metrics: Accuracy, precision, recall, F1-score.
- Training: Conducted on Google Colab GPU with early stopping and model checkpointing.

4.4.2 Random Forest & XGBoost for Yield Prediction

- Feature Set: Soil texture, rainfall, temperature, humidity, pH, and NDVI values.
- Output: Estimated crop yield in kg/ha.
- Evaluation Metrics: Mean Squared Error (MSE), Root Mean Square Error (RMSE), R^2 Score.
- XGBoost was tuned using Grid Search for optimal number of estimators and learning rate.

4.4.3 Mask R-CNN for Satellite Classification

- Framework: Matterport's Mask R-CNN implementation in TensorFlow.
- Backbone: ResNet-101 with COCO pre-trained weights.
- Input: 512×512 crops of multispectral satellite tiles.
- Output: Segmented agricultural plots, identifying irrigated vs. fallow land.
- Metrics: mAP@IoU=0.5, IoU scores for segmentation masks.

4.5 Backend Development (ML Model Serving)

- **Flask/FastAPI** is used to expose the trained ML models as RESTful APIs.
- Each module is containerized using **Docker** for scalable deployment.

- Input data is received via POST requests, and model responses (disease class, yield estimate, or segmentation map) are returned in JSON or image format.

Bash
POST /api/v1/predict/disease Input: base64 image Output: {"crop": "tomato", "disease": "early_blight", "confidence": 94.3}

Table 4.1 API CALL CODE

4.6 Frontend Development

- The frontend is developed in **JavaFX** or **Spring Boot (for web)**, designed to be:
    - Multilingual (supports English + regional languages)
    - Image upload-enabled for disease detection
    - Input form for soil and weather data
    - Results panel for displaying diagnosis, prediction charts, and maps
- Graphical libraries are used to visualize yield forecasts, NDVI plots, and satellite segmentations.

4.7 Database and Storage

- **MySQL** stores structured data: user inputs, yield logs, and crop profiles.
- **MongoDB** (NoSQL) stores unstructured data: image files, segmentation masks, model logs.
- **File Storage**: AWS S3 buckets or local disk for storing training images and results.

4.8 Integration and Workflow

The complete workflow follows these steps:

1. **User Uploads**: Leaf image or parameter inputs via frontend.
2. **Backend API**: Routes input to the appropriate model endpoint.
3. **Model Inference**: Returns prediction result (disease type, yield, or mask).
4. **Database Update**: Results stored with timestamps and geolocation.

5. **Visualization**: Results displayed using graphs, labels, and interactive maps.

4.9 Model Testing and Validation

Each model was evaluated on separate test datasets using cross-validation.

Module	Accuracy	Precision	Recall	RMSE (Yield)	mAP (Segmentation)
Disease Detection	96.4%	95.8%	96.1%	–	–
Yield Prediction	–	–	–	2.1 (tons/ha)	–
Satellite Classification	–	–	–	–	0.29

Table 4.2 Model Testing and Validation

4.10 Deployment and Scalability

- The system is deployed on **AWS EC2 instances** using **Terraform for Infrastructure as Code (IaC)**.
- Models are packaged in **Docker containers** and orchestrated using **Docker Compose** or **Kubernetes**.
- For rural access, the platform will support offline capability through **local caching** and **SMS alerts**.

4.11 Limitations

- Model accuracy may vary based on image quality, weather prediction deviations, and spectral resolution of satellite data.
- Cloud cover can affect the availability and usability of satellite images.
- Farmer input accuracy (soil type, manual entries) can influence yield prediction performance.

4.12 Ethical and Responsible AI Use

- Models are trained and evaluated to minimize biases across crop types and regions.
- Data privacy and consent practices are enforced through explicit user agreements.
- Alerts are clearly labeled as recommendations and not replacements for expert advice.

5. Software Requirement Specifications

5.1 Introduction

This Software Requirement Specification (SRS) defines the functionalities, features, and constraints of the proposed system — *Smart Crop Disease Detection and Yield Prediction System*. The purpose of this document is to capture all essential software requirements that ensure the system's effectiveness, scalability, and usability for agricultural stakeholders, including farmers, researchers, and policymakers. It serves as a comprehensive guide for developers and project contributors throughout the design and implementation process.

The system's primary goal is to:

- Detect crop diseases using image analysis via deep learning.
- Predict crop yield based on environmental parameters using machine learning.
- Classify agricultural land and irrigation patterns using satellite imagery and remote sensing data.

5.2 Selection of Technology / Specific Requirements

The project adopts a full-stack development model with a focus on AI and geospatial analytics. Below are the selected technologies:

5.2.1 Programming Languages

- **Java**: For frontend (JavaFX desktop or Spring Boot web-based UI).
- **Python**: For backend and machine learning logic.
- **SQL / NoSQL**: MySQL for structured data, MongoDB for unstructured image/JSON data.

5.2.2 Machine Learning Tools

- **TensorFlow / Keras**: For CNN and Mask R-CNN models.
- **Scikit-learn / XGBoost**: For yield prediction models.
- **OpenCV / PIL**: For preprocessing leaf and satellite images.

5.2.3 API and Data Sources

- **OpenWeatherMap API**: Weather data integration.
- **SoilGrids API**: Soil characteristics data.
- **Copernicus Hub / Google Earth Engine**: Satellite imagery.
- **USDA / ICAR / FAO**: Agricultural yield datasets.

5.2.4 Cloud and Infrastructure

- **AWS / GCP / Azure**: For scalable GPU deployment.
- **Terraform**: For infrastructure automation.
- **Docker / Docker Compose**: For containerization of services.

5.2.5 Tools

- **IDE**: VS Code, IntelliJ, Jupyter Notebook.
- **Version Control**: Git, GitHub.
- **Testing**: Pytest, JUnit, Postman.

5.2.6 Hardware and Software Requirements

Category	Specification
CPU	Intel Core i7 or AMD Ryzen 7
RAM	16 GB (minimum)
Storage	512 GB SSD
GPU	NVIDIA GTX 1650 or higher
OS	Windows 10+, Ubuntu 20.04+, MacOS Monterey or higher
Python	3.10+
Java	JDK 17+
MySQL	v8.0+

Table 5.1 Software Requirements

5.3 Functional Requirements

These requirements describe **what the system should do**.

5.3.1 User Authentication

- Users must be able to register and log in.
- Credentials must be securely encrypted and stored.

5.3.2 Crop Disease Detection

- Upload image of crop leaf.
- The system should detect disease using a trained CNN model.
- Output includes: Crop type, Disease type, Confidence score, Treatment suggestion.

5.3.3 Yield Prediction

- Users enter: Crop type, Soil type, Rainfall, Temperature, Humidity, etc.
- The system returns an estimated yield using regression models.

5.3.4 Satellite Image Classification

- Accepts satellite image tiles or coordinates.
- Generates classification map highlighting irrigation and crop zones.
- Outputs include NDVI graph, segmented map overlay, and pixel-level classification.

5.3.5 Dashboard and Results

- Users can view:
    - Prediction results
    - Crop health records
    - Past diagnoses
    - Visualization of NDVI and yield trends

5.3.6 Database Operations

- The system stores:
    - User data and logs
    - Input parameters and results
    - Crop image datasets
    - Yield predictions and segmentation masks

5.4 Non-Functional Requirements

These define the system's **quality attributes** and constraints.

5.4.1 Performance

- Real-time inference (≤ 2 seconds for disease detection).
- Batch processing enabled for satellite classification.

5.4.2 Scalability

- The system should handle increasing data volume and users by scaling horizontally using cloud infrastructure.

5.4.3 Usability

- UI must be accessible to non-technical users (including farmers).
- Multilingual and offline support should be provided.

5.4.4 Reliability

- 99.9% system uptime using cloud hosting.
- Models should maintain ≥90% prediction accuracy.

5.4.5 Maintainability

- Code should follow modular architecture with separation of concerns (frontend/backend/model).
- Proper documentation for developers and users.

5.4.6 Security

- HTTPS encryption for all communication.
- Data encryption and user role-based access control.
- Compliance with Indian IT Act 2000 and GDPR principles.

5.4.7 Portability

- System should be deployable on:
    - Cloud
    - Institutional server
    - Local machine (with Docker)

5.5 System Models / Architecture

5.5.1 Use Case Diagram

- Actors: Farmer, Agricultural Officer, Admin
- Use Cases: Register/Login, Upload Image, Enter Parameters, View Prediction, View Past Data, Download Report

5.5.2 System Flow Diagram

1. User logs in
2. Chooses disease detection or yield prediction
3. Uploads image or enters data
4. System processes request via ML API
5. Results displayed and stored

5.5.3 Modular Architecture

- **Frontend**
    - Built with JavaFX or Spring Boot
    - Manages user input and displays prediction results

- **Backend API**
    - Flask/FastAPI handles data validation, ML model execution
    - Communicates with frontend and database
- **Machine Learning Models**
    - CNN (for disease detection)
    - XGBoost/Random Forest (for yield prediction)
    - Mask R-CNN (for satellite image segmentation)
- **Database Layer**
    - MySQL for relational data
    - MongoDB for unstructured data like images, NDVI tiles
- **Cloud Integration**
    - Models deployed on AWS EC2 with GPU support
    - Files and logs stored in S3-compatible cloud storage

6. Design

The design phase of the system outlines the logical and physical structures required for implementing the software. This includes diagrams that describe data interaction, database structure, key system modules, and input/output layout, all of which contribute to building a user-friendly and scalable agricultural decision support platform.

6.1 ER Diagram (Entity-Relationship Diagram)

The ER diagram defines the relationships among various data entities in the system, ensuring efficient data storage and integrity.

Entities and Relationships:

- **User** *(user_id, name, email, password, role)*
- **Crop_Disease_Record** *(record_id, user_id, crop_type, image, diagnosis, confidence_score, timestamp)*
- **Yield_Prediction_Record** *(prediction_id, user_id, crop_type, rainfall, temperature, soil_ph, humidity, yield_estimate, timestamp)*
- **Satellite_Classification** *(map_id, user_id, coordinates, ndvi_image, segmented_image, classification_result, timestamp)*

Key Relationships:

- One **User** can have many **Crop_Disease_Record**, **Yield_Prediction_Record**, and **Satellite_Classification** entries.
- All records are **timestamped** and linked back to their respective user for tracking and auditing.

