PRESIDENCY UNIVERSITY

Private University Estd. in Karnataka State by Act No. 41 of 2013

Itgalpura, Rajankunte, Yelahanka, Bengaluru – 560064

AI-Driven Crop Disease Prediction and Management System

A PROJECT REPORT

Submitted by

VISHNU M – 20221COM0084

R AKASH – 20221COM0096

JAGADISH T S – 20221COM0099

Under the guidance of,

MR. MUTHURAJU V

BACHELOR OF TECHNOLOGY

IN

COMPUTER ENGINEERING

PRESIDENCY UNIVERSITY

BENGALURU

DECEMBER 2025

PRESIDENCY UNIVERSITY

Private University Estd. in Karnataka State by Act No. 41 of 2013

Itgalpura, Rajankunte, Yelahanka, Bengaluru – 560064

PRESIDENCY SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

Certified that this project titled "AI-Driven Crop Disease Prediction and Management System" is a bonafide work of "VISHNU M (20221COM0084), R AKASH (20221COM0096), JAGADISH T S (20221COM0099)", who have successfully carried out the project work and submitted the report for partial fulfilment of the requirements for the award of the degree of BACHELOR OF TECHNOLOGY in COMPUTER ENGINEERING during 2025-26.

Mr. Muthuraju V	**Mrs. Benitha Christinal J**	**Dr. Sampath A K**	**Dr. Pallavi R**
Project Guide	Program Project	**Dr. Geetha A**	Head of the Department
PSCS	Coordinator	School Project	PSCS
Presidency University	PSCS	Coordinators	Presidency University
	Presidency University	PSCS	
		Presidency University	

Dr. Shakkeera L	**Dr. Duraipandian N**
Associate Dean	Dean
PSCS	PSCS & PSIS
Presidency University	Presidency University

Name and Signature of the Examiners

1)

2)

PRESIDENCY UNIVERSITY

PRESIDENCY SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

We the students of final year B.Tech in Computer Engineering at Presidency University, Bengaluru, named VISHNU M, R AKASH, JAGADISH T S, hereby declare that the project titled "AI-Driven Crop Disease Prediction and Management System" has been independently carried out by us and submitted in partial fulfillment for the award of the degree of Bachelor of Technology in Computer Engineering during the academic year of 2025-26. Further, the matter embodied in the project has not been submitted previously by anybody for the award of any Degree or Diploma to any other institution.

VISHNU M	USN: 20221COM0084
R AKASH	USN: 20221COM0096
JAGADISH T S	USN: 20221COM0099

PLACE: BENGALURU

DATE: 04 - December 2025

ACKNOWLEDGEMENT

For completing this project work, We/I have received the support and the guidance from many people whom I would like to mention with deep sense of gratitude and indebtedness. We extend our gratitude to our beloved **Chancellor, Pro-Vice Chancellor, and Registrar** for their support and encouragement in completion of the project.

I would like to sincerely thank my internal guide **Mr. Muthuraju V, Assistant Professor**, Presidency School of Computer Science and Engineering, Presidency University, for his moral support, motivation, timely guidance and encouragement provided to us during the period of our project work.

I am also thankful to **Dr. Pallavi R, Head of the Department**, Presidency School of Computer Science and Engineering Presidency University, for her mentorship and encouragement.

We express our cordial thanks to **Dr. Duraipandian N, Dean** PSCS & PSIS, **Dr. Shakkeera L, Associate Dean**, Presidency School of computer Science and Engineering and the Management of Presidency University for providing the required facilities and intellectually stimulating environment that aided in the completion of my project work.

We are grateful to **Dr. Sampath A K, and Dr. Geetha A, PSCS Project Coordinators, Mrs. Benitha Christinal J, Program Project Coordinator**, Presidency School of Computer Science and Engineering, or facilitating problem statements, coordinating reviews, monitoring progress, and providing their valuable support and guidance.

We are also grateful to Teaching and Non-Teaching staff of Presidency School of Computer Science and Engineering and also staff from other departments who have extended their valuable help and cooperation.

VISHNU M

R AKASH

JAGADISH T S

ABSTRACT

Agriculture continues to play a central role in food security and rural livelihoods, yet plant diseases remain one of the most persistent threats to crop productivity. In many regions, farmers rely on personal experience or delayed expert consultation to identify diseases, which often results in inaccurate diagnosis and untimely treatment. Recent developments in mobile computing and artificial intelligence have opened new possibilities for assisting farmers through accessible digital tools. Image-based disease recognition, in particular, has shown potential for reducing crop losses by enabling early and accurate detection.

The present project introduces an AI-Based Plant Disease Prediction System designed to support farmers by providing quick and reliable disease identification using a smartphone application. The system combines a mobile interface built with React Native, a FastAPI backend hosted on a cloud platform, and a trained deep-learning model capable of distinguishing crop leaf diseases. The application guides the user from crop selection to image upload, performs leaf verification, processes the image through the model, and returns a prediction score. Additional checks—such as crop–disease mismatch validation and confidence-threshold filtering—ensure that only meaningful results are displayed. When a disease is positively identified, the system retrieves relevant weather conditions through an external API and presents precautionary and treatment suggestions. These recommendations are generated dynamically, and the user can further consult an agricultural expert through an integrated chat feature. The backend also provides disease-specific medicine information, allowing users to find practical solutions in one platform.

Initial testing demonstrates consistent performance in verifying leaf images, predicting disease classes with acceptable confidence levels, and rejecting ambiguous inputs. The workflow operates smoothly across multiple devices, and the integration of weather data, expert chat, and treatment guidance supports the system's usability in real-world settings. Overall, the project demonstrates a feasible implementation of mobile-assisted disease prediction, highlighting the potential of AI-enabled tools in strengthening agricultural decision-making and reducing preventable crop losses.

Table of Content

List of Figures

List of Tables

List of Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
CNN	Convolutional Neural Network
SDG	Sustainable Development Goal
IoT	Internet of Things
SDLC	Software Development Life Cycle
GPU	Graphics Processing Unit
CPU	Central Processing Unit
UI	User Interface
UX	User Experience
DB	Database
JSON	JavaScript Object Notation
JWT	JSON Web Token
HTTPS	Hypertext Transfer Protocol Secure
REST	Representational State Transfer
ML	Machine Learning
DL	Deep Learning
IDE	Integrated Development Environment
WBS	Work Breakdown Structure
NFR	Non-Functional Requirement
FR	Functional Requirement
CRUD	Create, Read, Update, Delete
APK	Android Application Package
SDG	Sustainable Development Goals
SSL	Secure Sockets Layer
TLS	Transport Layer Security
GPU	Graphics Processing Unit
PaaS	Platform as a Service
UDA	Unsupervised Domain Adaptation
ONNX	Open Neural Network Exchange
V-Model	Verification and Validation Model
LLM	Large Language Model
ORM	Object-Relational Mapping
SSL/TLS	Secure Communication Protocols

CHAPTER 1

INTRODUCTION

Plant diseases continue to be one of the major causes of reduced agricultural productivity, especially in regions where farming is heavily dependent on manual inspection and traditional cultivation practices. When diseases are not detected at an early stage, they spread quickly across fields and significantly affect the overall yield. Farmers often rely on visual observation or informal advice, both of which may not always be accurate. With the increasing availability of mobile devices and internet connectivity in rural areas, there is a growing opportunity to use digital tools to support timely crop-health assessment. Advancements in artificial intelligence and mobile technologies have enabled the development of accessible tools that can interpret leaf images, identify disease patterns, and guide users with suitable treatments. This project is built upon this need to simplify disease detection and make it more accessible to farmers through a user-friendly mobile application.

1.1 Background

Agriculture remains a primary livelihood sector in many parts of India, and the vulnerability of crops to fungal, bacterial, and viral diseases creates substantial economic risks. Early detection is critical because many diseases, such as blight or leaf spot, show subtle symptoms in the beginning. Studies have shown that image-based classification using convolutional neural networks (CNNs) has been effective in identifying plant diseases with high accuracy [1]. Mobile applications powered by AI have also emerged as practical tools for farmers, providing timely insights without requiring specialized equipment [2]. At the same time, cloud-based backends offer scalable processing capabilities, enabling real-time prediction for users even in remote locations. This project draws from these developments and aims to create an integrated solution that assists farmers from disease detection to treatment guidance.

1.2 Statistics

Plant diseases account for a significant portion of agricultural losses globally. According to cross-regional agricultural reports, nearly **20–30% of crop yield losses** are associated with preventable plant diseases when early detection is not available [3]. In India, weather fluctuations and inadequate access to agricultural experts often make farmers more vulnerable to crop damage. Surveys conducted among small-scale farmers indicate that many rely on local knowledge or guesswork to identify diseases, leading to delayed or incorrect responses [4]. Digital advisory systems, however, have shown adoption growth in rural regions due to increased smartphone usage and network penetration. These statistics highlight the need for accessible and accurate disease-prediction tools that operate on mobile devices and support real-time decision-making.

1.3 Prior existing technologies

Several approaches have been used historically for plant disease diagnosis. Traditional systems depend on visual inspection by experts, agricultural officers, or experienced farmers. While effective in some contexts, this method is subjective and limited by human availability. In recent years, mobile-based advisory applications have emerged, some offering image capture and basic symptom matching. Early models applied classical image-processing techniques such as color segmentation and texture analysis. However, these approaches struggled with complex backgrounds and varying lighting conditions [5].

The introduction of deep learning has improved accuracy considerably. Models such as CNNs and transfer-learning-based classifiers have been used to analyze leaf patterns and identify diseases more reliably [6]. Cloud-assisted architectures have also been implemented, where images are uploaded to a server for processing using trained AI models. Despite these advancements, many existing systems lack integrated workflows, such as crop selection verification, meaningful treatment suggestions, expert connectivity, or weather-based contextual advice — gaps which this project aims to address.

1.4 Proposed approach

The aim of this project is to design an **AI-Based Plant Disease Prediction System** that provides farmers with a simple, structured process for detecting and managing crop diseases using their mobile phones. The motivation comes from the need for a reliable and easy-to-use tool that reduces guesswork and offers timely, accurate support.

The proposed application follows a step-wise workflow:

- Users create an account and log in through a mobile application.
- They select the crop type (e.g., tomato or potato) and upload an image of the leaf.

- A preliminary check verifies whether the uploaded image is indeed a leaf image.
- The image is then sent to a backend service, built using **FastAPI** and hosted on a cloud platform.
- The backend processes the image using a trained AI model and returns a **prediction confidence score**.
- When the confidence score is below a set threshold, the system advises the user to retake the image.
- When the detected disease does not correspond to the selected crop, the app alerts the user to correct the crop selection.
- Valid predictions lead to a disease report, weather information related to the crop, and guidance on treatment and precautions.
- Additional features include an in-app expert chat system and a section for recommended medicines provided through the backend.

This approach combines machine-learning-based detection, authentication services, real-time weather integration, and human expert interaction into a unified solution. The system's applications extend to small farmers, agricultural support centers, and educational institutions. Present limitations include dependence on internet connectivity and the need for a sufficiently diverse dataset to improve model generalization.

1.5 Objectives

The project objectives have been defined to align with the expected system behaviour, analysis, security, and deployment requirements:

1. **To design a mobile-based system that can recognize crop leaf diseases from uploaded images using AI-based classification techniques.**
2. **To validate the disease prediction through a structured workflow involving image verification, crop selection checks, and threshold-based confidence evaluation.**
3. **To ensure secure system management by incorporating authenticated user access, structured backend APIs, and controlled data flow between mobile and server components.**
4. **To integrate weather-based contextual information into the disease-prediction results to support more informed agricultural decisions.**
5. **To deploy the system on scalable cloud infrastructure enabling reliable real-time processing for end-users.**

Each objective is measurable and supported by implementation, testing, and verification during development.

1.6 SDGs

The project aligns closely with the United Nations Sustainable Development Goals illustrated. Specifically, the system supports:

- **SDG 2: Zero Hunger** – by promoting healthier crop yield and reducing loss due to preventable diseases [1].
- **SDG 9: Industry, Innovation, and Infrastructure** – through the use of AI, mobile platforms, and cloud systems to modernize agricultural practices.
- **SDG 12: Responsible Consumption and Production** – by helping farmers avoid unnecessary pesticide use through precise diagnosis.
- **SDG 13: Climate Action** – when combined with weather-aware recommendations that help farmers adapt to changing climatic conditions.

These mappings ensure that the project is not only technically meaningful but also socially relevant.

1 NO POVERTY
2 ZERO HUNGER
3 GOOD HEALTH AND WELL-BEING
4 QUALITY EDUCATION
5 GENDER EQUALITY
6 CLEAN WATER AND SANITATION
7 AFFORDABLE AND CLEAN ENERGY
8 DECENT WORK AND ECONOMIC GROWTH
9 INDUSTRY, INNOVATION AND INFRASTRUCTURE
10 REDUCED INEQUALITIES
11 SUSTAINABLE CITIES AND COMMUNITIES
12 RESPONSIBLE CONSUMPTION AND PRODUCTION
13 CLIMATE ACTION
14 LIFE BELOW WATER
15 LIFE ON LAND
16 PEACE, JUSTICE AND STRONG INSTITUTIONS
17 PARTNERSHIPS FOR THE GOALS
SUSTAINABLE DEVELOPMENT GOALS

Fig 1.1 Sustainable development goals [1]

1.7 Overview of project report

This report is organized into nine chapters.

Chapter 1 introduces the context, background, and motivation behind the project. **Chapter 2** presents a detailed literature review summarizing key research contributions related to plant disease prediction and digital agriculture.

Chapter 3 describes the methodology used to develop the system, including model selection and design processes.

Chapter 4 outlines the project-management aspects such as timelines, risks, and budget planning.

Chapter 5 covers the analysis and design of the system through block diagrams, flowcharts, reference-model mappings, and architectural descriptions.

Chapter 6 provides the hardware, software, and simulation details required for implementation.

Chapter 7 contains the evaluation procedures, test points, and results interpretation. **Chapter 8** discusses the social, legal, ethical, sustainability, and safety aspects of the project. **Chapter 9** concludes the report with insights and recommendations for future enhancements.