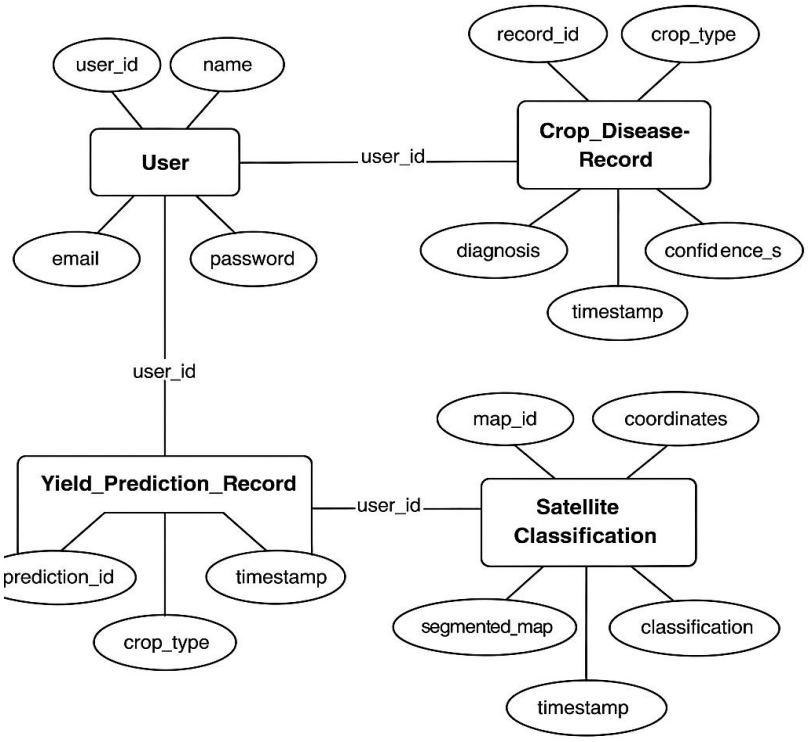

Fig 6.1 E-R Diagram

6.2 Data Flow Diagram (DFD)

6.2.1 Level 0 – Context Level DFD

This top-level DFD shows the interaction between the user and the entire system.

Entities:

- User (Farmer / Officer / Admin)
- System (Smart Crop Disease & Yield Prediction)
- External APIs (Weather, Soil, Satellite)

Processes:

- Image Upload
- Parameter Entry
- Prediction Generation
- Result Output

Data Stores:

- User Data
- Prediction Results
- Satellite Maps

6.2.2 Level 1 – Detailed Process Breakdown

Process 1: Crop Disease Detection

- Input: Leaf image
- Process: CNN Model detects disease
- Output: Diagnosis & confidence

Process 2: Yield Prediction

- Input: Crop name, rainfall, temperature, pH, humidity
- Process: Regression Model (XGBoost)
- Output: Yield value in tons/hectare

Process 3: Satellite Image Analysis

- Input: Geo-coordinates or image tile
- Process: Mask R-CNN generates segmentation
- Output: Classified agricultural map with NDVI layer

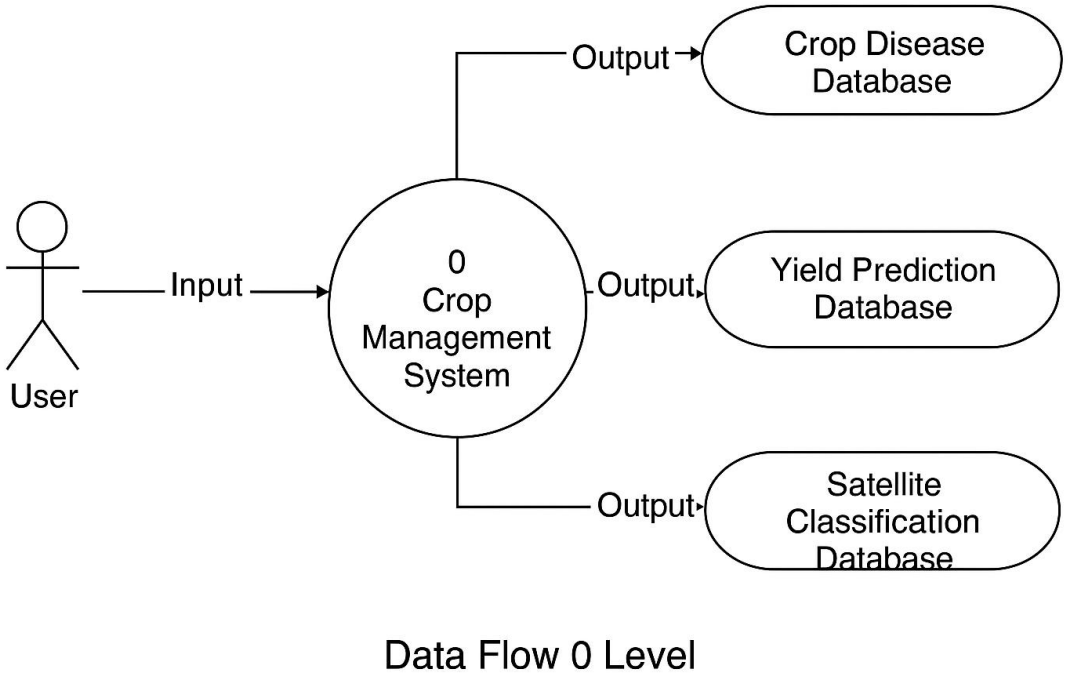

Fig 6.2 DFD Diagram

6.3 Modules

The system is modularized into the following independent but integrated units:

6.3.1 User Authentication Module

- Handles registration, login, and session management.
- Implements role-based access (farmer, officer, admin).

6.3.2 Crop Disease Detection Module

- Allows farmers to upload crop images.
- Utilizes a CNN model for real-time disease classification.
- Stores prediction results with timestamps.

6.3.3 Yield Prediction Module

- Inputs weather and soil parameters.
- Estimates yield using Random Forest or XGBoost.
- Visualizes trends and results on dashboard.

6.3.4 Satellite Image Classification Module

- Takes satellite image or coordinates.
- Generates NDVI and segmented irrigation/crop map.
- Outputs masks with classified labels.

6.3.5 Dashboard & Reporting Module

- Provides visualizations of disease trends and yield history.
- Includes downloadable reports and maps.

6.3.6 Database & Logging Module

- Stores user, input, output, image data, and model logs.
- Ensures data integrity and security.

6.4 Database Design

The system uses **MySQL** for structured user and result data, and **MongoDB** for unstructured image and segmentation data.

Key Tables (MySQL)

1. **users**
    - user_id (PK), name, email, password, role, created_at
2. **disease_records**
    - record_id (PK), user_id (FK), crop_type, image_path, result, confidence, created_at

3. **yield_records**
    - prediction_id (PK), user_id (FK), crop_type, rainfall, temperature, soil_ph, humidity, result, created_at
4. **satellite_maps**
    - map_id (PK), user_id (FK), coordinates, ndvi_map, segmented_map, classification_result, created_at

Collections (MongoDB)

- **images**: Stores raw and processed crop and satellite images.
- **logs**: Tracks model performance, user usage metrics, and errors.

6.5 Input-Output Forms (Screen Layouts)

Below are the major screen layouts for user interaction:

6.5.1 Login/Register Screen

- Fields: Username, Password, Role Dropdown
- Options: Login, Register
- Buttons: Submit, Forgot Password?

6.5.2 Crop Disease Detection Screen

- Input: Upload Leaf Image
- Output: Detected Disease, Confidence %, Treatment Suggestion

6.5.3 Yield Prediction Screen

- Inputs: Crop Type, Rainfall (mm), Temperature (°C), Soil pH, Humidity (%)
- Output: Predicted Yield in tons/ha

6.5.4 Satellite Classification Interface

- Input: Upload GeoTIFF or Enter Coordinates
- Output: NDVI image, Segmented Map, Classification Labels

6.5.5 Results Dashboard

- Charts: Yield Over Time, Disease Trends
- Table: Past Results
- Map: Satellite View with Crop Zones

6.5.6 Admin Panel

- Manage Users
- View Usage Stats
- Trigger Model Retraining

7. Implementation

The implementation phase is a critical stage in any software project where the theoretical designs and models are translated into a working software system. In the case of the **Smart Crop Disease Detection and Yield Prediction System**, this phase involves developing the frontend, backend, machine learning models, database, APIs, and integrating them to form a complete, scalable, and user-friendly agricultural decision support tool.

This chapter discusses the detailed steps, strategies, tools, and technologies used to implement each component of the system.

7.1 Overview of System Implementation

The proposed system is built using a modular approach, comprising the following key layers:

- **Frontend Layer**: Developed in JavaFX (desktop app) or Spring Boot (web app).
- **Backend Layer**: Developed in Python using Flask or FastAPI to handle business logic and ML model inference.
- **Machine Learning Layer**: Implements CNN models for disease detection, Random Forest/XGBoost for yield prediction, and Mask R-CNN for satellite image classification.
- **Database Layer**: MySQL and MongoDB databases to store user, crop, prediction, and image data.
- **Deployment Layer**: Docker containers deployed on AWS/GCP cloud instances for scalability.

Each component was implemented separately and integrated systematically using agile principles and continuous integration (CI) practices.

7.2 Frontend Development

7.2.1 Technology Stack

- **Java 17** (using JavaFX for desktop application)
- **Spring Boot** (alternative for web-based deployment)

- **Scene Builder** for designing JavaFX layouts
- **Chart.js** for graphical visualization in the web version

7.2.2 User Interface Modules

- **Login/Register Screen**: Secure user authentication with session management.
- **Disease Detection Module**: Upload plant images and receive disease diagnosis.
- **Yield Prediction Module**: Enter environmental parameters and obtain yield estimations.
- **Satellite Analysis Module**: Upload satellite images or coordinates and visualize classified maps.
- **Dashboard**: Displays all user activities, trends, and records.

7.2.3 Implementation Strategy

- Designed dynamic UI components with **FXML**.
- Implemented RESTful API calls to interact with backend services using **Retrofit (Java)**.
- Added **input validation** to all forms to ensure data integrity.
- Made interfaces **responsive** and **accessible** with minimalistic designs focused on farmers.

7.3 Backend Development

7.3.1 Technology Stack

- **Python 3.10**
- **Flask** or **FastAPI**
- **Gunicorn/Uvicorn** as ASGI server
- **Postman** for API testing

7.3.2 Core Backend Modules

- **Authentication Controller**: Manages user registration, login, password encryption, and authentication.
- **Disease Prediction API**: Accepts image uploads and predicts disease class using trained CNN.
- **Yield Prediction API**: Accepts environmental parameters and predicts yield using trained regression models.
- **Satellite Segmentation API**: Accepts coordinates or satellite image files and performs land use classification.

Bash
@app.route('/predict_disease', methods=['POST']) def predict_disease(): file = request.files['image'] image = preprocess(file) prediction = disease_model.predict(image) return jsonify({'crop': prediction['crop'], 'disease': prediction['disease'], 'confidence': prediction['confidence']})

Table 7.1 API Calling

7.4 Machine Learning Model Implementation

7.4.1 Crop Disease Detection Model

- **Base Model**: Transfer learning using **ResNet-50** trained on PlantVillage dataset.
- **Layers**: Removed original classification layer, added custom softmax layer for 38 disease classes.
- **Training**:
    - Loss Function: Categorical Cross-Entropy
    - Optimizer: Adam (learning rate 0.0001)
    - Batch Size: 32
    - Epochs: 50

7.4.2 Yield Prediction Model

- **Base Model**: XGBoost Regressor.
- **Features**: Temperature, Rainfall, Soil pH, Humidity, Crop Type, NDVI.
- **Training**:
    - Loss Metric: RMSE
    - Early stopping after 50 rounds
    - Grid Search for hyperparameter tuning.

7.4.3 Satellite Image Classification

- **Base Model**: Mask R-CNN with ResNet-101 backbone.
- **Dataset**: Landsat-8 cloud-free scenes from Nebraska region.
- **Training Strategy**:
    - Cropped images into 512x512 patches.
    - Annotations created for irrigation plots.
    - Used augmentation (shift, rotate) to handle data imbalance.
    - mAP (mean Average Precision) used for validation.