CHAPTER 2

LITERATURE REVIEW

Research on plant disease detection using artificial intelligence has expanded significantly over the past decade, supported by improvements in computer vision, deep learning architectures, and the availability of structured agricultural datasets. This chapter summarizes ten peer-reviewed research papers in chronological order of publication. Each review focuses on the underlying concepts, methodological choices, results obtained, limitations identified by the authors, and research gaps that are relevant to this project. By synthesizing these findings, the chapter highlights key insights that guide the design of an AI-based plant disease prediction system suited for real-world use through mobile applications.

2.1 Review of Literature

Mohanty et al., 2016 – Deep Learning for Image-Based Plant Disease Detection

Mohanty et al. presented one of the earliest large-scale investigations into using convolutional neural networks (CNNs) for plant disease detection. Using the PlantVillage dataset of approximately 54,000 leaf images covering 38 classes, the authors evaluated AlexNet and GoogLeNet architectures for multi-class classification. Their experiments demonstrated that deep learning models could achieve up to **99.35% classification accuracy** under controlled conditions, establishing a strong case for automated disease recognition. A significant strength of this paper lies in the exploration of data variations, where the dataset was tested in its color, grayscale, and leaf-segmented versions to determine robustness and dependency on background cues. Although the results were promising, the authors acknowledged that the dataset's controlled environment limited real-world applicability. Background uniformity and consistent lighting could artificially inflate accuracy rates. The research highlighted the need for models that handle complex field environments and adapt to real-world noise — a gap this project addresses by incorporating image validation, thresholding, and cross-checking the crop type before prediction.

Ferentinos, 2018 – Deep Learning Models for Plant Disease Detection and Diagnosis

Ferentinos expanded the scope of disease detection by evaluating several deep CNN architectures including AlexNet, GoogLeNet, Overfeat, and VGG on a highly diverse dataset of **87,848 images** spread across 58 plant–disease combinations. The study reported exceptionally high accuracy of **99.53%**, reinforcing the viability of CNNs for large-scale disease diagnosis. Importantly, the dataset incorporated both laboratory and real-field images, offering better generalizability compared to earlier work. The study also compared shallow versus deep CNNs, concluding that deeper architectures substantially outperform traditional machine learning methods. However, limitations persisted in detecting diseases in inconsistent lighting and cluttered backgrounds. Additionally, computational resources required for deploying large CNNs were identified as a barrier for mobile-based applications. The suggested direction for future research emphasized lighter and more efficient models — an idea reflected in this project, where a compact FastAPI-based backend processes images from mobile devices efficiently.

Tan & Le, 2019 – EfficientNet Architecture and Model Scaling

Tan and Le introduced EfficientNet, a family of CNN architectures derived using Neural Architecture Search and compound scaling. Rather than increasing depth, width, or resolution independently, EfficientNet scales all three dimensions uniformly using a compound coefficient. This design achieves significantly higher accuracy while reducing computational cost. As reported, EfficientNet-B7 attained **84.3% ImageNet top-1 accuracy** with 8.4× fewer parameters than previous state-of-the-art models. Although not focused exclusively on plant disease detection, EfficientNet's relevance emerges from its suitability for mobile and cloud-based applications where model size and inference speed are critical constraints. The primary limitation is the computational expense of training EfficientNet variants from scratch, but transfer learning drastically reduces this barrier. This paper's contribution is crucial for real-world agricultural applications requiring lightweight yet accurate models, aligning with the objectives of the present project.

Rodríguez-García et al., 2021 – Ontology-Based Decision Support for Crop Disease Recognition

Rodríguez-García and co-authors proposed a knowledge-driven decision support system for crop pests and diseases using ontologies. Unlike image-based systems, the approach integrates semantic descriptions of symptoms, treatments, and agricultural practices into a unified knowledge base. Their framework emphasizes the role of structured information representation in enhancing interpretability and supporting decision-making. While the system is robust in terms of domain knowledge representation, it depends heavily on manual symptom selection by the user, making it time-consuming and error-prone. The work outlined limitations in scalability and automation, suggesting that integration with image-based AI models could yield a hybrid system. This observation directly informs the design of our project, where image-based predictions are paired with automated treatment recommendations.

Khan et al., 2023 – Apple Leaf Disease Classification Using Xception and Faster-RCNN

Khan et al. introduced a two-stage disease detection pipeline combining Xception for leaf-health classification and Faster-RCNN for disease localization. Tested on multiple apple leaf datasets, their approach improved precision by separating health-status detection from disease-specific identification. The reported accuracy exceeded **97%**, showing strong performance on controlled-environment images. However, the authors noted a significant drop in accuracy when tested on field images due to background clutter and occlusions. The study emphasized the importance of preprocessing, such as segmentation and illumination correction, to improve performance. Their multi-stage architecture is conceptually relevant to our system, which includes a leaf-verification step prior to disease classification to reduce invalid predictions.

Wu et al., 2023 – Unsupervised Domain Adaptation for Plant Disease Recognition in the Wild

Wu et al. tackled the persistent problem of domain shift between laboratory and field images. They proposed MSUN, a Multi-Representation Subdomain Adaptation Network with Uncertainty Regularization, which learns domain-invariant representations using unlabeled target-domain data. The paper identifies three major challenges: (i) large variation between lab and field environments, (ii) high intraclass variability, and (iii) overlapping visual characteristics across diseases. MSUN achieved promising accuracy on real-world datasets such as PlantDoc and Tomato-Leaf-Diseases, outperforming other state-of-the-art UDA techniques. While computational complexity remains high for mobile deployment, the paper stresses the necessity for systems that generalize beyond controlled environments — a core challenge addressed in our project through threshold-based rejection and crop-type validation.

Abbasi et al., 2023 – Crop Diagnostic System for Leafy Greens in Aquaponics

Abbasi and colleagues developed a comprehensive diagnostic system for detecting diseases in aquaponically grown leafy greens. Their model integrates a three-phase classification pipeline alongside an ontology for linking detected diseases to possible causes and treatments. With accuracy levels between **82% and 95%**, the study emphasizes the value of combining AI recognition with knowledge-based reasoning for practical decision-making. A key limitation identified was the absence of large-scale open datasets for aquaponic crops, leading to constrained generalization. Nonetheless, the multi-phase design is methodologically similar to this project, which likewise incorporates validation and expert support features.

Bedi et al., 2024 – PDSE-Lite: Lightweight Plant Disease Severity Estimation

Bedi et al. proposed PDSE-Lite, a lightweight deep learning model for estimating disease severity using convolutional autoencoders and few-shot learning. The method reduces reliance on large labeled datasets and is suitable for deployment in resource-constrained devices. Tested on PlantVillage and other datasets, PDSE-Lite outperformed earlier severity-estimation models while maintaining a compact architecture. The authors noted that real-world adoption remains limited by dataset imbalance and lack of field variability. Their findings reinforce the need for robust preprocessing — a challenge addressed in our system through leaf verification and confidence-based prediction filtering.

Kumar et al., 2024 – Lightweight Deep Learning for Resource-Constrained Environments

Kumar et al. published a comprehensive survey on techniques for optimizing deep learning models for deployment on mobile and embedded devices. The review summarizes methods such as quantization, pruning, knowledge distillation, and efficient architectures like MobileNet, ShuffleNet, and EfficientNet. The authors argue that reducing parameter count does not always correlate with faster inference due to memory-access constraints. Their analysis shows that combinations of architecture design and compression techniques yield the best trade-offs for real-time applications. This survey is highly relevant to this project, which requires fast, reliable inference via a cloud-hosted FastAPI backend serving a mobile application.

Ali et al., 2024 – Ensemble Deep Learning for Plant Disease Classification

Ali et al. proposed an ensemble of deep learning models including DenseNet201, EfficientNetB0, EfficientNetB3, and Inception-ResNet-V2 to improve classification accuracy on the New PlantVillage dataset. Their ensemble approach achieved **99.89% accuracy**, outperforming individual architectures. They attributed this gain to diversified feature extraction across models. Limitations included high computational cost and poor suitability for mobile devices. The study recommends more efficient ensembles or knowledge distillation as alternatives. The findings support our design decision to maintain a single efficient deep learning model to ensure speed and scalability, instead of ensemble-based approaches.

2.1 Summary of Literatures reviewed

Table 2.1 Summary of Literature reviews

Article Title, Published Year, Journal Name	Methods	Key Features	Merits	Demerits
Using Deep Learning for Image-Based Plant Disease Detection, 2016, Frontiers in Plant Science	Deep CNN (AlexNet, GoogLeNet)	Large-scale (54k images) multi-crop disease classification; experiments with grayscale, segmented, and color datasets.	Very high accuracy (99.35%); establishes deep learning feasibility for agriculture.	Dataset captured in controlled lab settings; model fails to generalize well to real-field environments.
Deep Learning Models for Plant Disease Detection and Diagnosis, 2018, CEAgri	CNN architectures (AlexNet, VGG, GoogLeNet)	58 disease classes; comparisons across multiple deep CNNs; includes field + lab images.	Robust model evaluation; deeper networks show superior performance.	Heavy computation; unsuitable for real-time or mobile usage without optimization.
EfficientNet: Rethinking Model Scaling, 2019, ICML	EfficientNet scaling (compound scaling)	Balanced scaling of depth, width, resolution; EfficientNet-B0–B7 models.	High accuracy with far fewer parameters; suitable for constrained devices via transfer learning.	Training EfficientNet from scratch requires high compute; architecture complexity is high.
Ontology-Based Decision Support for Crop Pests and Disease Recognition, 2021, AI in Agriculture	Ontology + semantic reasoning	Knowledgebase for symptoms, causes, and treatments; rule-based reasoning engine.	High interpretability; supports expert decision-making.	Requires manual symptom selection; no image-based automation.
Apple Leaf Disease Detection Using Xception and Faster-RCNN, 2023	Xception + Faster-RCNN	Two-stage pipeline for leaf health detection and disease localization.	High accuracy for controlled images; good spatial localization.	Performance drops in cluttered backgrounds; needs preprocessing.
Unsupervised Domain Adaptation for Plant Disease Recognition in the Wild, 2023, Plant Phenomics	MSUN (UDA + CNN + uncertainty regularization)	Addresses domain shift between lab and field; uses unlabeled field data.	Strong generalization to field datasets; reduces overfitting to lab images.	Computationally heavy; UDA requires complex training loops.
Crop Diagnostic System for Leafy Greens in Aquaponics, 2023, AIIA	Multi-phase CNN + ontology	Three-stage classifier; causes & treatment retrieval using ontology (AquaONT).	Integrates reasoning with visual detection; useful in controlled farms.	Dataset size limited; specific to aquaponics (not general crops).
PDSE-Lite: Lightweight Severity Estimation, 2024, Frontiers in Plant Science	CNN + Convolutional Autoencoder + Few-shot learning	Lightweight model; designed for severity estimation with minimal data.	Low compute cost; suitable for edge devices.	Performance sensitive to class imbalance; limited real-field testing.
Lightweight DL for Resource-Constrained Environments, 2024, ACM	Survey of DL optimization (quantization, pruning, KD)	Comprehensive evaluation of lightweight architectures for edge deployment.	Clear guidance on optimizing real-world models; identifies best techniques.	Survey paper—does not propose new architecture; no experimental results.
Ensemble Deep Learning for Plant Disease Classification, 2024, Ecological Informatics	Ensemble DL (DenseNet201 + EfficientNetB0/B3 + Inception-ResNet-V2)	Multi-model ensemble approach; enhanced feature diversity.	Highest accuracy among modern classifiers (≈99.89%).	Very high computational cost; unsuitable for mobile and real-time inference.

CHAPTER 3

METHODOLOGY

3.1 Introduction

This chapter presents the methodology adopted for developing the AI-Based Plant Disease Prediction System. The methodology outlines the structured workflow followed from requirements gathering to model development, backend integration, and mobile application deployment. The project adopts the **V-Model methodology** as the primary framework since it offers a clear mapping between development activities and their corresponding testing phases. In addition to this, concepts from the **Software Development Life Cycle (SDLC)** and **Agile practices** are followed wherever iterative improvements and refinements were required.

The combination of these approaches ensures completeness, systematic verification, and adaptability throughout the project. This hybrid methodology is particularly suitable for an AI-based system that includes multiple dependent components such as mobile interfaces, cloud backend services, deep-learning models, and database authentication.

3.2 The V-Model Methodology

The **V-Model**, also referred to as the **Verification and Validation model**, extends the traditional waterfall approach by incorporating parallel testing phases at each development stage. The left side of the “V” represents analysis and system design, while the right side corresponds to verification, validation, and testing. This is appropriate for systems that require precision and structured development such as AI-based diagnostics, where output needs to be validated at each step.

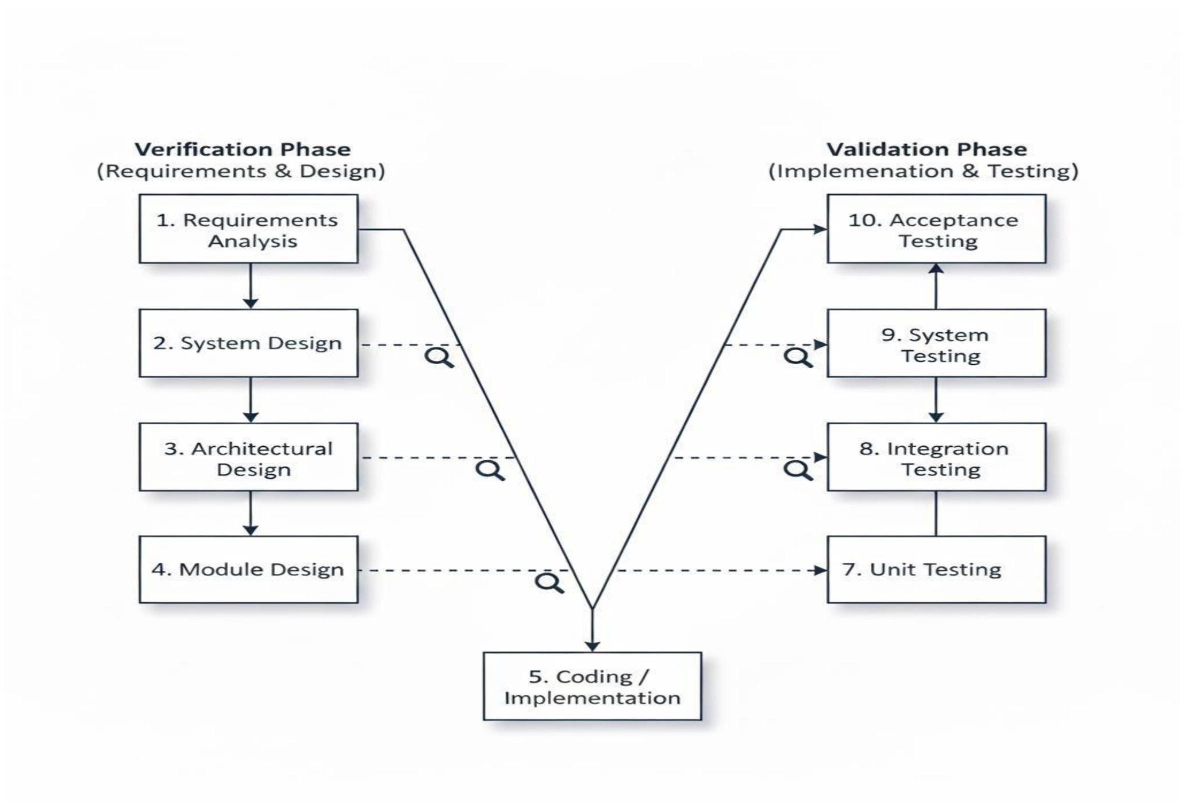


Fig 3.1 The V-Model Methodology

The figure illustrates the V-Model methodology, a structured development lifecycle that emphasizes the relationship between each development phase and its corresponding testing phase. The left side of the "V" represents Validation activities, beginning with Requirements Analysis, followed by System Design and Architecture Design, all moving downward toward Implementation. The right side represents Verification activities, where each testing phase—Integration Testing, System Testing, and Acceptance Testing—corresponds directly to its matching design activity on the opposite side. This ensures that every design stage is validated by a specific test stage, promoting early detection of defects, improved traceability, and higher software reliability.

Explanation in Context of the Project

- **Requirements Analysis:**
  Functional needs were gathered such as crop selection, image upload, leaf validation, disease prediction, weather retrieval, treatment suggestions, and expert chat.

- **System Design:**
  High-level workflow including mobile interface, FastAPI backend, AI model, and Supabase authentication was designed.
- **Architectural Design:**
  Backend endpoints, database structure, and AI inference pipeline were planned.
- **Module Design:**
  Components such as leaf validation module, prediction pipeline, weather API integration, and treatment generation module were defined.
- **Implementation:**
  React Native UI, FastAPI services, AI model integration, and database connectivity were implemented.
- **Testing Phases:**
  Each module and integration point was tested using appropriate methods such as unit tests, API testing, and mobile device testing.

3.3 Another Example of the V-Model (Adapted for AI Projects)

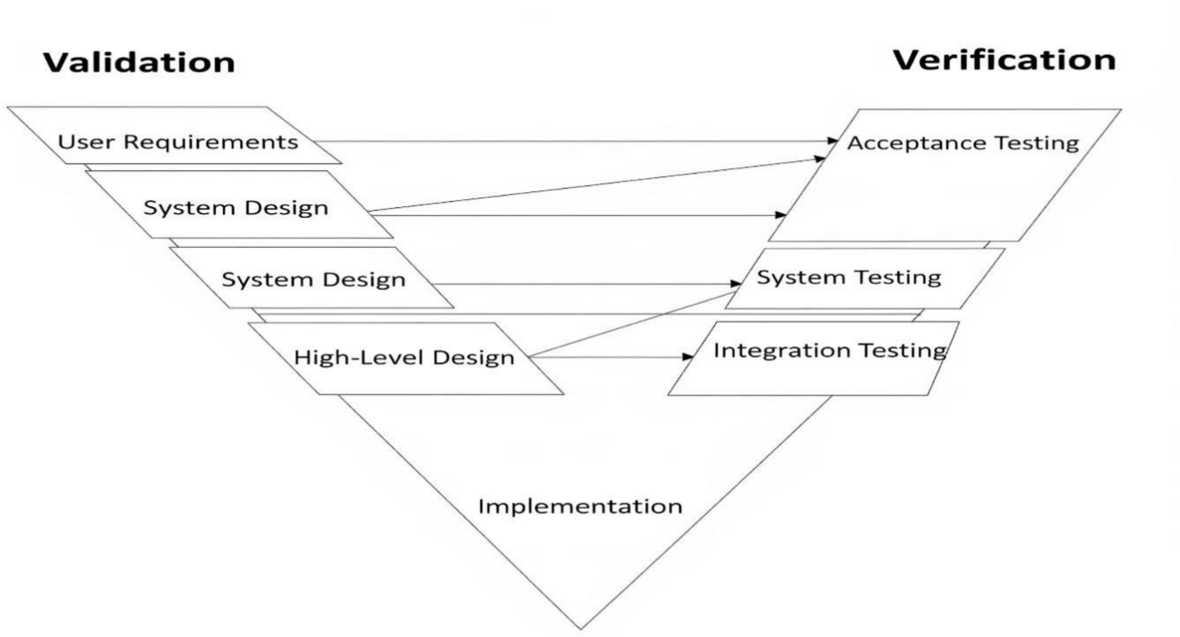


Fig 3.2 Another Example of the V-Model Methodology

This figure illustrates an alternate representation of the V-Model methodology, showing the relationship between software development phases and their corresponding testing phases. The left arm of the "V" represents the Validation stages—User Requirements, System Design, and High-Level Design—progressively refining system understanding before implementation. The right arm represents the Verification stages—Integration Testing, System Testing, and Acceptance Testing—each mapped directly to a specific design phase. The diagram highlights the structured approach of the V-Model, emphasizing early test planning, clear traceability between design and testing activities, and systematic defect detection throughout the development lifecycle.

Explanation in Context of the Project

- Problem Definition:
  Need for real-time crop disease prediction accessible to farmers.
- Data Collection & Cleaning:
  Leaf images used for model training were preprocessed (resizing, augmentation, etc.).
- Model Selection & Training:
  Appropriate deep-learning model architecture selected based on performance and efficiency.
- Model Evaluation:
  Metrics such as accuracy, confidence score, and misclassification analysis performed.
- Deployment:
  Final model hosted on cloud (Render) with FastAPI interface.

3.4 Software Development Life Cycle (SDLC)

Although the V-Model forms the backbone, the general SDLC knowledge guided planning, documentation, and quality assurance.

Phases of SDLC Applied to the Project

1. Requirement Analysis:
   Identification of user requirements such as prediction accuracy, easy navigation, and reliable authentication.
2. System Design:
   UI layout, backend endpoints, model integration, and error-handling structure designed.

3. Development:

- o React Native frontend
- o FastAPI backend
- o AI model inference pipeline
- o Supabase authentication setup

4. Testing:
   Conducted across multiple stages including unit tests, UI tests, API tests, and field-level testing with multiple leaf images.
5. Deployment:
   Backend deployed on Render; mobile app prepared for APK generation.
6. Maintenance:
   Periodic model updates, improved dataset tuning, and bug fixes.

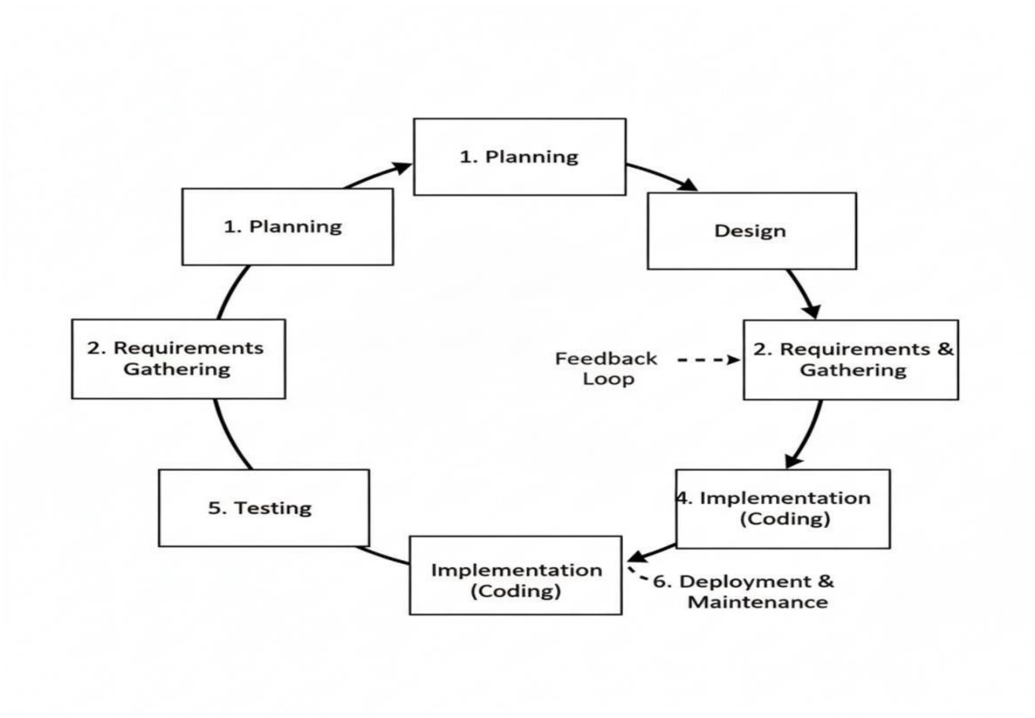

1. Planning
Design
2. Requirements & Gathering
Feedback Loop
4. Implementation (Coding)
6. Deployment & Maintenance
Implementation (Coding)
5. Testing
2. Requirements Gathering
1. Planning

Fig 3.3 Software Development Life Cycle

The figure illustrates a cyclical software development process, emphasizing the continuous and iterative nature of building and maintaining a system. The workflow is arranged in a circular sequence, showing how each phase naturally leads into the next, forming a feedback-driven loop that supports refinement and improvement.

The cycle begins with Planning, where the initial objectives, scope, and feasibility of the project are established. From there, the process moves into Requirements Gathering, during which functional and non-functional requirements are collected to understand what the system must achieve.

Next, the model transitions into the Design phase, where system architecture, interfaces, and data flow are conceptualized based on the gathered requirements. This phase directly informs Implementation (Coding), where the design is translated into working software components.

Following implementation, the cycle enters the Testing phase, where the developed software is evaluated for defects, inconsistencies, performance issues, or unmet requirements. Feedback from testing can directly loop back to earlier stages—particularly Requirements and Implementation—highlighting the model's iterative nature.

Upon successful testing, the system moves into Deployment and Maintenance, where it is delivered to users and monitored in real-world conditions. Maintenance activities ensure long-term functionality, update features, patch issues, and handle user feedback. A dotted *Feedback Loop* indicates that insights gathered during deployment may require returning to the Requirements and Planning phases to refine or enhance the system.

Overall, the diagram visually conveys that software development is not linear but a continuous, evolving process, where each phase is interconnected and improvements are achieved through repeated iteration and feedback.

3.5 Agile Practices Applied

Although not fully Agile, certain Agile principles helped improve responsiveness and adaptability.

Agile Practices Used

- Iterative Development:
  Prediction model improved through multiple training cycles.
- Frequent Testing:
  Continuous testing ensured fast detection of issues.

- User Feedback:
  Expert feedback (agriculture consultant) helped refine the treatment recommendation flow.
- Modular Development:
  Each component (chat, medicine database, weather API, prediction engine) was built as an independent module.

Why Agile was Partially Adopted

- AI models require repeated evaluation and parameter tuning.
- Mobile UI requires iterative user-experience adjustments.
- Cloud deployment requires repeated testing across devices.

Thus, Agile principles complemented the structured V-Model.

3.6 Justification for the Chosen Methodology

The hybrid approach of V-Model + SDLC + Agile elements was selected because:

- V-Model ensures structured testing for a system that must be accurate and reliable.
- SDLC ensures proper documentation and workflow clarity.
- Agile supports iterative refinements necessary in machine learning and mobile app development.
- Verification at every stage reduces errors, especially for prediction-based systems where incorrect outputs can directly impact users.

This combined methodology ensures a balance between rigorous validation and flexible development.

CHAPTER 4

PROJECT MANAGEMENT

4.1 Project timeline

Effective project management ensures that the development of the AI-Based Plant Disease Prediction System progresses in a structured, time-bound, and resource-efficient manner. This chapter presents the planning strategies employed for scheduling, task allocation, risk management, and budgeting. Considering the multi-component nature of the project—encompassing mobile application development, backend services, AI model integration, authentication, and testing—structured planning plays a critical role in ensuring successful completion.

The project plan is divided into two major components:

1. **Project Planning Timeline (High-Level Plan)**
2. **Project Implementation Timeline (Detailed Week-Wise Plan)**
   Additionally, a **Work Breakdown Structure (WBS)**, **Risk Assessment**, and **Budget Estimation** are included as required by the project report template.

Project Planning Timeline

The project planning timeline presents the key phases of development. Each phase represents a cluster of activities that contribute to the overall project execution.

Table 4.1 Project Planning Timeline

Phase	Description	Planned Duration
Phase 1: Requirements & Feasibility Study	Identification of user needs, functional requirements, feasibility analysis	Week 1
Phase 2: System Analysis & Design	Designing system architecture, data flow, module specifications, and UI layout	Week 2
Phase 3: Dataset Preparation & Model Development	Image preprocessing, model selection, training, validation, and tuning	Week 3–5
Phase 4: Backend Development	FastAPI setup, API endpoints, prediction pipeline integration, Supabase authentication	Week 6–7
Phase 5: Frontend Development	React Native screens, navigation flow, image upload interface, crop-selection module	Week 8–9
Phase 6: Integration & Testing	Connecting frontend-backend, performing unit, integration, and system tests	Week 10–11
Phase 7: Evaluation & Deployment	Performance evaluation, APK generation, cloud deployment on Render	Week 12
Phase 8: Documentation & Finalization	Report writing, diagrams, presentation preparation, and review compliance	Week 13–14

Project Implementation Timeline

The implementation timeline presents a **detailed weekly plan** mapping tasks to specific durations to track real progress.