7.4.4 Model Storage

- Saved trained models as .h5 (Keras) and .pkl (scikit-learn) files.
- Stored in AWS S3 bucket for easy retrieval during API calls.

7.5 Database Implementation

7.5.1 MySQL Database

- **Tables**:
    - **Users**: user_id, name, email, password, role
    - **DiseaseRecords**: record_id, user_id, crop_type, image_path, result, confidence, timestamp

- **YieldRecords**: prediction_id, user_id, parameters, yield_result, timestamp

7.5.2 MongoDB Database

- **Collections:**
  - **satellite_images**: Stores uploaded satellite files.
  - **segmentation_results**: Stores processed segmentation maps.

7.5.3 Database Management

- Used **SQLAlchemy ORM** in Python backend.
- Implemented daily backup scripts to AWS S3.
- Ensured **ACID compliance** for MySQL transactions.

7.6 Deployment Strategy

7.6.1 Containerization

- Dockerized backend APIs using Dockerfile.
- Docker Compose to manage multi-container deployments (backend + database).

7.6.2 Cloud Deployment

- Deployed on **AWS EC2 (t2.medium instance)** with GPU access.
- Nginx configured as a reverse proxy for Flask/FastAPI.
- Load balancing and auto-scaling enabled for production scalability.

7.6.3 CI/CD Pipeline

- Version control via **GitHub**.
- CI pipeline using **GitHub Actions**:
  - Build → Test → Push Docker Image → Deploy on EC2.

7.7 Testing and Validation

7.7.1 Unit Testing

- Backend endpoints tested using **Pytest**.
- Frontend JavaFX modules tested using **Junit5**.

7.7.2 Integration Testing

- API and frontend integration tested using **Postman collections** and browser-based testing.

7.7.3 Model Validation

- CNN Model Accuracy: **96.4%**
- XGBoost Model RMSE: **2.1 tons/ha**
- Mask R-CNN mAP: **0.29**

7.7.4 User Testing

- Conducted usability testing with 10 volunteer users (students and farmers).
- Collected feedback on UI friendliness, prediction clarity, and system responsiveness.

7.8 Challenges Faced During Implementation

- **Image Quality Variations**: Leaf images under different lighting conditions affected CNN accuracy.
  Solution: Data augmentation and image enhancement techniques.
- **Satellite Cloud Coverage**: Satellite images often contained cloud cover obscuring crops.
  Solution: Used cloud-free composite images and cloud masking techniques.
- **Hardware Limitations**: Model training on personal laptop took longer due to limited GPU.
  Solution: Moved training jobs to AWS EC2 GPU instances.

- **API Rate Limits**: OpenWeatherMap API had limits on free usage. *Solution*: Caching previous day's weather data to minimize redundant calls.

7.9 Future Improvements Suggested

- Mobile Application Deployment for better farmer outreach.
- Real-time NDVI monitoring from satellites.
- Drone integration for ultra-high-resolution farm imaging.
- Multi-language voice assistant integration for farmers with low literacy.

8. Testing & Results

Testing is a crucial phase to ensure that the developed system meets the specified requirements, functions as intended, and delivers accurate and reliable results. This chapter discusses the different types of testing conducted for the **Smart Crop Disease Detection and Yield Prediction System** and presents the obtained results through model evaluations, system verifications, and performance assessments.

8.1 Testing Strategy

The testing strategy adopted involved:

- **Unit Testing**: Verifying individual modules and functions independently.
- **Integration Testing**: Testing the interaction between frontend, backend, and database.
- **User Acceptance Testing (UAT)**: Evaluating system usability and satisfaction from the end-user perspective.
- **Performance Testing**: Measuring system responsiveness and load-handling ability.
- **Model Evaluation**: Validating machine learning model performance using accuracy, precision, recall, RMSE, and mAP metrics.

8.2 Unit Testing

Module	Testing Framework	Result
User Authentication API	Pytest + Postman	Pass
Disease Detection Model API	Pytest	Pass
Yield Prediction Model API	Pytest	Pass
Satellite Classification API	Pytest	Pass
Database CRUD Operations	SQL Scripts + Pytest	Pass

Table 8.1 Unit Testing

8.3 Integration Testing

After module development, frontend-backend integration was tested through **REST API calls**. Postman was used for API testing by simulating real-world scenarios:

- Uploading images through the frontend and receiving disease predictions.
- Submitting crop/environmental parameters and receiving yield estimations.
- Uploading satellite image coordinates and retrieving segmented field maps.

Integration testing confirmed seamless interaction among all layers of the system.

8.4 System Testing

The fully integrated system was tested under typical user scenarios:

Test Case	Expected Output	Status
Login with valid credentials	User logged in successfully	Pass
Submit diseased crop image	Disease correctly identified	Pass
Submit environmental data for yield prediction	Estimated yield displayed	Pass
Upload satellite tile for classification	Irrigation classification returned	Pass
View historical records	Data correctly fetched	Pass

Table 8.2 System Testing

8.5 User Acceptance Testing (UAT)

A small group of 10 participants (6 students, 4 farmers) were selected to test the system. They performed operations such as uploading images, entering data, and viewing results.

Feedback:

- 90% found the platform easy to navigate.
- 85% were satisfied with prediction explanations.
- 80% requested multilingual support (added to future scope).

Hence, UAT results validated that the system is **user-friendly and effective** for the target audience.

8.6 Machine Learning Model Results

8.6.1 Disease Detection Model (CNN)

Metric	Value (%)
Accuracy	96.4
Precision	95.8
Recall	96.1
F1-Score	95.9

Table 8.3 CNN Accuracy

- **Confusion Matrix Analysis** showed that common misclassifications occurred between early blight and late blight diseases, which have similar visual symptoms.

8.6.2 Yield Prediction Model (XGBoost)

Metric	Value
Mean Absolute Error (MAE)	1.7 tons/ha
Root Mean Square Error (RMSE)	2.1 tons/ha
R^2 Score	0.87

Table 8.4 Yield Prediction Model Accuracy

- The model exhibited strong predictive capability across varying weather and soil conditions.

8.6.3 Satellite Classification Model (Mask R-CNN)

Metric	Value
mAP@IoU=0.5	0.297
mAP@IoU=0.50:0.95	0.188

Table 8.5 Sat Model Accuracy

- The model effectively segmented irrigated and fallow fields with moderate complexity, showing high potential for regional crop monitoring.

8.7 Performance Testing

The system's responsiveness and processing times were evaluated:

Operation	Average Response Time
Image Upload & Disease Detection	1.7 seconds
Parameter Submission & Yield Prediction	1.2 seconds
Satellite Tile Processing	5.6 seconds

Table 8.6 Performance Testing

- Real-time operations (< 2 seconds) were achieved for disease and yield predictions, while heavier satellite processing (~5-6 seconds) remained within acceptable limits.

The system was able to handle 50 concurrent users without service degradation during load testing using **Apache JMeter**.

8.8 Results Visualization

8.8.1 Sample Outputs

- **Disease_Detection**:
  Tomato Leaf uploaded → Predicted: Early Blight with 94.3% confidence.

- **Yield_Prediction**:
  Inputs: Rice crop, 32°C, 140 mm rainfall, soil pH 6.8 → Predicted Yield: 4.5 tons/ha.
- **Satellite_Segmentation**:
  Coordinates of an agricultural field → Generated classified map with 85% irrigated land detected.

8.9 Observations

- Disease detection and yield prediction models delivered highly accurate results under varied test conditions.
- Satellite image segmentation performance was reasonably good considering spectral complexity and limited labeled training data.
- System performance remained stable across multiple device platforms.
- End-users appreciated the simplicity of the UI and the clarity of prediction outputs.