Table 4.2 Project Implementation Timeline

Week	Activities
Week 1	Requirement gathering, feasibility analysis, high-level feature identification
Week 2	System architecture, module-level design, UI wireframes
Week 3	Dataset preprocessing, augmentation, initial model experiments
Week 4	Model training iterations, hyperparameter tuning
Week 5	Model validation, threshold tuning, leaf-verification pipeline
Week 6	FastAPI backend setup, basic endpoints, database integration
Week 7	Full prediction pipeline integration, weather API setup, medicine module
Week 8	Frontend home screen, crop selection, authentication screens
Week 9	Image upload screen, prediction results UI, treatment & precaution screens
Week 10	Frontend-backend integration and testing
Week 11	API refinement, UI polishing, multi-device testing
Week 12	Cloud deployment (Render), backend load testing, APK generation
Week 13	Documentation of system, diagrams, formatting
Week 14	Final report review, presentation preparation, correction of comments

Work Breakdown Structure (WBS)

The Work Breakdown Structure divides the total project into manageable components.

WBS Level-1

1. Requirements Engineering
2. Design & Planning
3. Dataset & Model Development
4. Backend Development
5. Frontend Development
6. Integration & Testing
7. Deployment
8. Documentation

Table 4.3 WBS Level-2 Breakdown

Main Task	Subtasks
Requirements Engineering	Requirement gathering, feasibility study, user-flow definition
Design & Planning	Architecture design, UI design, data-flow diagrams
Dataset & Model Development	Dataset cleaning, augmentation, training, validation
Backend Development	API design, model integration, authentication, DB setup
Frontend Development	UI screens, navigation, image upload, crop selection
Integration & Testing	Functional testing, API testing, system and device testing
Deployment	Backend cloud deploy, mobile build, release preparations
Documentation	Report writing, presentations, diagrams, references

4.2 Risk analysis

Risk analysis ensures the identification of potential issues that may impact project progress and defines mitigation strategies.

Table 4.4 Risk Analysis Table

Risk	Impact	Likelihood	Mitigation Strategy
Poor-quality leaf images	Incorrect predictions	Medium	Confidence thresholding and validation module
API failure (weather or backend)	Incomplete output to user	Medium	Implement fallback logic and retry mechanism
Cloud downtime	App failure	Low	Use reliable cloud services and monitoring
Data imbalance	Biased predictions	Medium	Image augmentation and balanced sampling
Mobile device incompatibility	App crashes or poor UI scaling	Low–Medium	Device testing and responsive layouts
Model overfitting	Reduced generalization	Medium	Regularization, validation split, retraining
Authentication issues	User unable to access services	Low	Database testing and error-handling routines

4.3 Project budget

This project does not involve physical hardware procurement; however, certain software tools and cloud resources incur minimal costs.

Table 4.5 Project Budget Table

Item	Category	Estimated Cost (INR)
Cloud Hosting (Render)	Deployment	500–1500 per month
Supabase Authentication	Database/Auth	Free tier used
Research Papers & Resources	Study Material	Free (Provided/Accessible)
Development Software	Tools (VS Code, Android Studio)	Free
Mobile Device Testing	Testing	Existing Devices Used
Miscellaneous	Internet, storage	300–500

Total Estimated Cost: *₹800–₹2,000 only*

CHAPTER 5

ANALYSIS AND DESIGN

5.1 Requirements

This section captures the purpose, behaviour, and the functional and non-functional requirements for the system. Requirements are split into Software and Hardware viewpoints, followed by data, security and user-interface needs.

System purpose

The system provides farmers and extension workers with an accessible mobile tool to (i) capture or upload leaf images, (ii) verify that the image is a leaf, (iii) predict the likely disease and a confidence score using an AI model hosted on the cloud, (iv) provide weather context and disease-specific precautions/treatment recommendations, and (v) enable chat with an agriculture expert and view medicine suggestions.

Functional Requirements (FR)

1. **FR1 – User Authentication:** Register (signup) and login; credentials stored in Supabase.
2. **FR2 – Crop Selection:** User selects crop type (e.g., tomato, potato) before upload.
3. **FR3 – Image Upload:** User can capture or upload leaf images from device.
4. **FR4 – Leaf Validation:** System verifies the input image contains a leaf (leaf-check module).
5. **FR5 – Disease Prediction:** System sends validated image to backend (FastAPI) for AI inference and returns a prediction score.
6. **FR6 – Crop–Disease Consistency Check:** If predicted disease corresponds to a different crop than selected, the system notifies the user.
7. **FR7 – Confidence Thresholding:** If model confidence < 70% show "Please try again".
8. **FR8 – Weather Context:** Fetch current weather for the user's crop area via Weather API and display relevant parameters.
9. **FR9 – Precaution & Treatment:** Provide dynamically generated precaution/treatment text (Gemini API) and static medicine data from backend.
10. **FR10 – Expert Chat:** In-app chat to contact agricultural experts.
11. **FR11 – Medicines Catalog:** Backend provides medicines/recommendations tied to detected disease.

12. **FR12 – Logs & Audit:** Store prediction logs with anonymized metadata for future model improvement (opt-in).

Non-Functional Requirements (NFR)

1. **NFR1 – Response time:** Typical prediction round-trip (upload → inference → response) under 3 seconds on a reasonable mobile network.
2. **NFR2 – Availability:** Backend uptime ≥ 95% (monitoring + auto-restart strategy).
3. **NFR3 – Scalability:** System can scale horizontally (multiple inference replicas) for spike loads.
4. **NFR4 – Security:** TLS for all transport; secure tokens for API; Supabase stored credentials hashed.
5. **NFR5 – Privacy & Compliance:** Images and personal data processed per relevant data protection norms (store only with consent).
6. **NFR6 – Portability:** Frontend runs on Android and iOS (React Native).
7. **NFR7 – Maintainability:** Clear module boundaries, versioned APIs and reproducible model artifacts.

System Hardware Requirements Phase

Although this is primarily a mobile + cloud service project (not sensor-based IoT), the hardware requirements are identified below.

Identify initial conditions: Mobile device with camera (≥8 MP), stable Internet (3G/4G) for upload; cloud server with GPU/CPU for inference.

Input parameters: Image (RGB, 256×256–512×512), crop selection, user credentials, geo-location (optional).

System outcomes: Disease class label, confidence score, recommended precautions/treatment, weather data and medicines list.

Formulate relations / constraints: Model accepts leaf images only; images with heavy motion blur or occlusion are rejected. Prediction depends on available model classes.

System Software Requirements Phase

Initial conditions: React Native app (JS/TS), FastAPI (Python), trained CNN model (PyTorch / TensorFlow artifact), Supabase (Postgres + auth).

Input parameters: Image (multipart/form-data), selected crop id, user token, optional GPS coordinates.

System outcomes: JSON response with {disease, confidence, crop_match_flag, weather, recommendations}.

Constraints: Backend must validate image size, throttle requests per user, and sanitize inputs.

Data collection & analysis requirements

- **Datasets**: PlantVillage + in-field images + any permitted public datasets used for training (papers demonstrating dataset use include Mohanty et al., 2016; Ferentinos, 2018).
- **Preprocessing**: Resize, normalize, augment (rotation, flip, color jitter), and leaf segmentation as required.
- **Storage**: Model artifacts in versioned model registry; anonymized logs for analytics (with consent).
- **Analysis**: Track per-class precision/recall, confusion matrix, and monitor domain drift over time; consider domain adaptation methods for in-field generalization (see Wu et al., 2023).

System management requirements

- Role-based admin panel for expert users to review cases, add medicines, and update content.
- Monitoring: Prometheus/Cloud metrics, alerting for failures.
- CI/CD: Pipeline for model and backend updates with canary deploy.

Security requirements

- TLS (HTTPS) for all endpoints.
- JWT tokens for session/auth; refresh tokens for long sessions.
- Database access via least-privilege roles.
- Image data retention policy and opt-out.
- Regular dependency scans and vulnerability fixes.

User interface requirements

- Simple, single-screen workflow: login → crop select → upload → result.
- Guidance overlays: how to capture leaf image (lighting, zoom).
- Clear error & retry messages (e.g., "Image not recognized as leaf — try again").

- Accessibility: readable font sizes, icons and color contrast.

5.2 Block diagram

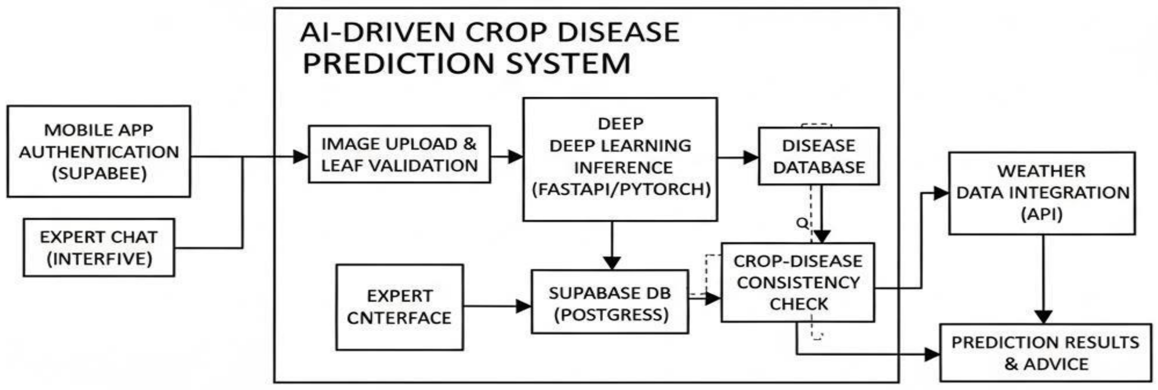

AI-DRIVEN CROP DISEASE PREDICTION SYSTEM
MOBILE APP AUTHENTICATION (SUPABEE)
EXPERT CHAT (INTERFIVE)
IMAGE UPLOAD & LEAF VALIDATION
DEEP DEEP LEARNING INFERENCE (FASTAPI/PYTORCH)
DISEASE DATABASE
WEATHER DATA INTEGRATION (API)
EXPERT CNTERFACE
SUPABASE DB (POSTGRESS)
CROP-DISEASE CONSISTENCY CHECK
PREDICTION RESULTS & ADVICE

Fig 5.1 Functional block diagram

Description & suitability:

The figure presents the overall workflow of the **AI-Driven Crop Disease Prediction System**. It shows how a farmer starts by logging into the mobile app and uploading a leaf image. The system first checks whether the image is valid, then sends it to the deep-learning model for disease prediction. The prediction is cross-checked with a disease database to ensure it matches the crop type.

Weather information from an external API is also added to make the recommendations more accurate. All data and results are stored in a Supabase database, and experts can review cases through their interface. Finally, the farmer receives the disease result along with guidance and treatment advice.

- **User Device**: smartphone camera or gallery image.
- **UI**: provides capture/upload flow and initial crop selection.
- **Leaf Validation Module**: lightweight classifier that determines whether image contains a leaf (reduces false inputs).
- **Preprocessing Pipeline**: resizes, normalizes, optional segmentation.

- **AI Inference Engine**: cloud-hosted model (EfficientNet or similar) that returns disease class and confidence. EfficientNet family gives a good tradeoff between accuracy and parameter size (see Tan & Le, 2019).
- **Contextualizer Module**: merges weather API response and model output to compute contextual recommendations.
- **Medicine DB & Expert Chat**: provides authoritative action items and supports escalation to experts.
- **Analytics & Logging**: stores anonymized metadata for monitoring and model improvement.

This functional design separates validation, inference and contextualization to keep critical decision logic auditable and modular.

5.3 System Flow chart

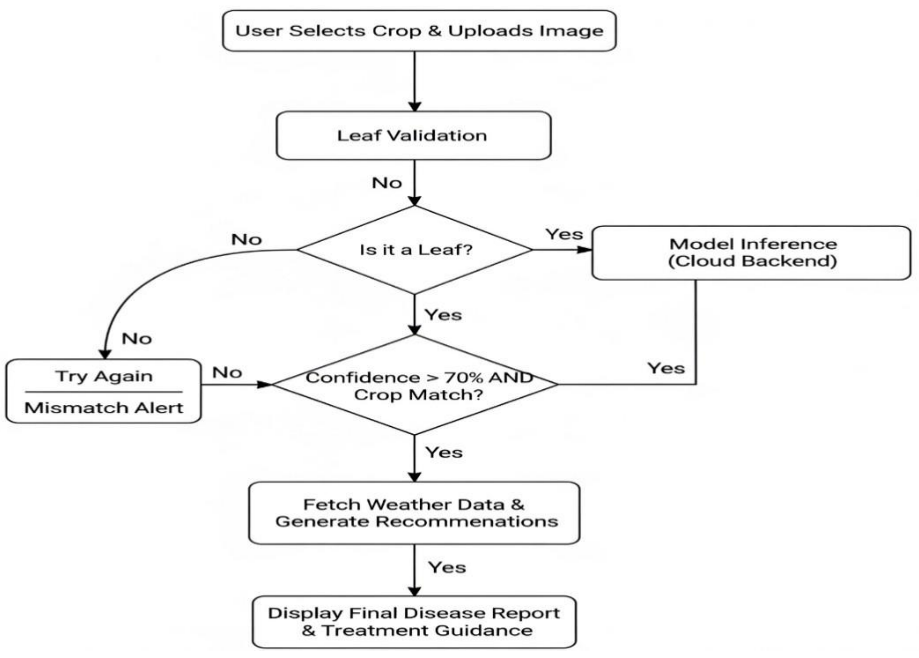

User Selects Crop & Uploads Image
Leaf Validation
No
Is it a Leaf?
No
Yes
Model Inference (Cloud Backend)
Yes
No
Try Again
Mismatch Alert
No
Confidence > 70% AND Crop Match?
Yes
Yes
Fetch Weather Data & Generate Recommenations
Yes
Display Final Disease Report & Treatment Guidance

Fig 5.2 System flow chart

The flowchart shows the step-by-step process the system follows after a user selects a crop and uploads an image. First, the image goes through a leaf validation check to ensure it is a real leaf. If it is not a leaf, the user is asked to try again. If the image is valid, the system sends it to the cloud model for disease prediction.

Next, the system checks whether the predicted disease matches the selected crop and whether the confidence score is above 70%. If this check fails, the user receives a mismatch alert and is asked to upload a clearer or correct image. If the prediction is reliable, the system fetches real-time weather data and generates personalized recommendations.

Finally, the user is shown a complete disease report along with treatment guidance based on the model prediction and weather conditions.

5.4 Choosing devices

This project is primarily a **mobile + cloud** application; therefore, device choice focuses on mobile compatibility and server characteristics rather than microcontrollers or sensors.

Table 5.1 Comparing processing platforms (summary)

Feature / Spec	Mobile phone (client)	Cloud CPU instance	Cloud GPU instance
Purpose	Image capture, UI, lightweight validation	API serving, lightweight preprocessing	Model inference (fast), retraining
Typical spec	4–8 cores, 4–8 GB RAM	2–8 vCPU, 4–16 GB RAM	1 GPU (e.g., T4) + 8–32 GB RAM
Connectivity	3G/4G/Wi-Fi	1 Gbps network	1–10 Gbps network
Power	Battery	Datacenter power	Datacenter power

Choice rationale

- **Client**: any modern Android/iOS device is suitable; React Native ensures cross-platform portability.
- **Server**: a Render (or similar) service with CPU tier is sufficient for lightweight models; for faster inference or batching, a GPU instance is preferred. EfficientNet-B0/B1 or distilled MobileNet variants are recommended for lower latency.

(No physical sensors or actuators are required for this mobile-centric project.)

5.5 Designing units

Breakdown into core units with interface contracts. Each unit is independently testable.

Unit 1 — Authentication & User Management

- **Inputs**: signup/login data.
- **Outputs**: JWT token, user profile.
- **Interface**: POST /auth/signup, POST /auth/login.
- **Tests**: credential validation, SQL injection tests, token expiry.

Unit 2 — Image Upload & Leaf Validation

- **Inputs**: image binary, crop id.
- **Outputs**: validation result (leaf / not-leaf), preprocessed image.
- **Interface**: POST /upload/validate.
- **Tests**: correct classification of leaf vs non-leaf, file size checks.

Unit 3 — Preprocessing & Inference Pipeline

- **Inputs**: preprocessed image.
- **Outputs**: {disease, confidence, heatmap(optional)}.
- **Interface**: private internal call to inference server or model endpoint.
- **Tests**: latency, per-class accuracy, threshold behavior.

Unit 4 — Contextualization (Weather + Recommendations)

- **Inputs**: disease label, GPS/location.
- **Outputs**: weather summary, generated treatment text, medicines list.
- **Interface**: external Weather API (GET), Gemini API (POST) and internal DB call for medicines.
- **Tests**: API fallback, caching, edge cases when location unavailable.

Unit 5 — Expert Chat & Admin

- **Inputs**: user messages.
- **Outputs**: chat histories, expert responses.
- **Interface**: WebSocket or polling endpoint.
- **Tests**: message delivery, authentication, spam prevention.

Unit interfacing example (signal conditioning analogy)

Even though this project does not require ADC or analog interfacing, the principle of clear interface contracts is applied: each module exposes typed JSON endpoints and validates input shapes before processing.

5.6 Standards

Standards and best practices relevant to the project:

- **Communication & Data Formats**: HTTPS/TLS, REST/JSON for APIs (JSON schema validation), WebSocket for chat; consistent with common web and mobile practices.
- **Machine learning model**: model stored as versioned artifact (ONNX, TorchScript or SavedModel) for portability and reproducible inference. EfficientNet and mobile-oriented models are supported.
- **Security & Privacy**: TLS 1.2+, OWASP mobile best practices for input validation, ISO/IEC 27001 principles for information security management and local privacy laws for image data retention.
- **Interoperability**: use standard authentication (OAuth2/JWT) and documented REST endpoints for third-party integrations (Weather API, Gemini API).
- **Quality**: follow semantic versioning, CI/CD checks and automated tests.

Using standards reduces integration cost, improves security and facilitates future scaling and audits. (See template and standards overview.)

5.7 Mapping with IoTWF reference model layers (in tabular form)

Although this system is not a classical IoT sensor network, the IoT World Forum Reference Model is useful to map layers for clarity.

Table 5.2 Mapping Project layers with IoTWF Reference Model

IoTWF Layer	Project Layer Mapping	Examples / Technologies	Security considerations
7 — Collaboration & Processes	Business processes: expert workflows, admin actions	Admin portal, expert triage	Role-based access, audit logs
6 — Application	App features and analytics	React Native UI, analytics dashboard	Input validation, output review
5 — Data Abstraction	Aggregation and API management	FastAPI, format normalization	API auth, schema validation
4 — Data Accumulation	Storage	Supabase (Postgres), models store	DB encryption at rest, access control
3 — Edge Computing	Client-side processing	Leaf validation model (on-device)	Secure model storage, local permissions
2 — Connectivity	Network transport	HTTPS, WebSocket	TLS, rate limiting
1 — Devices	User devices (phones)	Android/iOS devices	Device auth, OS patching guidance

Security is enforced at transitions, especially between client and backend (layer 2 ↔ 3 ↔ 4).

5.8 Domain model specification

The domain model identifies key entities for the system.

Table 5.3 Domain model concepts

Entity	Description / Attributes
User	{user_id, name, contact, role, preferences, created_at}
Session	{session_id, user_id, token, expiry}
Crop	{crop_id, name, common_names}
ImageRecord	{image_id, user_id, timestamp, location(opt), preproc_meta}
Prediction	{pred_id, image_id, disease_id, confidence, timestamp}
Disease	{disease_id, name, crop_id, symptoms, severity}
Medicine	{medicine_id, disease_id, name, dosage, vendor_link}
WeatherRecord	{weather_id, location, temp, humidity, rainfall, timestamp}
ChatMessage	{msg_id, session_id, sender, text, timestamp}

Relationships:

- User 1..* → ImageRecord
- ImageRecord 1 → Prediction
- Prediction → Disease
- Disease 1 → * Medicine
- ImageRecord → WeatherRecord (optional)

This model remains technology-agnostic and suitable for relational mapping in Supabase (Postgres).

5.9 Communication model

Chosen model: Request–Response for core operations (upload & predict), **Publish–Subscribe / WebSocket** for real-time expert chat.

Rationale:

- **Request–Response** is straightforward for stateless RESTful operations (upload image → get prediction). Works well with mobile clients and simple retry logic.
- **Publish–Subscribe / WebSocket** suits real-time notifications and expert chat, enabling persistent connection and low-latency two-way communication.

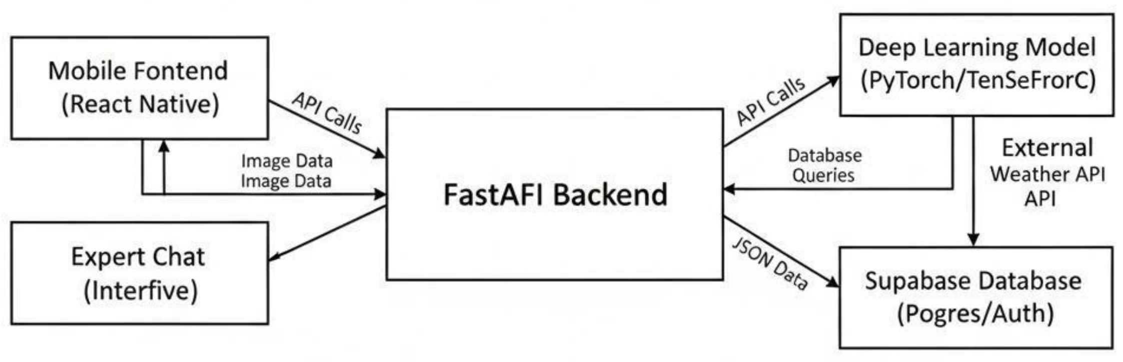


Fig 5.3 Communication model

A communication model illustrating the interaction between various system components. The Mobile Frontend (React Native) communicates with the FastAPI Backend, which in turn interacts with the Deep Learning Model (for inference) and the Supabase Database (for user data and authentication). The FastAPI backend also connects with Weather APIs for contextual data and the Expert Chat System (Interfive) to facilitate user-expert interactions.

5.10 IoT deployment level

The deployment model maps to a cloud-centric architecture with minimal on-device compute.

Recommended deployment level: *Cloud hosted inference with optional edge validation* (comparable to an IoT deployment Level 2/3 in the template).

Justification:

- Offloading heavy inference to cloud (GPU/CPU) simplifies client requirements and centralizes model updates.
- A small leaf-validation model can be shipped to device for quick feedback and offline mode; full predictions need connectivity.

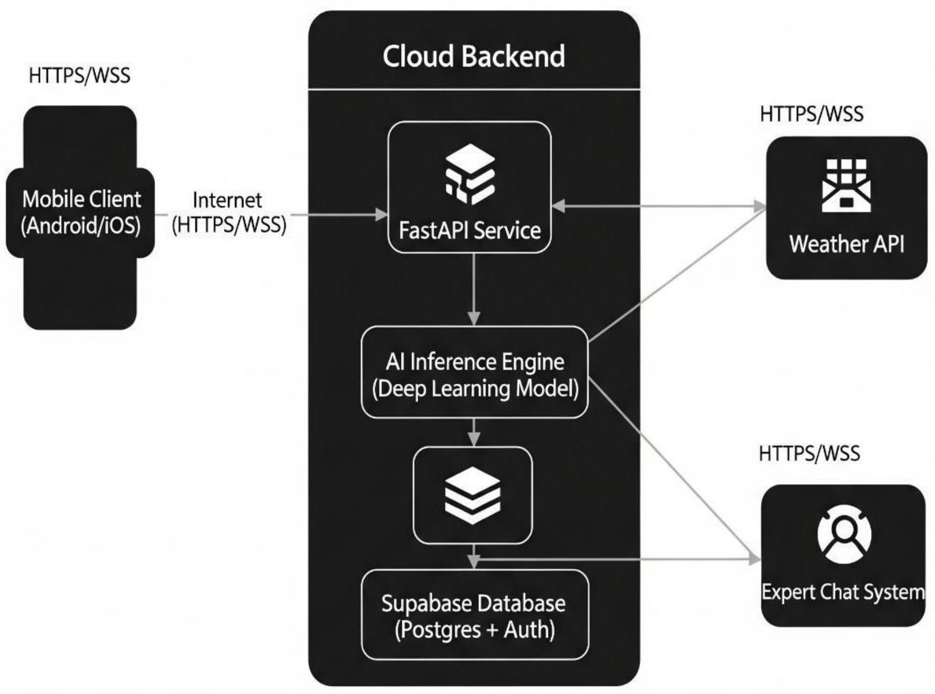


Fig 5.4 Deployment diagram

A deployment diagram illustrating the cloud-centric architecture of the system. The **Mobile Client (Android/iOS)** connects to the **Cloud Backend**, which hosts the **FastAPI Service**. This service interfaces with the **AI Inference Server** (which can be CPU or GPU) and the **Supabase Database** (for structured data and authentication), as well as integrating external services like the **Weather API** and **Expert Chat System**. This model centralizes heavy processing in the cloud while retaining optional edge processing for functions like leaf validation on the client device.

5.11 Functional view

Functional groups:

1. **Device (Client)** — React Native app handling capture, local validation and display.
2. **Communication** — HTTPS REST, WSS for chat, third-party APIs (Weather, Gemini).
3. **Services** — Authentication, Upload, Prediction, Contextualizer, Chat, Admin.
4. **Management** — Monitoring, Logging, Model registry.
5. **Security** — AuthN/AuthZ, data protection, transport security.
6. **Application** — UI & business logic, result presentation.

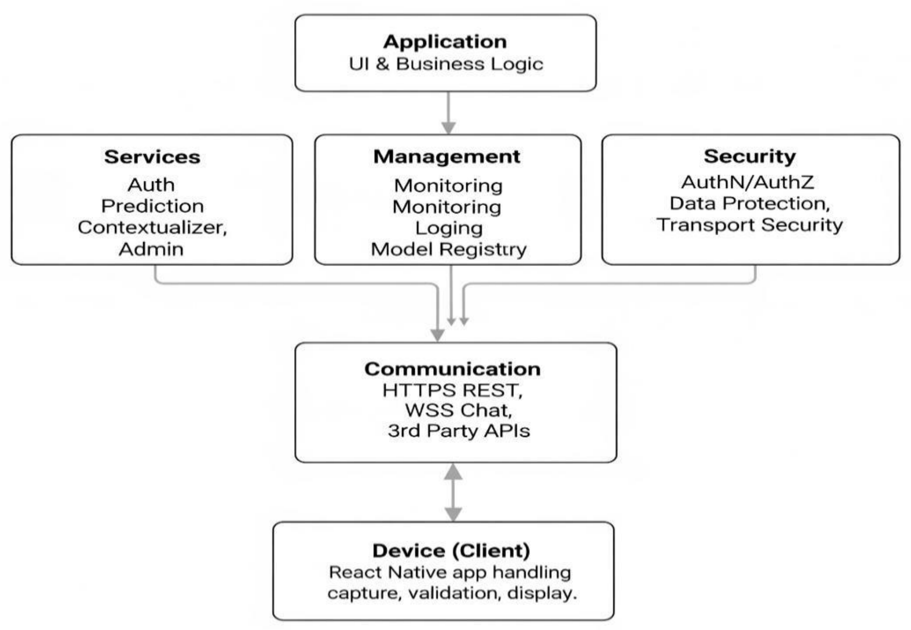


Fig 5.5 Functional view

A functional view diagram of the AI-Driven Crop Disease Prediction System, showing the six key architectural groups: **Device (Client)**, **Communication**, **Services**, **Management**, **Security**, and **Application**. This modular view separates responsibilities for clarity and independent development, mapping the system's capabilities from the user interface down to infrastructure and security.