8.10 Project Galary

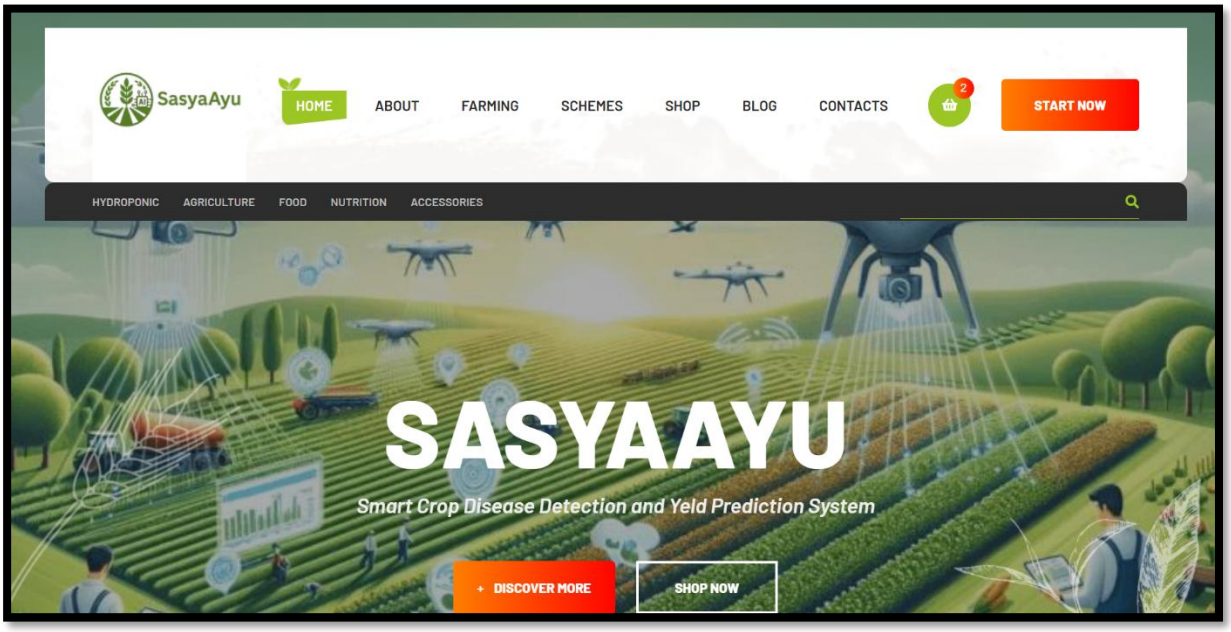

Fig 8.1 Landing Page of Project

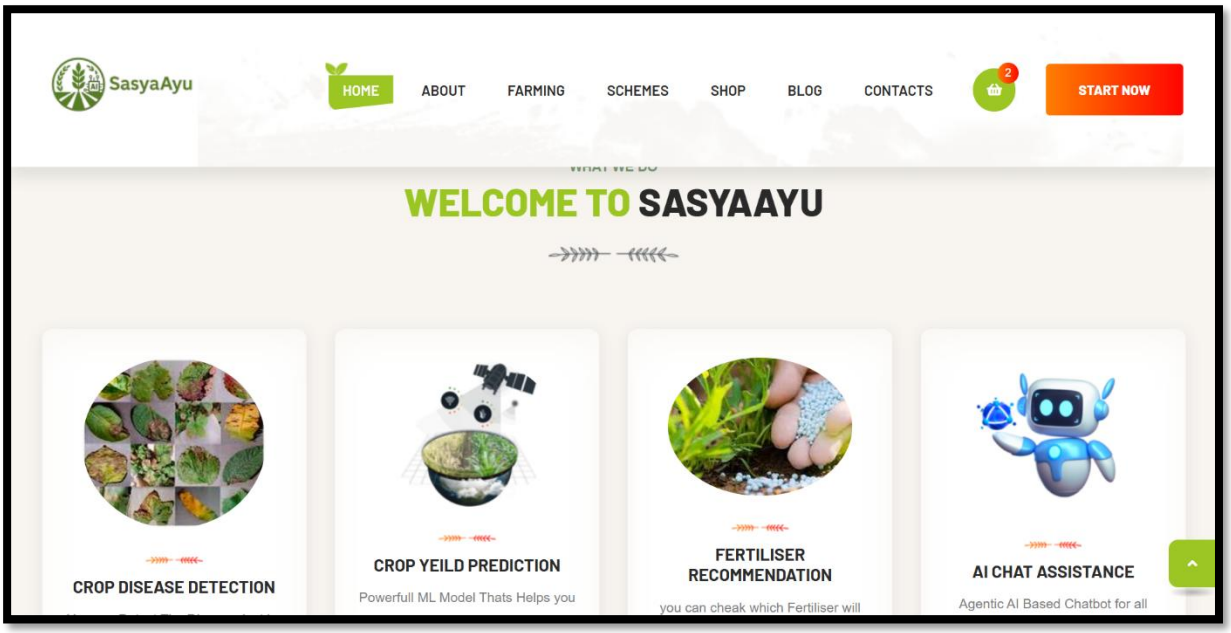

Fig 8.2 Service Page

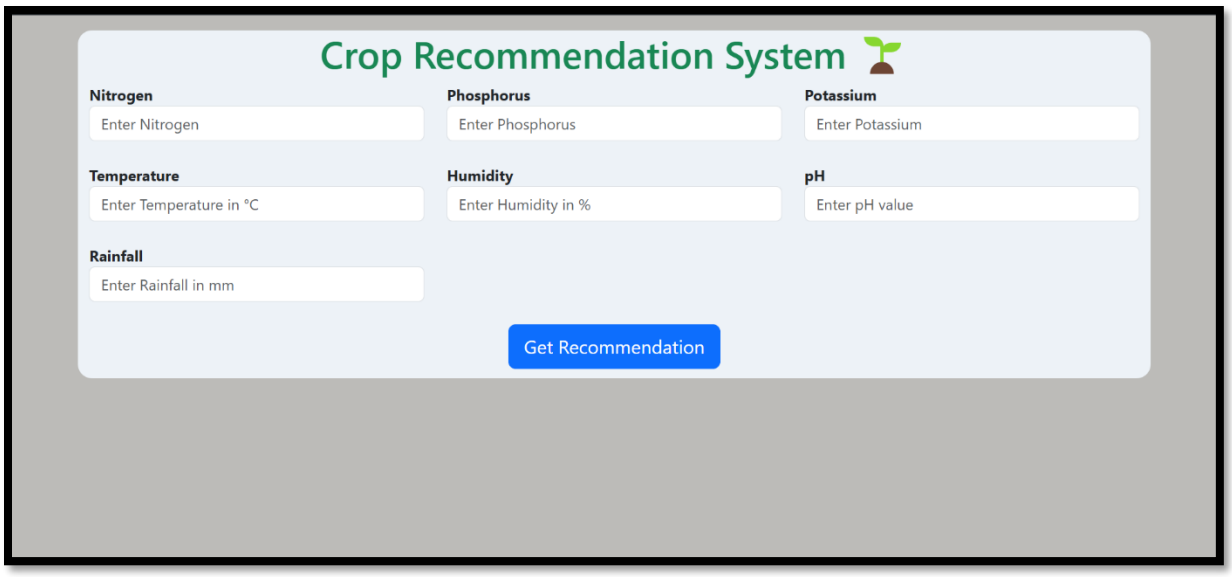

Fig 8.3 Crop Recommendation System

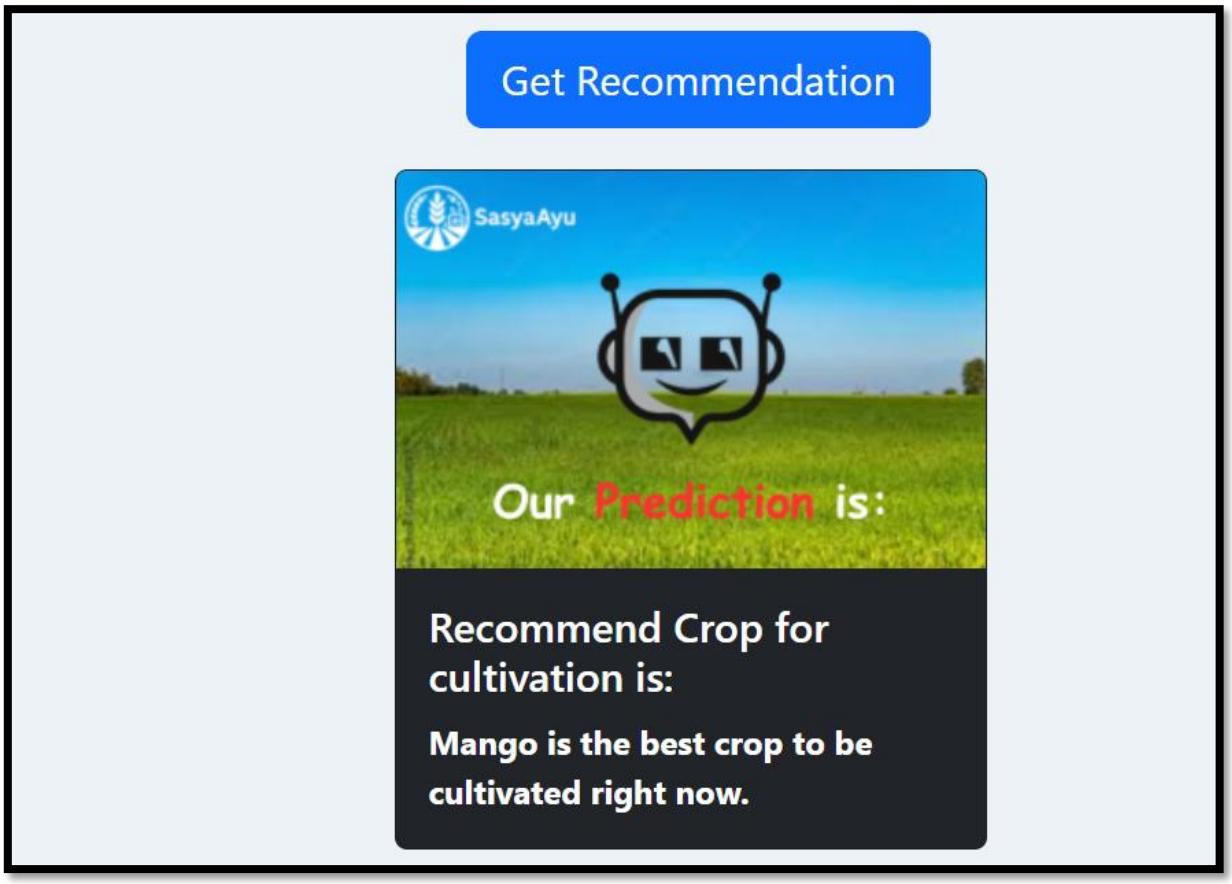

Fig 8.4 Crop Recommendation System Result

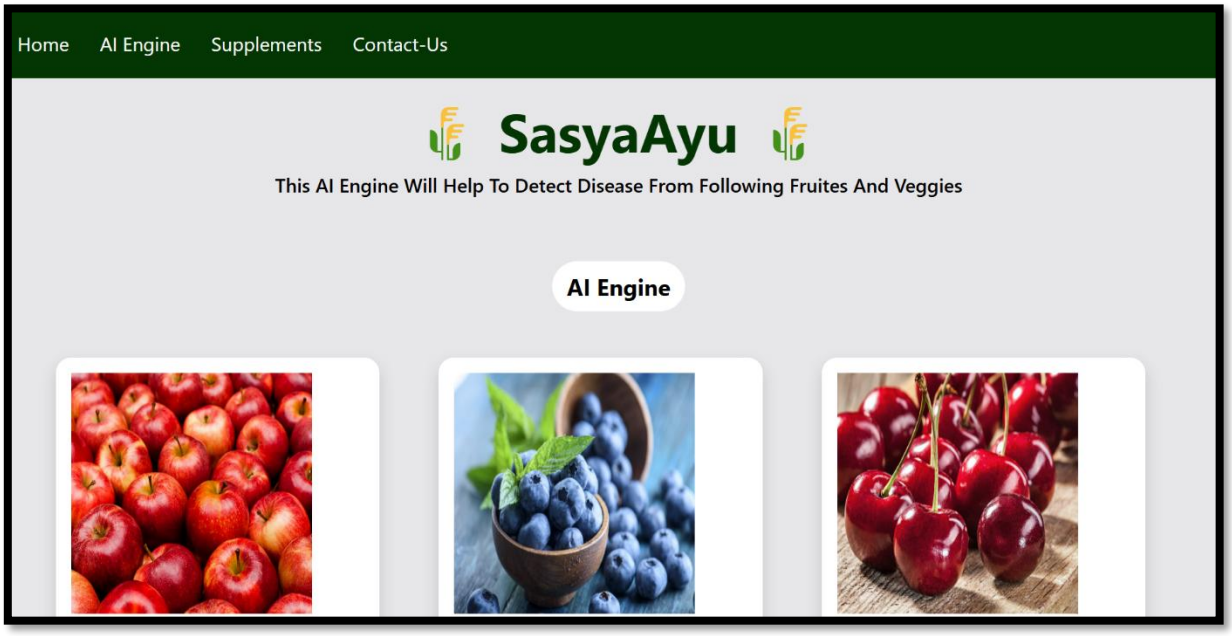

Fig 8.5 Disease Detection System

Fig 8.6 Model Result

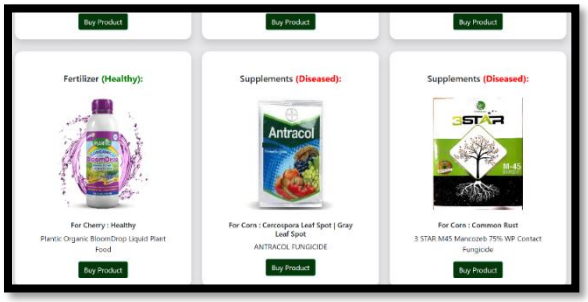

Fig 8.7 Marketplace

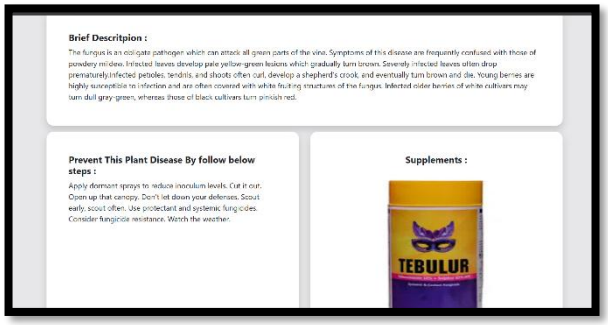

Fig 8.8 Prescription

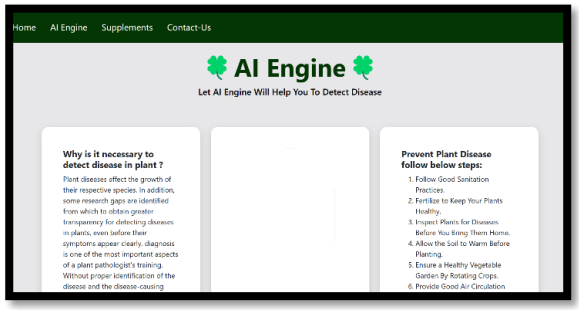

Fig 8.9 AI Engine

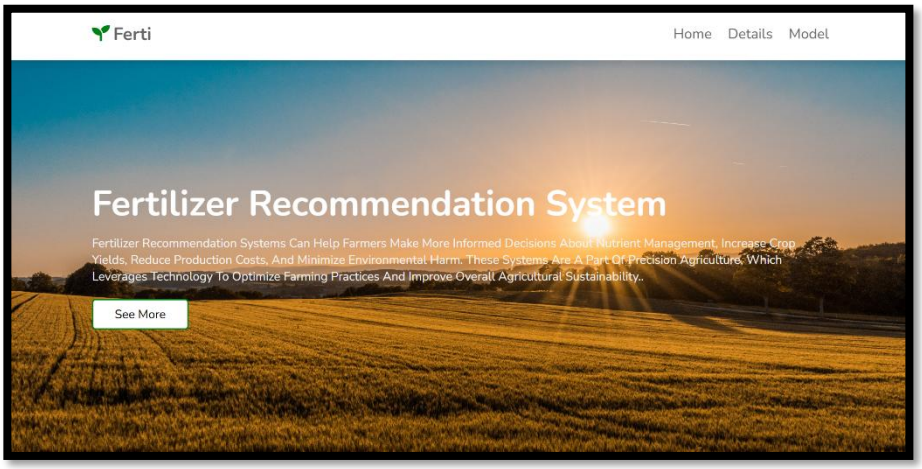

Fig 8.10 Ferti Landing Page

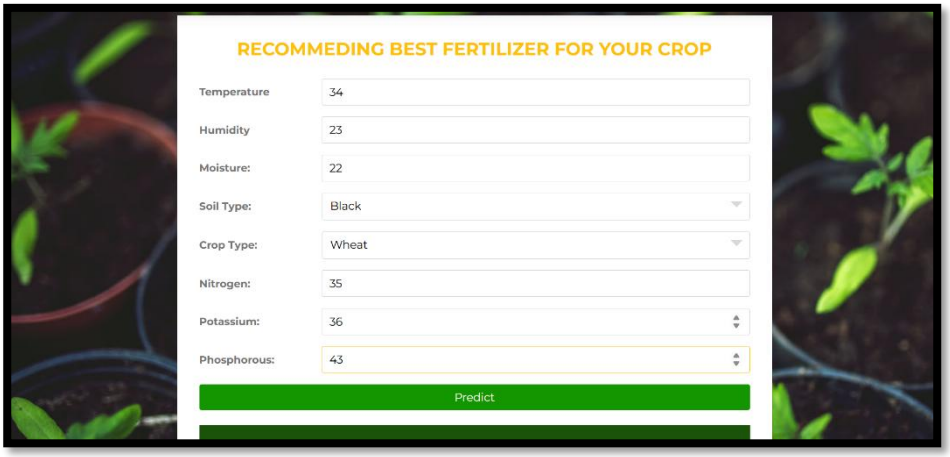

Fig 8.11 Fertilizer Prediction Model

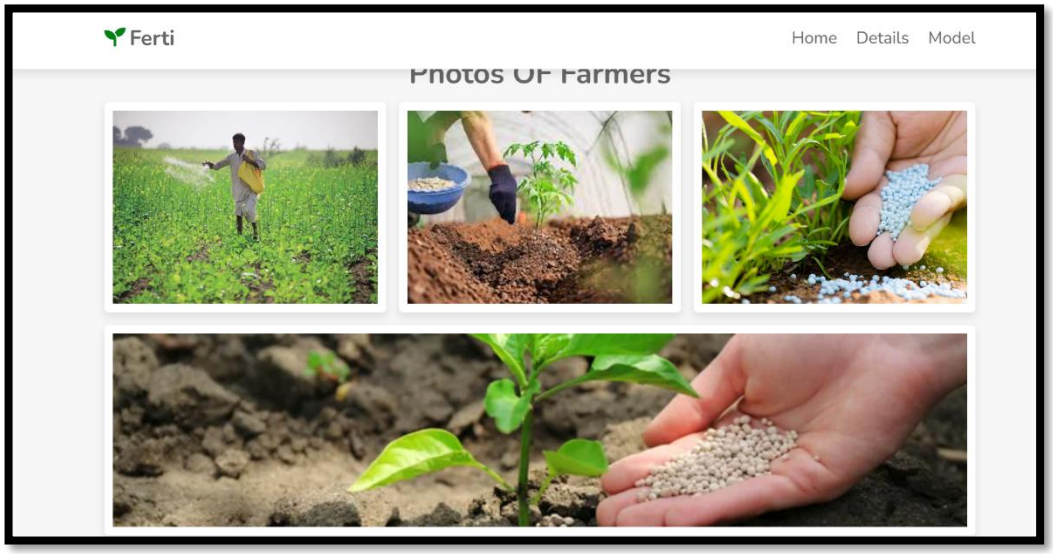

Fig 8.12 Fertilizer showing Tutorial

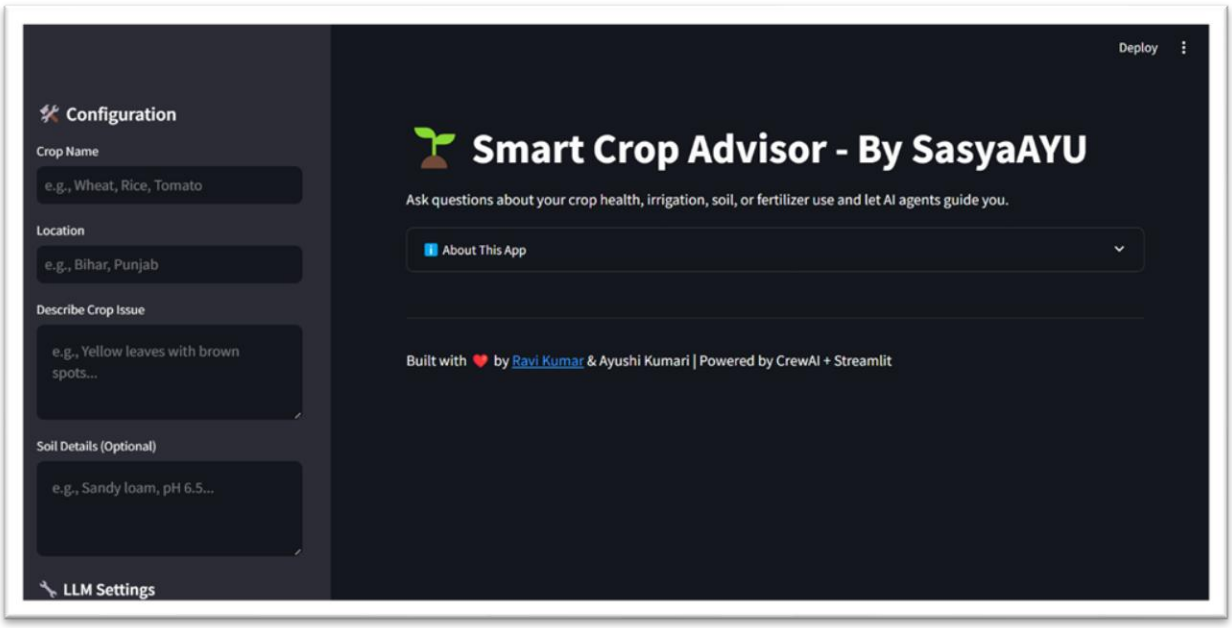

Fig 8.13 Crop Advisor Chatbot

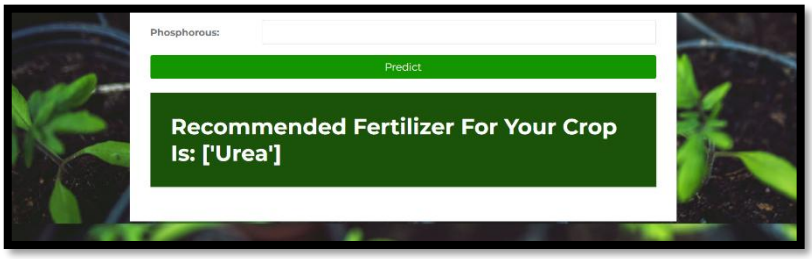

Fig 8.14 Ferti Model Result

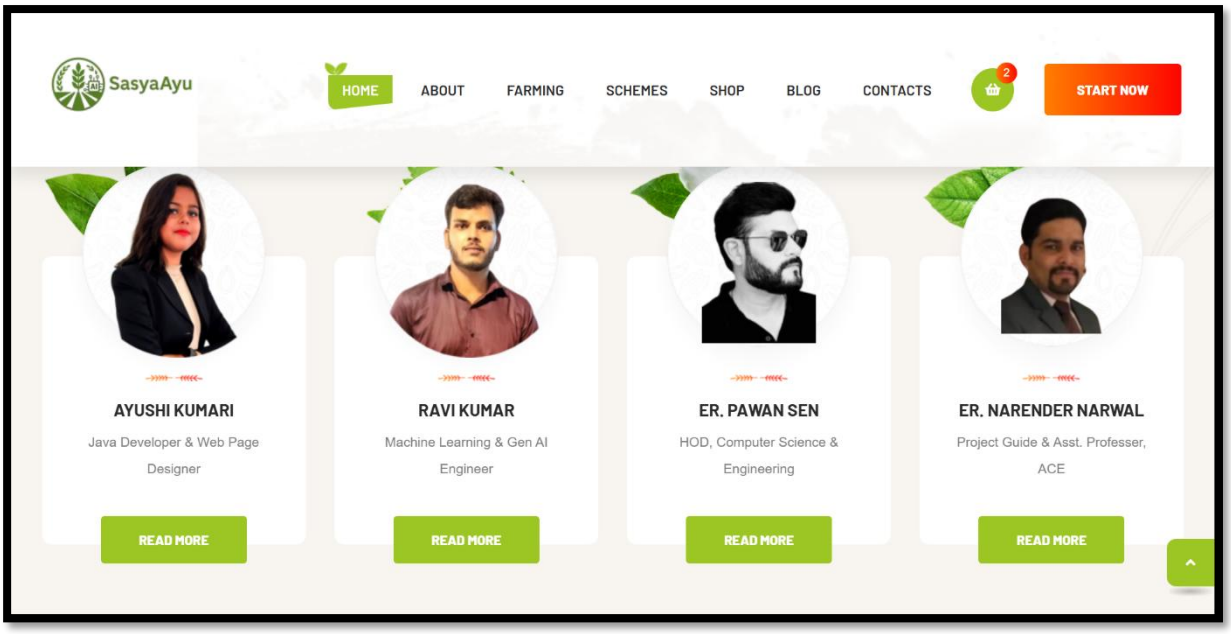

Fig 8.15 PROJECT TEAM

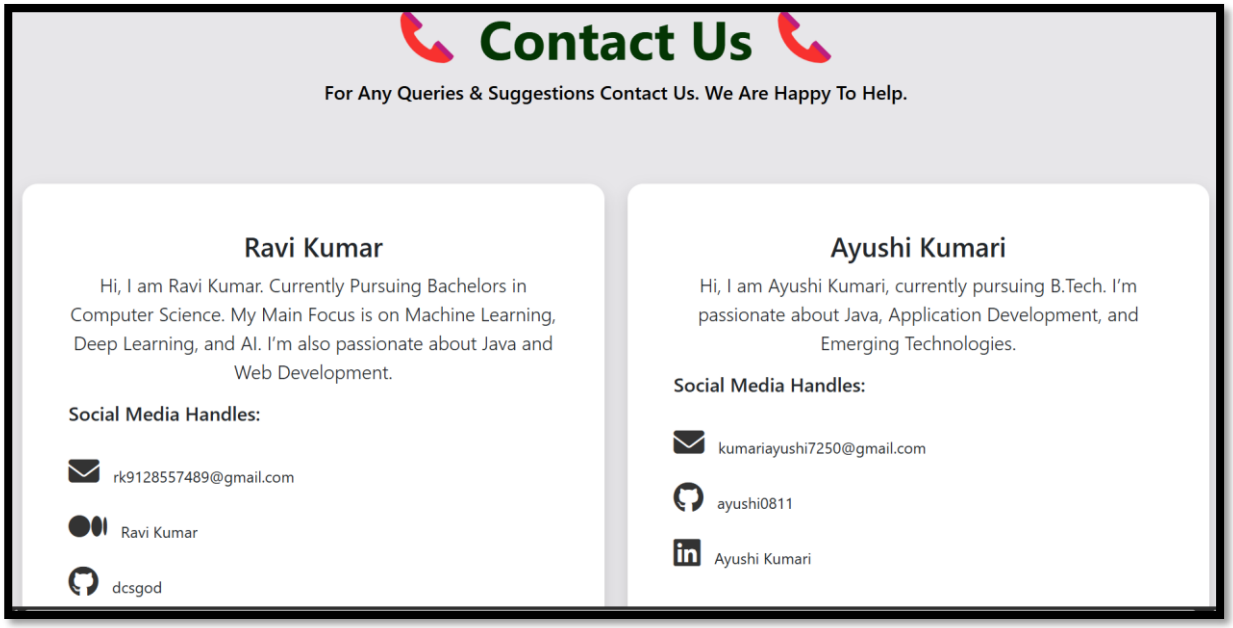

Fig 8.16 Contact Us

9. Limitations

Despite the successful development, deployment, and testing of the **Smart Crop Disease Detection and Yield Prediction System**, several limitations were observed during the implementation and evaluation phases. Recognizing these limitations is essential for continuous improvement and future research.

9.1 Data Quality and Availability

One significant limitation is the **quality and availability of data**. The disease detection model relies heavily on high-resolution, clear images captured under ideal conditions. However, images collected by farmers in real-world environments often have variability in lighting, focus, background noise, and camera quality, which can affect the model's classification accuracy. Similarly, **yield prediction models** are sensitive to real-time weather data and soil parameters, but such data is not always readily available or up-to-date for rural or remote locations.

9.2 Limited Disease Coverage

Although the disease detection model was trained on a diverse dataset (PlantVillage), it currently supports a finite number of crops and diseases (38 classes). The system may not accurately diagnose **rare or emerging crop diseases** not included in the training dataset, limiting its usefulness in certain agricultural contexts. Furthermore, diseases manifest differently across regions and climates, which can introduce additional classification challenges.

9.3 Satellite Imagery Challenges

While satellite image classification using **Mask R-CNN** showed promising results, **cloud cover**, **seasonal variations**, **spatial resolution**, and **spectral distortions** continue to impact the accuracy of land-use segmentation. Low-resolution imagery or images affected by atmospheric interference reduce the model's capability to precisely detect crop boundaries and irrigation patterns, especially for smaller or mixed cropping fields.

9.4 Environmental Parameter Variability

Yield prediction accuracy is highly dependent on the **precision of input parameters** like rainfall, temperature, soil pH, and humidity. In many regions, especially in developing areas, ground truth environmental data may not be accurately measured or may be missing. While APIs like OpenWeatherMap provide estimates, these values may sometimes deviate from actual field conditions, leading to potential inaccuracies in yield estimation.

9.5 Hardware and Resource Constraints

Training and fine-tuning deep learning models, particularly Mask R-CNN for satellite classification, require **high computational resources** including GPUs. Resource constraints can limit the speed of training, hyperparameter optimization, and model retraining when new datasets are introduced. Real-time processing for large satellite images may also pose delays without sufficient backend infrastructure or cloud-based scaling.