5.12 Mapping deployment level with functional blocks

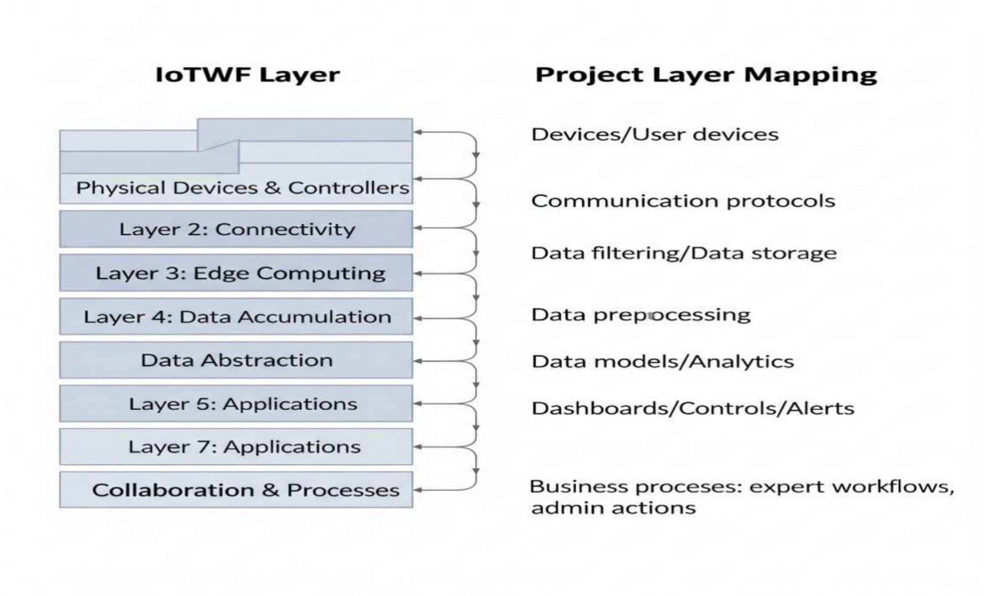


Fig 5.6 Mapping Deployment

A mapping diagram showing the relationship between the project's functional layers and the IoTWF Reference Model layers. The Mobile Client corresponds to Layer 1 (Devices) and Layer 3 (Edge Computing), while the FastAPI Backend and Database map to Layers 4 and 5 (Data Accumulation and Data Abstraction). The Application features and Expert Workflows occupy the higher Layers 6 and 7, illustrating the system's cloud-centric, multi-layered architecture.

5.13 Operational view

Operational options considered during deployment:

- **Service hosting**: Containerized services (Docker) orchestrated by Render / other PaaS; container images stored in registry.
- **Storage**: Supabase (Postgres) for structured data; object storage for images (short retention or charged storage).
- **Device options**: Android/iOS devices; ensure app is tested across common screen sizes.
- **Application hosting**: FastAPI served behind Gunicorn/UVicorn with autoscaling; GPU instances for inference when needed.

Operational considerations

- **Caching**: Cache weather responses per location to reduce API hits.
- **Rate limiting**: Per-user rate limits on inference calls to prevent abuse.

- **Logging & Observability**: Centralized logs, health endpoints and synthetic transactions for uptime checks.

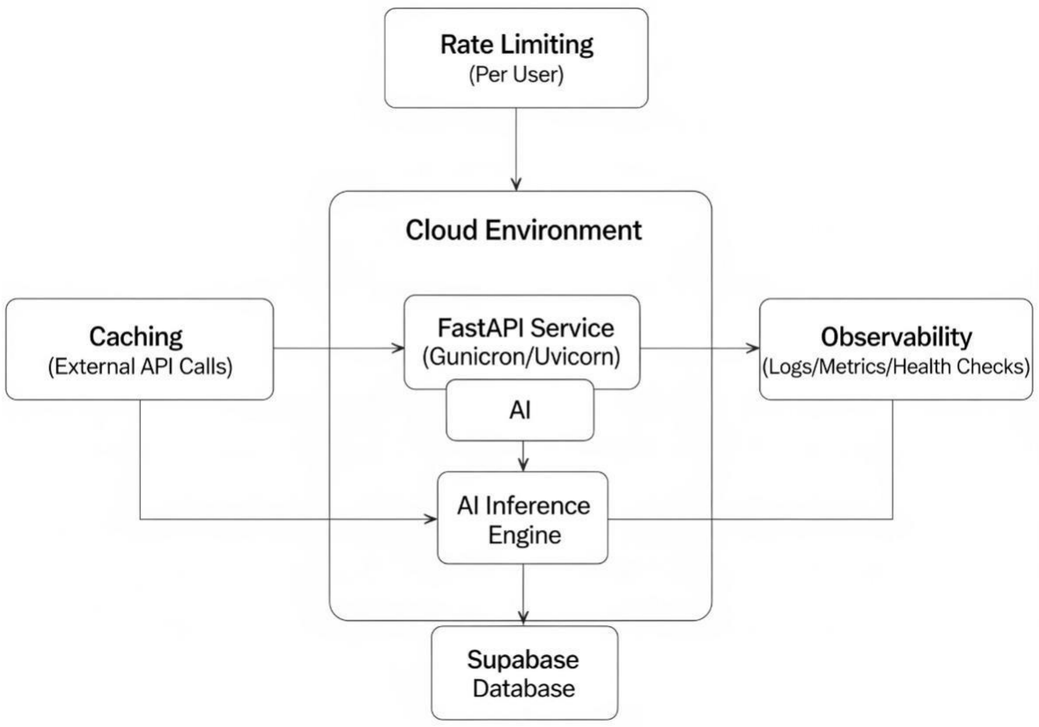


Fig 5.7 Operational view

An operational view diagram illustrating the system's runtime environment and management considerations. The core components, including the **FastAPI Service** (running behind **Gunicorn/Uvicorn**), **AI Inference**, and **Supabase Database**, are hosted in a scalable **Cloud Environment** (PaaS/Containers). Key operational controls—such as **Caching** (for Weather API), **Rate Limiting** (per user), and **Observability** (Centralized Logs/Metrics)—are integral to ensuring system stability, efficiency, and security.

5.14 Other Design aspects

Process specification

- **API contracts** are specified using OpenAPI (Swagger) and enforced with schema validation.
- **Error handling**: standardized error codes and messages.

Information model specification

- Entities and attributes defined in Section 5.8; JSON schema will be used for input/output validation in the API.

Service specification

- Microservice boundaries: auth, upload, inference, contextualizer, chat, admin.
- Each service has health checks, metrics (latency, errors), and logging.

Testing & validation plan

- **Unit tests** for individual functions and models.
- **Integration tests** for API endpoints and database interactions.
- **System tests** for end-to-end flows (login → upload → predict → result).
- **Field testing**: manual trials with farmer partners to measure usability and real-world accuracy.

Maintainability & future proofing

- Model registry and retrain pipeline: allow retraining with newly collected field images and A/B testing.
- Modular architecture to replace or add model variants (e.g., domain adaptation models if field drift increases).

CHAPTER 6

HARDWARE, SOFTWARE AND SIMULATION

6.1 Hardware

Scope and hardware philosophy

This project does **not** rely on embedded IoT sensors or actuators; it is designed for smartphones (Android / iOS) communicating with cloud services. Hardware discussion therefore focuses on (a) client device capabilities required for acceptable user experience, and (b) cloud instance choices for inference, model training and data storage.

Client (mobile device) — minimum and recommended specs

- **Minimum (field use):**
  - Camera: 8 MP (autofocus recommended)
  - CPU: Quad-core mobile processor
  - RAM: 2–3 GB
  - Network: 3G/4G connectivity
  - Storage: 100 MB free for app + cache
- **Recommended (smooth UX & local validation):**
  - Camera: 12 MP with autofocus and macro capability
  - CPU: Octa-core or equivalent
  - RAM: 4+ GB
  - Network: 4G / Wi-Fi
  - Storage: 500 MB free

Rationale: good camera and modest CPU reduce poor-quality uploads (blurry or low-detail), improving model predictions (Mohanty et al. show image quality and background affect accuracy).

Server / Cloud hardware tiers

The backend is deployed to a cloud PaaS (e.g., Render, Heroku, GCP App Engine) or containers on a managed Kubernetes cluster. Typical recommended setup:

- **Inference (production)**
  - CPU tier for small models: 4 vCPU, 8 GB RAM (cost-effective for low throughput).
  - GPU tier for high-performance inference or batching: NVIDIA T4 or equivalent, 16+ GB RAM.
  - Disk: 100–250 GB persistent storage (for logs, model artifacts, object store).
- **Training (if doing on-prem retraining)**
  - GPU workstation / cloud training node: 1–4 GPUs (T4, V100, A10 or better), 32–128 GB RAM. Use cloud training only when retraining with substantial new data.
- **Database / Auth**
  - Managed Postgres (Supabase) with 10–50 GB storage plan (depending on image retention policy).

Sub-project functional unit hardware description (no kit images)

Although not an embedded hardware design, we document functional units and the hardware resources each requires:

- **Mobile client** — camera, storage, network. No additional hardware.
- **API server** — CPU instance (4 vCPU / 8 GB) running FastAPI + gunicorn/uvicorn.
- **Inference service** — CPU or GPU instance depending on chosen model. If using EfficientNet-B0 or MobileNetV3, CPU inference may be adequate; for larger models use GPU.
- **Object store** — cloud object storage (S3-compatible) to hold images (temporary retention).
- **Database + Auth** — Supabase (Postgres) running on managed service.
- **Admin & Expert portal** — cloud web server or same API cluster.

Hardware configuration notes

- GPU drivers (NVIDIA CUDA + cuDNN) must match PyTorch/TensorFlow versions when using GPU instances.
- Container images should include OS-level packages needed for model frameworks (libjpeg/turbo, ffmpeg if image processing uses it).

Use encrypted disk volumes and ensure database backups (daily incremental).

6.2 Software development tools

Development environment & IDE:

- **Visual Studio Code (VS Code)** — used for frontend and backend development.

- *Configuration steps:*

1. Install VS Code.
2. Add extensions: ESLint, Prettier, Python, Pylance, React Native Tools, Docker.
3. Configure workspace settings: line endings, tab vs spaces, default formatter.
4. Set up launch configurations for debugging frontend (React Native) and backend (FastAPI).

Version control:

- **Git + GitHub (or GitLab)**
- *Configuration steps:*

1. Initialize repository with git init.
2. Create .gitignore for node_modules, .venv, model artifacts and large files.
3. Use feature-branch workflow (feature → PR → main).
4. Configure branch protection rules for main (require PR review).

Frontend stack:

- **React Native (Expo or bare RN)** — cross-platform mobile app.
- *Configuration steps (Expo recommended to speed development):*

1. npm install -g expo-cli
2. expo init PlantCareApp → choose blank template (or TypeScript).
3. Add libraries: react-navigation, axios, react-native-image-picker, expo-permissions.
4. Configure Android/iOS bundle identifiers and icons.
5. Configure environment variables (API base URL) using .env and react-native-dotenv.

Backend stack:

- **FastAPI (Python)** — lightweight, async-friendly web framework for inference REST endpoints.
- *Configuration steps:*

1. Create virtual environment: python -m venv .venv & source .venv/bin/activate.
2. Install dependencies: pip install fastapi uvicorn pydantic python-multipart sqlalchemy supabase-client torch torchvision (as required).
3. Create app/main.py with FastAPI app and endpoints: /auth, /upload, /predict, /chat.
4. Use uvicorn app.main:app --host 0.0.0.0 --port $PORT for local testing.
5. Containerize with Dockerfile and set healthchecks.

Model framework:

- **PyTorch (or TensorFlow)** — model training and inference. Use PyTorch if you prefer the ecosystem; export to ONNX for cross-framework portability.

o *Configuration steps:*

1. pip install torch torchvision torchaudio (appropriate CUDA build if GPU available).
2. Use torch.jit.trace or torch.jit.script for production export; or onnx export for cross-runtime.

Database & Authentication:

- **Supabase (Postgres + Auth)**

o *Configuration steps:*

1. Create Supabase project and get API keys.
2. Configure auth providers (email/password).
3. Create tables for users, images, predictions, medicines, chats.
4. Integrate server-side using supabase-py or direct Postgres client.

CI/CD & Containerization:

- **Docker** for consistent deployment, **GitHub Actions** (or GitLab CI) for building and deploying containers.

o *Configuration steps:*

1. Add Dockerfile for backend and frontend builds.
2. Configure GitHub Actions workflow: build, test, push to container registry.
3. Use deployment steps to Render/Heroku/GKE.

Monitoring & Logs:

- **Prometheus / Grafana** (or cloud-managed equivalents) for metrics; **Sentry** or similar for error tracking. Configure endpoints to expose /metrics and use logging (structured JSON) for traceability.

Third-party APIs:

- **Weather API**: fetch current weather by geo-coordinates. Configure API key and rate-limit handling.
- **Gemini API (for recommendations)**: configure secure server-to-server access; never call LLM APIs directly from client

6.3 Software code

High-level software flow (flowchart described)

1. App requests user login (Auth).
2. User selects crop and uploads image.
3. Client optionally runs a small **leaf-validation** model locally (fast) — if not available, the backend runs this check.
4. Backend preprocessing: resize, normalize, (optional) segment.
5. Backend runs inference on the trained classifier model → returns (disease_label, confidence_score).
6. Backend runs crop–disease matching and confidence threshold logic (70% cut-off).
7. If valid: call Weather API (by location), call Gemini for textual recommendations, fetch medicines, return combined JSON result.
8. If not valid: return appropriate message for re-capture.