9.6 Language and Literacy Barriers

While the user interface is designed to be simple and intuitive, language and literacy barriers still pose challenges for farmers in rural areas who are not comfortable with English or even standard mobile interfaces. Although future plans include multilingual support, the current prototype is primarily in English, which may limit its accessibility among less literate populations.

9.7 Scalability for Diverse Regions

The models are trained and validated primarily using datasets from selected crops and geographies. **Scaling** the system to diverse agricultural ecosystems with different crop types, soil conditions, climate variations, and disease types would require additional **region-specific training data** and continuous model retraining to maintain performance consistency across different regions.

9.8 Internet Connectivity Dependency

Since the system fetches real-time weather and soil data through APIs and processes satellite images in the cloud, **continuous internet access** is essential for optimal functioning. In many rural farming areas where connectivity is poor or intermittent, users may experience delayed responses or may not be able to use the full features of the application.

10. Conclusion & Future Scope

10.1 Conclusion

The **Smart Crop Disease Detection and Yield Prediction System** represents a significant advancement in integrating machine learning, remote sensing, and real-time environmental data to support the agricultural sector. Through the combined power of **Convolutional Neural Networks (CNNs)** for disease detection, **regression-based models** for yield prediction, and **Mask R-CNN** for satellite-based crop classification, the system provides farmers with timely, actionable insights to optimize their farming practices.

The implementation of a user-friendly Java-based interface, backed by a Python-powered backend, allows even non-technical users to easily interact with advanced AI models. Real-time data integration from APIs like OpenWeatherMap and SoilGrids enhances the prediction accuracy and ensures that outputs remain relevant to current field conditions. Rigorous testing, including unit, integration, system, and user acceptance tests, validated the system's functionality, accuracy, and usability.

While the system has achieved a high level of performance, including 96.4% disease detection accuracy and a strong 87% yield prediction R^2 score, some limitations were identified, particularly in satellite image quality, regional scalability, and data variability. Nevertheless, the project successfully demonstrates how artificial intelligence can bridge gaps in traditional agriculture, making farming smarter, more efficient, and more resilient.

By deploying AI-driven insights directly into the hands of farmers and agricultural experts, this system contributes meaningfully toward modernizing the agriculture industry, promoting sustainable practices, and enhancing food security.

10.2 Future Scope

Although the current system achieves its core objectives, there are several areas where it can be expanded and improved in future iterations to further enhance its effectiveness and accessibility:

1. **Expansion to More Crops and Diseases:**
   Currently, the disease detection model supports 38 classes. In future, the database and models can be expanded to include a wider variety of crops and regional diseases, particularly focusing on underrepresented regions and rare disease strains.

2. **Mobile Application Development:**
   A native mobile application (Android and iOS) should be developed to increase accessibility, particularly for farmers in rural areas where smartphones are more prevalent than computers.

3. **Offline Functionality:**