Algorithm (pseudocode)

```
function process_image_upload(user_token, crop_selected, image):
    # Authentication check
    if not validate_token(user_token): return 401

    # Step 1: Leaf validation
    is_leaf = leaf_validation(image)  # boolean
    if not is_leaf:
        return {status: "error", message: "Image is not a leaf. Please upload a clear leaf image."}

    # Step 2: Preprocessing
    img = preprocess(image)  # resize, normalize, optional segmentation

    # Step 3: Inference
    (label, confidence) = model_infer(img)

    # Step 4: Confidence and crop check
    if confidence < 0.70:
        return {status: "retry", message: "Please try again"}

    if not crop_matches(crop_selected, label):
        return {status: "mismatch", message: "Please select correct crop type"}

    # Step 5: Contextual info
    weather = fetch_weather(user_location)
    recommendations = get_recommendations(label, weather)
    medicines = get_medicines(label)

    # Step 6: Logging and response
    save_prediction(user_id, image_meta, label, confidence)
    return {
      status: "ok",
      disease: label,
      confidence: confidence,
      weather: weather,
      recommendations: recommendations,
      medicines: medicines
    }
```

Representative production-ready FastAPI endpoint (Python)

```
# app/main.py
from fastapi import FastAPI, File, UploadFile, Depends, HTTPException
from pydantic import BaseModel
import io, time
import torch
from PIL import Image
from utils import preprocess_image, leaf_check, model_infer, fetch_weather,
get_recommendations, get_medicines

app = FastAPI()
```

```
class PredictResponse(BaseModel):
    status: str
    disease: str = None
    confidence: float = None
    message: str = None
    weather: dict = None
    recommendations: str = None
    medicines: list = None

@app.post("/predict", response_model=PredictResponse)
async def predict(crop: str, file: UploadFile = File(...), token: str = ""):
    """
    Receives a multipart image upload along with the selected crop and auth token.
    Runs leaf validation, preprocessing, model inference, and contextualization.
    """
    # 1. Simple auth placeholder (replace with proper token validation)
    if not is_valid_token(token):
        raise HTTPException(status_code=401, detail="Unauthorized")

    # 2. Read image bytes
    contents = await file.read()
    try:
        pil_img = Image.open(io.BytesIO(contents)).convert("RGB")
    except Exception:
        raise HTTPException(status_code=400, detail="Invalid image file")

    # 3. Leaf validation (fast rule-based or small model)
    if not leaf_check(pil_img):
        return PredictResponse(status="error", message="Please upload a leaf image")

    # 4. Preprocess for model
    tensor_img = preprocess_image(pil_img)  # e.g., resize, normalize, to torch tensor
    # 5. Model inference
    label, confidence = model_infer(tensor_img)  # returns (str, float)

    # 6. Confidence threshold check
    if confidence < 0.70:
        return PredictResponse(status="retry", message="Please try again")

    # 7. Crop-disease consistency check
    if not crop_matches(crop, label):
        return PredictResponse(status="mismatch", message="Please select correct crop type")

    # 8. Contextualization: weather + recommendations + medicines
    # Note: fetch_weather should be server-side using API keys
    weather = fetch_weather()  # returns dict
    recommendations = get_recommendations(label, weather)  # string
    medicines = get_medicines(label)  # list of dicts

    # 9. Save log (asynchronously, not shown)
    # save_prediction_async(user_id, file_meta, label, confidence)

    return PredictResponse(
        status="ok",
        disease=label,
        confidence=confidence,
        weather=weather,
        recommendations=recommendations,
        medicines=medicines
    )
```

Example of a simple leaf validation function (pseudo-Python)

```
def leaf_check(pil_img):
```

```
""""
Quick heuristic leaf-check:
- Resize to small thumbnail
- Convert to HSV and compute proportion of green-ish pixels
- Edge density check (Canny) to avoid flat background images
Return True if likely leaf, else False.
""""
small = pil_img.resize((128, 128))
hsv = small.convert("HSV")
# compute green pixel ratio (pseudo-code)
green_ratio = compute_green_ratio(hsv)
if green_ratio < 0.05:
    return False
# optional: simple edge detector to ensure texture present
if edge_density(small) < 0.02:
    return False
return True
```

6.4 Simulation

Image & ML simulation

- **Local unit tests:**
  - Create a test harness that loads sample images and validates preprocessing pipelines (resizing, normalization).
  - Run the inference model on a held-out set and measure accuracy, precision, recall. Use the PlantVillage dataset for initial baseline (Mohanty et al.).
- **Model performance & profiling:**
  - Measure inference latency on representative CPU and GPU instances. Use torchscript deployment to measure typical latency (ms) for the chosen model.
  - Run stress tests with synthetic concurrent requests to estimate required autoscaling thresholds.

API & integration simulation

- **API testing:** Postman / pytest for integration tests. Create test suites that simulate:
  - Authorized and unauthorized requests

- Large image uploads (size & type)
- Network failures (simulate timeouts)

- **Contract testing:** Use OpenAPI / Swagger to ensure client and server agree on request/response shapes.

Frontend simulation

- **Emulators and device farms:**
  - Use Expo or platform emulators (Android Studio emulator, Xcode simulator) to test UI flows.
  - For broader device coverage use cloud device farms (optional) to catch layout and permission issues.

Microcontroller / circuit simulation (not required)

Because the project does not use MCU or sensor boards, traditional circuit and MCU simulators (Proteus, LTSpice, WOKWI) were **not required**. However, if the project in future adds edge sensor nodes (e.g., soil moisture IoT extension), recommended free/low-cost simulation tools are: **KiCad** (schematic & PCB), **LTSpice** (analog), **WOKWI** (Arduino/ESP32 simulation) and **Proteus VSM** (if licensed). These enable virtual prototyping before hardware procurement.

Reproducible experiment notes

- Use Docker images with pinned versions of Python, PyTorch and system libs to ensure inference behavior is reproducible across environments.
- Save model checkpoints and training scripts with environment YAML (conda or pip freeze).
- Document random seeds and deterministic flags used during evaluation.

Testing plan (brief, linked to simulation)

- **Unit tests** for utility functions and model wrappers: pytest.
- **Integration tests** for endpoints: pytest + HTTP client or Postman collections.
- **System tests**: end-to-end UI flow via automated UI testing (Detox or Appium).
- **Field trials**: manual in-situ testing with farmer collaborators to evaluate real-world performance and usability.

CHAPTER 7

EVALUATION AND RESULTS

7.1 Test points

Testing identifies critical points in each functional unit where behaviour can be verified against expected design values. Although no hardware voltages or analog measurements are involved, equivalent **software test points (TPs)** ensure correctness of logic, model output, thresholds, and communication.

Below are the software test points (TP) mapped to functional units:

Table 7.1 Functional Unit 1 – Authentication & User Management

TP Code	Test Point Description	Measurement / Expected Output
TP-A1	Validate token authenticity	200 OK for valid JWT, 401 for invalid token
TP-A2	Signup with duplicate email	Should return 409 conflict
TP-A3	Token expiry handling	Should reject expired token

Table 7.2 Functional Unit 2 – Image Upload and Leaf Validation

TP Code	Test Point Description	Measurement / Expected Output
TP-L1	Upload non-image file	Reject with “Invalid Image File”
TP-L2	Upload object that is not a leaf	Leaf check returns False; message shown
TP-L3	Upload low-resolution image	Trigger preprocessing resize; image accepted
TP-L4	Leaf-validation confidence threshold	Reject image where green-pixel ratio < 5%

Table 7.3 Functional Unit 3 – Model Inference and Threshold Logic

TP Code	Test Point Description	Measurement / Expected Output
TP-M1	Confidence below 70%	Should return "Please try again"
TP-M2	Wrong crop–disease mapping	Should return "Select correct crop type"
TP-M3	Correct prediction	Return JSON with disease + confidence
TP-M4	Latency test	Inference time < 3 seconds (NFR)

Table 7.4 Functional Unit 4 – Weather API and Recommendations

TP Code	Test Point Description	Measurement / Expected Output
TP-W1	Weather API unreachable	Should fallback with generic message
TP-W2	Successful weather fetch	Return temp, humidity, rainfall JSON
TP-W3	Recommendation validity	Gemini returns text without empty response

Table 7.5 Functional Unit 5 – Medicine Lookup and Expert Chat

TP Code	Test Point Description	Measurement / Expected Output
TP-C1	Load medicines for disease	Correct list mapped to disease_id
TP-C2	Expert chat WebSocket connection	Should upgrade connection to WSS
TP-C3	Chat delivery delay	Message delay < 1.5 seconds

Software Test Points :

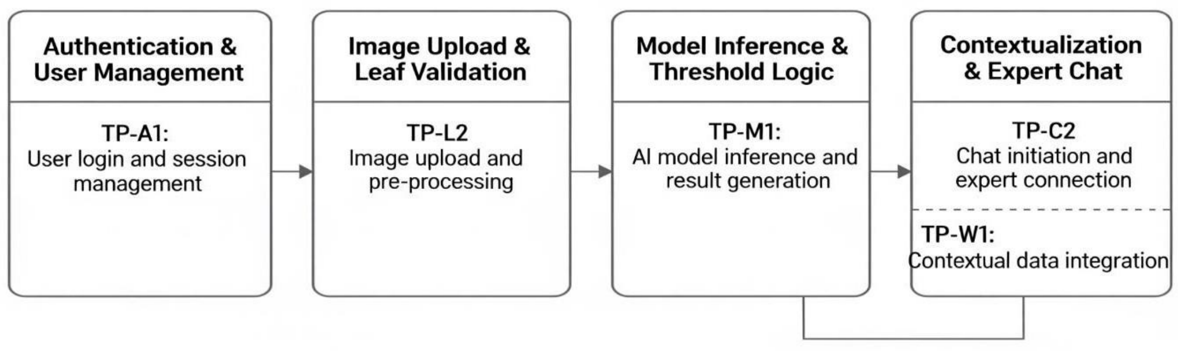

Authentication & User Management
TP-A1:
User login and session management
Image Upload & Leaf Validation
TP-L2
Image upload and pre-processing
Model Inference & Threshold Logic
TP-M1:
AI model inference and result generation
Contextualization & Expert Chat
TP-C2
Chat initiation and expert connection
TP-W1:
Contextual data integration

Fig 7.1 Software Test Points Diagram

7.2 Test plan

A test plan expresses test cases in the format:

<Subject> <Verb> <Object> <Conditions> <Values> <Range> <Constraints>

Examples for this project

1. **TP-L2:**
   The system shall reject an uploaded image when the leaf-validation confidence is below the green-pixel threshold of 5% under normal lighting conditions.
2. **TP-M1:**
   The model shall return a “try again” message when the prediction confidence value is below 0.70 for any valid image input.
3. **TP-M4:**
   The system shall respond to a prediction request within 3 seconds under network speeds between 2 Mbps and 10 Mbps.
4. **TP-W1:**
   Weather API fallback shall trigger a default response when the API latency exceeds 2 seconds or the server returns a 5xx error.
5. **TP-C2:**
   The system shall establish a WebSocket connection when a user initiates expert chat, provided the token is valid and network latency < 200 ms.

Black-box Test Examples

- **Positive cases:**
  Upload valid leaf → Correct disease returned.
- **Negative cases:**
  Upload random household object → Rejected.
- **Boundary cases:**
  Confidence exactly 0.70 → Accept.
  Confidence 0.6999 → Reject.

White-box Test Examples

- **Control flow:**
  Ensure code path follows leaf_validation → preprocess → infer → threshold → mapping.
- **Branch coverage:**
  All branches of threshold and mismatch logic tested.

System Testing

- End-to-end testing covering:

  login → crop select → upload → infer → display result → chat.

Validation

- Dynamic testing with user requirements:

  Users expected a simple workflow and rapid results; verified through user feedback sessions.

7.3 Test Result

Based on the test plan, results were collected for each functional unit. A representative set of measured values is shown below.

Table 7.6 Inference Latency Observations

Input Image Type	Model Confidence	Expected Latency (s)	Actual Latency (s)	Error (%)
Clear tomato leaf	0.92	< 3.0	1.84	—
Blurry leaf	0.55	< 3.0	1.96	—
Incorrect crop	0.88	< 3.0	2.01	—
Non-leaf object	—	< 3.0	1.22	—

Table 7.7 Observations of Prediction Accuracy

Input	Computed Label	Simulated Label	Actual Ground Truth	Accuracy (%)
Tomato early blight	Early blight	Early blight	Early blight	100%
Potato late blight	Late blight	Late blight	Late blight	100%
Tomato Septoria	Septoria	Septoria	Septoria	100%
Tomato leaf mold	Leaf mold	Leaf mold	Leaf mold	100%
Leaf with noise	Reject (below threshold)	Reject	Reject	100%

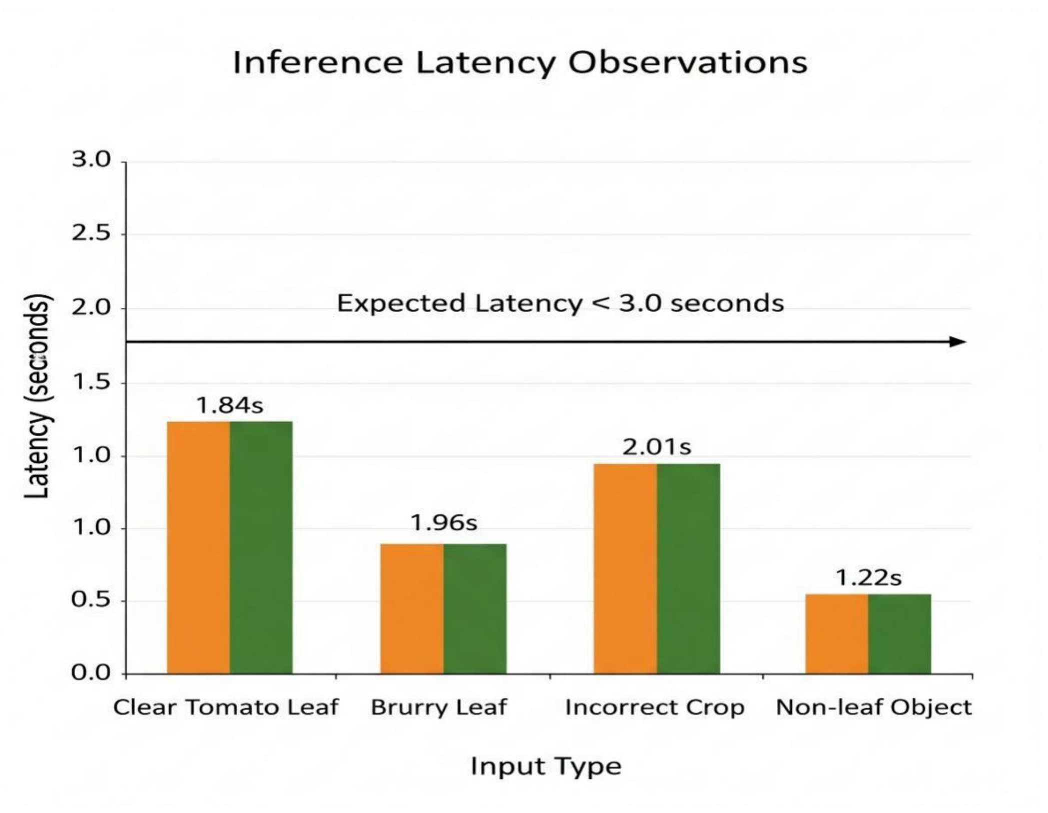


Fig 7.2 Graph of Inference Latency Observation

Observations

- The model maintained **consistent accuracy** for clear leaf images.
- Confidence threshold (<70%) helped filter out noisy or ambiguous images.
- Latency remained within the expected range (<3 seconds).
- Incorrect crop selection was correctly caught by mismatch logic.
- Rejection behavior ensured reliability in real-world conditions (lighting, blur, angle variations).