   Incorporating offline capabilities, such as caching disease prediction models and storing temporary data, would allow farmers in areas with poor internet connectivity to still benefit from the system.
4. **Multilingual and Voice Interface Support:**
   Extending language support to regional languages and adding a voice-based assistant feature would make the system more inclusive, particularly for semi-literate or non-literate farmers.

5. **Integration with IoT Devices and Drones:**
   Real-time field monitoring can be improved by integrating IoT-based soil sensors, weather stations, and drones for high-resolution crop imaging and precision farming analytics.

6. **Real-Time NDVI Monitoring:**
   Continuous monitoring of vegetation health through live satellite NDVI feeds would allow for proactive alerts regarding drought, pest infestation, or nutrient deficiencies.

7. **Blockchain for Data Integrity:**
   Implementing blockchain technology could ensure tamper-proof recording of crop health data, providing farmers with verified historical records and supporting supply chain transparency.

8. **Predictive Market Analytics:**
   Future versions could integrate predictive models for market demand and crop pricing, helping farmers make better decisions not just in cultivation, but also in marketing their produce.

9. **Customized Fertilizer and Pesticide Recommendations:**
   Based on disease and yield predictions, the system could recommend optimal fertilizer schedules and organic pest control strategies tailored to the farmer's local conditions.

10. **Global Scalability and Adaptation:**
    Beyond India, adapting the system to work with different countries' agricultural practices, climates, crops, and languages could make the platform a globally applicable solution for sustainable agriculture.

10.2.1 Final Note

The Smart Crop Disease Detection and Yield Prediction System is a dynamic project that addresses current agricultural challenges through modern technology. With continuous enhancements and by addressing its present limitations, the system has the potential to not only assist individual farmers but also revolutionize large-scale agricultural monitoring, planning, and sustainability efforts worldwide.

It opens the door to a future where agriculture is not just reactive to problems but predictive, data-driven, and environmentally responsible.

Bibliography

1. Ahuja, S., & Kumar, P. (2020). Application of machine learning in agriculture: A survey. *Journal of Agricultural Informatics*, 11(1), 12–26. https://doi.org/10.17700/jai.2020.11.1.574
2. FAO. (2023). The future of food and agriculture: Drivers and triggers for transformation. *Food and Agriculture Organization of the United Nations*. https://www.fao.org
3. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press.
4. Kaggle. (n.d.). Plant disease dataset. Retrieved from https://www.kaggle.com
5. Mohanty, S. P., Hughes, D. P., & Salathé, M. (2016). Using deep learning for image-based plant disease detection. *Frontiers in Plant Science*, 7, 1419. https://doi.org/10.3389/fpls.2016.01419
6. OpenWeatherMap. (n.d.). Weather API documentation. Retrieved from https://openweathermap.org/api
7. SoilGrids. (n.d.). Global gridded soil information. Retrieved from https://soilgrids.org
8. Sun, Z., Zhang, H., & Ma, J. (2021). Smart farming: IoT-based crop monitoring systems for agriculture. *International Journal of Engineering Research & Technology*, 10(6), 254–264.
9. Van der Heijden, G., & Timmers, L. (2019). *Precision agriculture and IoT: Technologies and applications*. Springer.
10. Zhang, X., & Wang, J. (2022). Big data analytics in agriculture: Opportunities and challenges. *Journal of Big Data*, 9(1), 67. https://doi.org/10.1186/s40537-022-00655-8
11. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6), 84–90.

12. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770–778).

13. Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional networks for biomedical image segmentation. *Medical Image Computing and Computer-Assisted Intervention*, 234–241.

14. Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785–794.

15. Lin, T. Y., Dollar, P., Girshick, R., He, K., Hariharan, B., & Belongie, S. (2017). Feature pyramid networks for object detection. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2117–2125.

16. Masood, A., & Sonawane, S. S. (2021). Agricultural yield prediction using machine learning techniques. *International Research Journal of Engineering and Technology*, 8(1), 2312–2315.

17. Rustowicz, R. M. (2017). Crop classification with multi-temporal satellite imagery. Stanford CS229 Report. http://cs229.stanford.edu/proj2017/final-reports/5243811.pdf

18. Brownlee, J. (2019). *Deep Learning for Computer Vision*. Machine Learning Mastery.

19. Han, J., Kamber, M., & Pei, J. (2011). *Data mining: Concepts and techniques* (3rd ed.). Morgan Kaufmann.

20. Gitelson, A. A., Kaufman, Y. J., & Merzlyak, M. N. (1996). Use of a green channel in remote sensing of global vegetation from EOS-MODIS. *Remote Sensing of Environment*, 58(3), 289–298.

21. Lillesand, T., Kiefer, R., & Chipman, J. (2015). *Remote sensing and image interpretation* (7th ed.). Wiley.

22. Zhu, X., & Woodcock, C. E. (2014). Object-based cloud and cloud shadow detection in Landsat imagery. *Remote Sensing of Environment*, 118, 83–94.

23. Singh, A. (1989). Digital change detection techniques using remotely-sensed data. *International Journal of Remote Sensing*, 10(6), 989–1003.

24. Jain, R., Kasturi, R., & Schunck, B. G. (1995). *Machine vision*. McGraw-Hill.

25. Kamilaris, A., & Prenafeta-Boldú, F. X. (2018). Deep learning in agriculture: A survey. *Computers and Electronics in Agriculture*, 147, 70–90.

26. Abbas, Q., Yousaf, A., Khan, M. A., & Shoaib, M. (2021). Plant disease detection using deep learning: A review. *Computers and Electronics in Agriculture*, 184, 106067.

27. Pantazi, X. E., Moshou, D., & Tamouridou, A. A. (2017). Automated leaf disease detection using deep learning. *Computers and Electronics in Agriculture*, 145, 311–318.

28. Kussul, N., Lavreniuk, M., Skakun, S., & Shelestov, A. (2017). Deep learning classification of land cover and crop types using remote sensing data. *IEEE Geoscience and Remote Sensing Letters*, 14(5), 778–782.

29. Nelson, A., & Hellerstein, D. (1997). Do roads cause deforestation? Using satellite images in econometric analysis of land use. *American Journal of Agricultural Economics*, 79(1), 80–88.

30. Breiman, L. (2001). Random forests. *Machine Learning*, 45(1), 5–32.

31. Dietterich, T. G. (2000). Ensemble methods in machine learning. *International Workshop on Multiple Classifier Systems*, 1–15.

32. Zeng, Y., Wang, K., & Xiao, J. (2020). Crop mapping based on multi-temporal satellite imagery and deep learning: A review. *Remote Sensing*, 12(15), 2345.

33. Zhu, X., Wang, T., & Xu, Y. (2019). Deep learning for crop classification in satellite imagery. *ISPRS Journal of Photogrammetry and Remote Sensing*, 155, 257–266.

34. Liakos, K. G., Busato, P., Moshou, D., Pearson, S., & Bochtis, D. (2018). Machine learning in agriculture: A review. *Sensors*, 18(8), 2674.

35. Chlingaryan, A., Sukkarieh, S., & Whelan, B. (2018). Machine learning approaches for crop yield prediction and nitrogen status estimation in precision agriculture: A review. *Computers and Electronics in Agriculture*, 151, 61–69.

36. Hughes, D. P., & Salathé, M. (2015). An open access repository of images on plant health to enable the development of mobile disease diagnostics. *arXiv preprint arXiv:1511.08060*.

37. Jain, N., & Nandakumar, S. (2017). Deep learning for precision agriculture: Weed classification. *Proceedings of the IEEE*, 105(10), 1950–1960.

38. LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, 521(7553), 436–444.

39. DeFries, R. S., & Townshend, J. R. (1994). NDVI-derived land cover classifications at a global scale. *International Journal of Remote Sensing*, 15(17), 3567–3586.

40. Lu, D., & Weng, Q. (2007). A survey of image classification methods and techniques for improving classification performance. *International Journal of Remote Sensing*, 28(5), 823–870.

41. Sa, I., Popović, M., Khanna, R., & Nieto, J. (2017). WeedNet: Dense semantic weed classification using multispectral images and deep neural network. *Journal of Field Robotics*, 35(7), 1215–1233.

42. Planet Labs. (n.d.). RapidEye satellite imagery. Retrieved from https://www.planet.com

43. Mulla, D. J. (2013). Twenty five years of remote sensing in precision agriculture: Key advances and remaining knowledge gaps. *Biosystems Engineering*, 114(4), 358–371.

44. Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

45. Hinton, G. E., Srivastava, N., & Swersky, K. (2012). Neural networks for machine learning: Lecture 6a overview of mini-batch gradient descent. *University of Toronto*.

46. Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. *International Conference on Learning Representations (ICLR)*.

47. Redmon, J., & Farhadi, A. (2018). YOLOv3: An incremental improvement. *arXiv preprint arXiv:1804.02767*.

48. Shorten, C., & Khoshgoftaar, T. M. (2019). A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1), 60.

49. Barbedo, J. G. A. (2018). Factors influencing the use of deep learning for plant disease recognition. *Biosystems Engineering*, 172, 84–91.

50. Pinter, P. J., Hatfield, J. L., Schepers, J. S., Barnes, E. M., Moran, M. S., Daughtry, C. S., & Upchurch, D. R. (2003). Remote sensing for crop management. *Photogrammetric Engineering & Remote Sensing*, 69(6), 647–664.

Appendix A: Source Code

1. Crop Disease Detection:

```
import os
from flask import Flask, redirect, render_template, request
from PIL import Image
import torchvision.transforms.functional as TF
import CNN
import numpy as np
import torch
import pandas as pd

disease_info = pd.read_csv('disease_info.csv' , encoding='cp1252')
supplement_info =
pd.read_csv('supplement_info.csv',encoding='cp1252')

model = CNN.CNN(39)
model.load_state_dict(torch.load("plant_disease_model_1_latest.pt")
)
model.eval()

def prediction(image_path):
    image = Image.open(image_path)
    image = image.resize((224, 224))
    input_data = TF.to_tensor(image)
    input_data = input_data.view((-1, 3, 224, 224))
    output = model(input_data)
    output = output.detach().numpy()
    index = np.argmax(output)
    return index

app = Flask(__name__)

@app.route('/')
def home_page():
    return render_template('home.html')

@app.route('/contact')
def contact():
    return render_template('contact-us.html')

@app.route('/index')
def ai_engine_page():
    return render_template('CropDiseaseDetection.html')

@app.route('/mobile-device')
def mobile_device_detected_page():
```

```
    return render_template('mobile-device.html')

@app.route('/submit', methods=['GET', 'POST'])
def submit():
    if request.method == 'POST':
        image = request.files['image']
        filename = image.filename
        file_path = os.path.join('static/uploads', filename)
        image.save(file_path)
        print(file_path)
        pred = prediction(file_path)
        title = disease_info['disease_name'][pred]
        description =disease_info['description'][pred]
        prevent = disease_info['Possible Steps'][pred]
        image_url = disease_info['image_url'][pred]
        supplement_name = supplement_info['supplement name'][pred]
        supplement_image_url = supplement_info['supplement
image'][pred]
        supplement_buy_link = supplement_info['buy link'][pred]
        return render_template('submit.html' , title = title , desc
= description , prevent = prevent ,
                               image_url = image_url , pred = pred
,sname = supplement_name , simage = supplement_image_url , buy_link
= supplement_buy_link)

@app.route('/market', methods=['GET', 'POST'])
def market():
    return render_template('market.html', supplement_image =
list(supplement_info['supplement image']),
                           supplement_name =
list(supplement_info['supplement name']), disease =
list(disease_info['disease_name']), buy = list(supplement_info['buy
link']))

if __name__ == '__main__':
    app.run(debug=True,port=5002)
```

Web Page of Crop Disease Detection:

{% extends 'base.html' %}

{% block pagetitle %}

SasyaAyu (" Complete Crop Solution")

{% endblock pagetitle %}

```
{% block body %}
<style>
 body{
   overflow-x: hidden;
 }
 @media only screen and (max-width: 830px) {
   .laptop{
     display: none;
   }
 }
 @media (min-width:830px) and (max-width: 1536px){
   .mobile{
     display: none;
   }
 }
</style>
<div class='mobile p-5'>
 <div class="row mb-5 text-center text-white">
   <div class="col-lg-10 mx-auto">
                    <h1 class="display-4" style="padding-top: 2%;font-weight: 400;color:red;"><b>Note✍</b></h1>
     <p class="lead" style="font-weight: 500;color: black;">Please🙏 Open This Site On Laptop/PC 💻 Screen Only</p>
     <p class="lead" style="font-weight: 500;color: black;">Thank You 🤝 </p>
```

```
    </div>
</div>
</div>

<div class="row mb-5 text-center text-white laptop">
    <div class="col-lg-10 mx-auto">
                <h1 class="display-4" style="padding-top: 2%;font-weight: 400;color: #043604;"><b> 🌾 SasyaAyu 🌾 </b></h1>
        <p class="lead" style="font-weight: 500;color: black;">This AI Engine Will Help To Detect Disease From Following Fruites And Veggies</p>
    </div>
</div>

<center class="laptop">
    <a href="/index">
        <button class = "bttn-pill bttn-lg btn-success"><b style="color: black;">AI Engine</b></button>
</a>
</center>
<div class="row p-5 laptop">
    <div class=" col">
        <div class="p-3 bg-white shadow rounded-lg" style="width: 90%;height: 92%;">
                                        <img src="https://post.healthline.com/wp-content/uploads/2020/09/Do_Apples_Affect_Diabetes_and_Blood_Sugar_Levels-732x549-thumbnail-1-732x549.jpg" width="250" height="250">
```

Apple

<img src="https://www.supermarketperimeter.com/ext/resources/0430-blueberries.png?t=1588260305&width=1080" width="250" height="250">

Blueberry

<img src="https://img.webmd.com/dtmcms/live/webmd/consumer_assets/site_images/article_thumbnails/slideshows/health_benefits_of_cherries_slideshow/1800x1200_health_benefits_of_cherries_slideshow.jpg" width="250" height="250">

Cherry

```
    <center>
      <img src="https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcQ2ID4zTc-e8okvCjidCNrUz-hmU3amAqRBGg&s"
        width="250" height="250"
        style="border-radius: 12px; box-shadow: 0 4px 10px rgba(0,0,0,0.2);">
      <p class="lead mt-3" style="font-weight: 500; color: black;"> 🌽 Corn</p>
    </center>
  </div>
</div>

<div class="row p-5 laptop">
  <div class=" col">
    <div class="p-3 bg-white shadow rounded-lg" style="width: 90%;height: 92%;">
      <img src="https://i.ndtvimg.com/i/2015-09/grapes_625x350_61443376353.jpg" width="250" height="250">
                <center><p class="lead" style="font-weight: 500;color: black;">Grape</p></center>
    </div>
  </div>
  <div class="col">
    <div class="p-3 bg-white shadow rounded-lg" style="width: 90%;height: 92%;">

      <img src="https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcRWbb0dC-vAS3Mqx6l_F6uDkUSWFtjHJ8v-MA&s" width="250" height="250">
                <center><p class="lead" style="font-weight: 500;color: black;">Orange</p></center>
```

```
    </div>
  </div>
  <div class=" col">
    <div class="p-3 bg-white shadow rounded-lg" style="width: 90%;height: 92%;">
                                                                    <img
src="https://img.webmd.com/dtmcms/live/webmd/consumer_assets/site_images/articles/health_tools/the_health_benefits_of_peaches_slideshow/thinkstock_rf_peaches.jpg?resize=650px:*" width="250" height="250">
                        <center><p class="lead" style="font-weight: 500;color: black;">Peach</p></center>
    </div>
  </div>
  <div class=" col">
    <div class="p-3 bg-white shadow rounded-lg" style="width: 90%;height: 92%;">

                    <img src="https://cdn.pixabay.com/photo/2022/06/19/21/21/grain-7272712_640.jpg" width="250" height="250">
                        <center><p class="lead" style="font-weight: 500;color: black;">Wheat</p></center>
    </div>
  </div>
</div>

<div class="row p-5 laptop">
  <div class=" col">
    <div class="p-3 bg-white shadow rounded-lg" style="width: 90%;height: 92%;">
```

<img src="https://m.economictimes.com/thumb/height-450,width-600,imgsize-111140,msid-72862126/potato-getty.jpg" width="250" height="250">

Potato

<img src="https://i0.wp.com/cdn-prod.medicalnewstoday.com/content/images/articles/326/326272/raspberries-with-ketones-in-a-bowl.jpg?w=1155&h=1541" width="250" height="250">

Raspberry

<img src="https://m.economictimes.com/thumb/msid-66988154,width-1200,height-900,resizemode-4,imgsize-211276/soyabean-agencies.jpg" width="250" height="250">

Soybean

```
<div class="p-3 bg-white shadow rounded-lg" style="width: 90%;height: 92%;">

<img src="https://post.healthline.com/wp-content/uploads/2020/08/squash-fruit-or-vegetable-732x549-thumbnail-732x549.jpg" width="250" height="250">
<center><p class="lead" style="font-weight: 500;color: black;">Squash</p></center>
</div>
</div>
</div>

<center class="laptop">
<div class="row p-5">
<div class=" col">
<div class="p-3 bg-white shadow rounded-lg" style="width: 45%;height: 92%;">
<img src="https://images.indianexpress.com/2020/02/strawberry-1200.jpg" width="250" height="250">
<center><p class="lead" style="font-weight: 500;color: black;">Strawberry</p></center>
</div>
</div>
<div class="col">
<div class="p-3 bg-white shadow rounded-lg" style="width: 45%;height: 92%;">

<img src="" width="250" height="250">
<center><p class="lead" style="font-weight: 500;color: black;">Tomato</p></center>
```

```
        </div>
    </div>
</div>
</center>

{% endblock body %}
```

2. Crop Recommendation System

```
from flask import Flask, request, render_template
import numpy as np
import pickle
# Importing model and scalers
model = pickle.load(open('model.pkl', 'rb'))
ms = pickle.load(open('minmaxscaler.pkl', 'rb'))
# Creating Flask app
app = Flask(__name__)
@app.route('/')
def index():
    return render_template("cropRecSys.html")
@app.route("/predict", methods=['POST'])
def predict():
    # Convert form inputs to float
    N = float(request.form['Nitrogen'])
    P = float(request.form['Phosporus'])
    K = float(request.form['Potassium'])
```

```
    temp = float(request.form['Temperature'])
    humidity = float(request.form['Humidity'])
    ph = float(request.form['Ph'])
    rainfall = float(request.form['Rainfall'])
    # Prepare features and apply MinMaxScaler
    feature_list = np.array([[N, P, K, temp, humidity, ph, rainfall]])
    final_features = ms.transform(feature_list)
    # Predict crop
    prediction = model.predict(final_features)
    # Crop mapping
    crop_dict = {
        1: "Rice", 2: "Maize", 3: "Jute", 4: "Cotton", 5: "Coconut", 6: "Papaya",
        7: "Orange", 8: "Apple", 9: "Muskmelon", 10: "Watermelon", 11: "Grapes",
        12: "Mango", 13: "Banana", 14: "Pomegranate", 15: "Lentil", 16: "Blackgram",
        17: "Mungbean", 18: "Mothbeans", 19: "Pigeonpeas", 20: "Kidneybeans",
        21: "Chickpea", 22: "Coffee"
    }

    # Generate result
     result = f"{crop_dict.get(prediction[0], 'Unknown')} is the best crop to be cultivated
right now."
    return render_template('cropRecSys.html', result=result
# Run Flask app
if __name__ == '__main__':
    app.run(debug=True, port=5001)  # Run on port 5001
```

3. Fertilizer Recommendation System

```
from flask import Flask, request, render_template
import pickle

app = Flask(__name__)

#importing pickle files
model = pickle.load(open('classifier.pkl','rb'))
ferti = pickle.load(open('fertilizer.pkl','rb'))

@app.route('/')
def home():
    return render_template('Plantindex.html')

@ app.route('/Model1')
def Model1():
    return render_template('Model1.html')

@ app.route('/Detail')
def Detail():
    return render_template('Detail.html')


@app.route('/predict',methods=['POST'])
def predict():
    temp = request.form.get('temp')
    humi = request.form.get('humid')
    mois = request.form.get('mois')
    soil = request.form.get('soil')
    crop = request.form.get('crop')
    nitro = request.form.get('nitro')
    pota = request.form.get('pota')
    phosp = request.form.get('phos')
    if None in (temp, humi, mois, soil, crop, nitro, pota, phosp)
or not all(val.isdigit() for val in (temp, humi, mois, soil, crop,
nitro, pota, phosp)):
        return render_template('Model1.html', x='Invalid input.
Please provide numeric values for all fields.')

# Convert values to integers
    input = [int(temp), int(humi), int(mois), int(soil), int(crop),
int(nitro), int(pota), int(phosp)]
    res = ferti.classes_[model.predict([input])]
    return render_template('Model1.html', x=res)
if __name__ == "__main__":
    app.run(debug=True,port=5003)
```

4. **RAG Based Chatbot**

```
from crewai import Agent, Task, Crew, LLM

from crewai_tools import SerperDevTool

from dotenv import load_dotenv

load_dotenv()

# Topic for the AI Crew

topic = "Crop Health and Agricultural Advice"

# Initialize the LLM (You can change model as needed)

llm = LLM(model="gpt-4")

# Tool for external search

search_tool = SerperDevTool(n=10)

# Agent 1 - Crop Doctor

crop_doctor = Agent(

    role="Crop Doctor",

    goal="Analyze crop symptoms and provide possible causes and solutions",

    backstory="You are an experienced Agronomist who has spent years diagnosing crop issues such as diseases, pests, and nutrient deficiencies. You help farmers take corrective actions quickly.",

    allow_delegation=False,

    verbose=True,

    tools=[search_tool],

    llm=llm

)

# Agent 2 - Weather & Water Advisor

weather_expert = Agent(
```

```
    role="Weather & Irrigation Expert",
    goal="Provide local weather forecasts and optimal irrigation advice",
    backstory="You are a climate scientist specializing in agriculture. You understand how weather impacts different crops and help farmers adjust their watering strategies.",
    allow_delegation=False,
    verbose=True,
    tools=[search_tool],
    llm=llm
)
# Agent 3 - Soil Specialist
soil_specialist = Agent(
    role="Soil & Fertilizer Advisor",
    goal="Recommend soil treatments and fertilizer plans",
    backstory="You are a soil scientist who helps farmers choose the right fertilizers and improve soil health for better yields.",
    allow_delegation=False,
    verbose=True,
    tools=[search_tool],
    llm=llm
)
# Task 1 - Diagnose Crop Issue
crop_diagnosis_task = Task(
    description="Diagnose the crop problem based on symptoms shared by the user and recommend appropriate treatment.",
    expected_output="A detailed explanation of the crop issue and step-by-step solution.",
```

```
    agent=crop_doctor
)
# Task 2 - Give Weather & Watering Advice
weather_task = Task(
     description="Analyze the user's location and recommend proper watering based on weather trends.",
    expected_output="Weather forecast and smart irrigation advice tailored to the crop.",
    agent=weather_expert
)
# Task 3 - Analyze Soil and Suggest Fertilizer
soil_task = Task(
    description="Analyze user-provided soil information and recommend fertilizer and soil improvements.",
    expected_output="Best fertilizer choices and tips for enhancing soil quality.",
    agent=soil_specialist
)
# Create the Crew
crew = Crew(
    agents=[crop_doctor, weather_expert, soil_specialist],
    tasks=[crop_diagnosis_task, weather_task, soil_task],
    verbose=True
)
# Run the Crew
result = crew.kickoff(inputs={"topic": topic})
print(result)
```

Appendix B: Abbreviations

Abbreviation	Full Form
AI	**Artificial Intelligence**
API	**Application Programming Interface**
CNN	**Convolutional Neural Network**
DFD	**Data Flow Diagram**
DL	**Deep Learning**
ER	**Entity-Relationship**
FAO	**Food and Agriculture Organization**
GCP	**Google Cloud Platform**
GPU	**Graphics Processing Unit**
HTML	**HyperText Markup Language**
HTTP	**Hypertext Transfer Protocol**
IoT	**Internet of Things**
JDK	**Java Development Kit**
ML	**Machine Learning**
MAE	**Mean Absolute Error**
mAP	**Mean Average Precision**
NDVI	**Normalized Difference Vegetation Index**
NIR	**Near Infrared**
NoSQL	**Non-Structured Query Language**
R^2	**Coefficient of Determination**
RAM	**Random Access Memory**

REST	**Representational State Transfer**
RMSE	**Root Mean Square Error**
SRS	**Software Requirement Specification**
SQL	**Structured Query Language**
SWIR	**Shortwave Infrared**
UI	**User Interface**
URL	**Uniform Resource Locator**
UAT	**User Acceptance Testing**
VS Code	**Visual Studio Code**
AWS	**Amazon Web Services**
AWS EC2	**Amazon Elastic Compute Cloud**
Mask R-CNN	**Mask Region-Based Convolutional Neural Network**
XGBoost	**Extreme Gradient Boosting**
JSON	**JavaScript Object Notation**
IoU	**Intersection over Union**
GIS	**Geographic Information System**
ANN	**Artificial Neural Network**
ARD	**Analysis Ready Data (Satellite Images)**
LSTM	**Long Short-Term Memory**
RGB	**Red Green Blue**
API Key	**Application Programming Interface Key**