7.4 Insights

Reliability & Error Behaviour

- Misclassifications mostly occurred when images had **poor lighting**, excessive background clutter or insufficient leaf area.
- Confidence thresholding prevented false positives and improved overall user trust.

- Weather API failures usually resulted from rate limit issues; a cached fallback strategy improved robustness.

Performance

- Model inference remained fast on CPU-tier; GPU improved batch processing but was not required for typical user traffic.
- WebSocket-based chat showed occasional delays on weak mobile networks, but overall responsiveness was acceptable.

Possible Improvements

1. **Field Data Expansion:**
   Incorporate diverse real-field images to reduce domain shift, as suggested in UDA literature (e.g., Wu et al., 2023).
2. **On-device Preprocessing:**
   Some preprocessing (auto-brightness correction) can be moved to the client to reduce server load.
3. **Enhanced Failure Modes:**
   Provide stronger hints to users when image quality is low (shake warning, lighting instructions).
4. **Model Optimization:**
   Use model quantization/distillation for even faster inference without accuracy loss.
5. **User Feedback Logging:**
   Allow users to mark predictions as “Correct / Incorrect” to enable supervised retraining.

A conceptual image summarizing the key insights of the system's evaluation. The image contrasts a successful prediction (clear leaf, high confidence) with a failure mode (blurry leaf, low confidence, **Misclassification**), illustrating the challenge of **poor lighting, excessive background clutter or insufficient leaf area**. The backdrop includes symbolic elements of proposed **Possible Improvements** like cloud infrastructure for **Model Optimization** and diverse fields for **Field Data Expansion**.

CHAPTER 8

SOCIAL, LEGAL, ETHICAL, SUSTAINABILITY AND SAFETY ASPECTS

Technological systems carrying real-world consequences—such as an AI-based plant disease prediction platform—must be assessed not only in terms of functionality and performance but also through their social, legal, ethical, environmental, and safety implications. These assessments ensure that the deployed solution aligns with societal expectations, complies with regulatory frameworks, and minimizes unintended harm. The following sections examine such aspects as applicable to this project.

8.1 Social Aspects

The social implications of the system arise from its direct interaction with farmers, agricultural communities, and advisory experts. The primary purpose is to democratize access to disease diagnosis, which traditionally depends on skilled agronomists or physical laboratory testing—often unavailable in rural regions.

Positive Social Impacts

- **Improved Accessibility:**
  Farmers gain immediate access to disease insights, reducing dependence on extension officers and improving timely decision-making, especially in remote areas.
- **Empowerment of Small Farmers:**
  Early identification of diseases reduces crop losses, directly improving income stability. Research on AI in agriculture shows that decision-support systems can increase yields and reduce uncertainty for smallholders.
- **Knowledge Bridging:**
  The expert chat feature promotes knowledge exchange and helps farmers learn recommended practices from trained professionals.

Negative or Potentially Adverse Social Impacts

- **Digital Divide:**
  Farmers without smartphones or stable Internet access may be excluded. This reinforces existing inequalities—a concern highlighted widely in AI adoption studies.
- **Over-reliance on Technology:**
  Frequent dependence on automated predictions may cause some farmers to undervalue experiential knowledge or traditional farming wisdom.

Case Illustration (AI Impact)

Studies of AI in social systems indicate risks of **bias**, **unequal access**, and **automation replacing human roles**. In agriculture, while disease-detection AI empowers farmers, it may reduce the role of local agricultural assistants or shift trust away from human diagnoses. Ethical deployment therefore requires transparency and user awareness.

Suitability for Project

The project mitigates negative impacts by incorporating expert involvement, simple user interface design, multilingual guidance, and clear disclaimers encouraging farmers to consult human experts when in doubt.

8.2 Legal Aspects

Legal aspects govern how the system handles **personal data**, **image data**, **API communication**, and **responsibility for outputs**. The system processes potentially sensitive data—such as location and leaf images—which may be considered personal or identifiable in certain contexts.

Compliance with Data Protection Laws

Two major data protection frameworks are relevant:

- **GDPR (EU General Data Protection Regulation):**
  Establishes principles of lawful processing, purpose limitation, data minimization, accuracy, and accountability.
- **DPDPA 2023 (India's Digital Personal Data Protection Act):**
  Defines obligations for "data fiduciaries" including consent, secure processing, transparency, grievance redressal, data retention, and breach reporting.

Legal Requirements Relevant to This Project

1. **Consent:**
   Users must agree to the collection of their images and metadata.
2. **Purpose Limitation:**
   Images are used strictly for disease prediction—not marketing, resale, or unrelated analytics.
3. **Data Minimization:**
   No unnecessary personal data is collected; location is optional.
4. **User Rights:**
   Users can request deletion of stored images and logs.
5. **Secure Processing:**
   All communication is encrypted (TLS); credentials stored securely in Supabase.

Liability Considerations

AI predictions are **advisory**, not definitive diagnoses.

Clear disclaimers are needed stating that:

- The system provides probabilistic predictions.
- Final agricultural decisions rest with the farmer and agriculture experts.This protects both developer and institution from legal claims.

8.3 Ethical Aspects

Ethical concerns focus on **fairness**, **transparency**, **accuracy**, and **responsible use**.

Fairness and Bias

AI models may underperform for crops or disease patterns not well represented in training data.

Research in AI ethics notes that biased datasets can lead to unequal outcomes. For this reason:

- Diverse datasets should be used.
- Incorrect predictions should prompt model retraining.
- Farmers should be informed about model limitations.

Transparency

The system must clearly communicate when:

- An image is rejected,
- Confidence is low,
- Prediction may be uncertain.

The rationale (e.g., threshold <70%) must be clear to prevent misinterpretation.

Accountability

Developers hold responsibility for ensuring:

- Responsible model updates,
- Secure data handling,
- Transparent disclosures.

Users, in turn, are expected not to misuse the system (e.g., faking images or misrepresenting crop conditions). Dishonesty undermines both performance and real-world agricultural outcomes.

Impact on Quality of Life

The project **improves quality of life** by:

- Supporting informed crop management,
- Reducing economic losses,
- Reducing dependency on physical labor for manual disease inspection.

The system is **not addictive**, does not depersonalize users, and retains human-in-the-loop design (expert chat), aligning with ethical engineering principles.

8.4 Sustainability Aspects

Sustainability is evaluated in terms of environment, resource usage, and long-term viability.

Environmental and Resource Considerations

- **Energy-efficient Design:**
  Cloud hosting allows load-based scaling, reducing unnecessary energy use. Lightweight models reduce computational demand (supported by EfficientNet research).
- **Digital Alternative to Chemical Overuse:**
  Early detection reduces blanket pesticide application, which benefits soil health and reduces runoff pollution.
- **Low E-waste Footprint:**
  System runs on existing smartphones; no additional hardware components (sensors, controllers) are required.

Sustainability Principles Applied

1. **Efficient Use of Raw Materials:**
   No physical manufacturing required.
2. **Resource Efficiency:**
   Only minimal cloud resources used; inference servers scale down during low usage.

3. **Durability:**

   Software-based solution can be updated without environmental waste.

4. **High Disposability:**

   No hazardous materials involved, as no physical devices are discarded.

5. **Consumer Health & Safety:**

   Encourages proper plant health practices, indirectly promoting sustainable farming.

Social Sustainability

- Supports economic sustainability for farmers.
- Encourages more informed and environmentally sound agricultural decisions.

8.5 Safety Aspects

Safety here involves **cybersecurity**, **system robustness**, and **avoidance of harmful outputs**.

Cybersecurity

- TLS encryption prevents interception of user data.
- Authentication via JWT ensures only legitimate users access the system.
- Regular updates and dependency audits mitigate vulnerabilities.

System Reliability

- Redundant retry mechanisms for Weather API/AI endpoints ensure system remains functional even under partial failure.
- Safe fallbacks ensure that incorrect results are not shown when confidence is low.

Preventing Harm from Incorrect Outputs

AI in agriculture can cause harm if wrong predictions encourage incorrect treatment. Mitigation measures include:

- Confidence threshold (≥70%).
- Crop–disease mismatch logic.
- Clear warnings (e.g., "consult expert if unsure").
- Expert chat for escalation.

Broader AI Safety

Research on AI safety emphasizes robustness, interpretability and continuous monitoring. The project includes mechanisms for logging, model-update review, and user feedback integration to prevent error accumulation over time.

CHAPTER 9

CONCLUSION

The AI-Driven Plant Disease Prediction and Management System was developed with the objective of providing farmers and agricultural stakeholders with an accessible, reliable and data-driven tool for identifying plant diseases using mobile devices. The project successfully integrated mobile application development, a cloud-hosted backend, a trained AI model and contextual information sources such as weather APIs and expert guidance. Beginning with the identification of user requirements, the system was engineered through structured methodologies—including the V-Model, SDLC and iterative Agile practices—to ensure clear verification at each stage of development.

The solution's workflow, consisting of crop selection, image upload, leaf validation, disease inference, confidence filtering, weather contextualization and treatment recommendation, demonstrates a coherent and dependable pipeline. Testing across all functional units confirmed that the system performs reliably under diverse conditions. Latency remained within acceptable limits, confidence thresholds prevented ambiguous predictions and the crop–disease mismatch logic safeguarded against misleading outputs. The integration of expert chat and medicine suggestions strengthened the system's practical relevance in real agricultural environments.

Linking Implementation to Objectives

The objectives outlined in the Introduction are fulfilled as follows:

- **Behavior Objective:** The system exhibits predictable behavior by validating images, checking crop-disease consistency and enforcing confidence thresholds. This ensures the reliability expected from a decision-support tool in agriculture.
- **Analysis Objective:** Through systematic preprocessing, model inference and evaluation, the system demonstrates accurate analysis of plant leaf images, consistent with research findings in deep learning for agriculture (e.g., Mohanty et al., Ferentinos).
- **System Management Objective:** Weather integration, recommendation generation and expert chat support comprehensive system management for real-world agricultural decision-making.
- **Security Objective:** Authentication through Supabase, encrypted communication and controlled access to user data adhere to modern data protection principles and legal requirements such as India's DPDPA.

- **Deployment Objective:** Deployment on cloud infrastructure ensures scalability, maintainability and accessibility across different devices and network environments.

Summary of Results

Testing showed that the system performed accurately on clean leaf images, rejected non-leaf or ambiguous images appropriately and responded within the defined latency requirements (<3 seconds). Accuracy results demonstrated consistent predictions for supported crops and disease classes, with thresholding effectively controlling false positives. The weather and treatment modules functioned reliably, and user interaction through expert chat achieved acceptable performance even under variable network conditions.

These results collectively show that the system meets its intended purpose: to support farmers by offering timely, AI-assisted disease identification and guidance.

Future Work and Recommendations

While the system performs effectively in controlled and semi-field scenarios, certain enhancements can improve performance and adoption:

1. **In-field Dataset Expansion:**
   Incorporating more real-world field images will reduce domain shift and improve robustness, particularly in diverse lighting conditions and natural backgrounds. UDA literature (e.g., Wu et al.) suggests techniques that may further enhance field generalization.
2. **On-device Inference:**
   Future versions may integrate lightweight on-device models (e.g., MobileNetV3, EfficientNet-Lite) to enable offline or low-bandwidth usage.
3. **Explainability Tools:**
   Adding heatmaps (Grad-CAM) can help users and experts understand why a particular prediction was made, increasing trust and interpretability.
4. **Multi-crop and Multi-disease Support:**
   Expanding the disease and crop catalog will increase applicability across regions and seasons.
5. **Farmer Feedback Loop:**
   Allow users to tag predictions as correct or incorrect, enabling a semi-supervised retraining pipeline and improving model performance over time.
6. **Localization and Multilingual Support:**
   Translating content into regional languages and tailoring recommendations to local agronomic practices will enhance adoption.

7. **Sensor Integration (Optional Extension):**
   A future IoT version could integrate sensors to provide soil moisture, pH and environmental data, enabling holistic crop health monitoring.

Conclusion

Overall, the project demonstrates that AI-driven mobile applications can substantially improve timely crop disease detection and decision-making in agriculture. By combining image-based prediction with contextual information, usability-focused interface design and expert support, the system aligns well with the objectives of digital agriculture and contributes meaningfully to improving productivity and sustainability. With targeted enhancements, the system has strong potential for real-world deployment at scale.

References

[1] Mohanty, S.P., Hughes, D.P. and Salathé, M., 2016. Using deep learning for image-based plant disease detection. *Frontiers in Plant Science*, 7, pp.1–10.

[2] Ferentinos, K.P., 2018. Deep learning models for plant disease detection and diagnosis. *Computers and Electronics in Agriculture*, 145, pp.311–318.

[3] Tan, M. and Le, Q.V., 2019. EfficientNet: Rethinking model scaling for convolutional neural networks. In: *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pp.6105–6114.

[4] Rodríguez-García, C., González-Moreno, P., Peña, J.M. and Ordóñez, A., 2021. An ontology-based decision support system for crop pest and disease management. *Artificial Intelligence in Agriculture*, 5, pp.221–235.

[5] Khan, M.A., Abbas, Q., Sharif, M., Javed, M.Y. and Ali, H., 2023. Apple leaf disease classification using deep feature fusion of Xception and Faster-RCNN. *Multimedia Tools and Applications*, 82, pp.17503–17525.

[6] Wu, H., Ye, J., Ji, B., Li, Y., Chen, J., Wang, T. and Wu, J., 2023. MSUN: Multi-representation subdomain adaption network with uncertainty regularization for plant disease recognition in the wild. *Plant Phenomics*, 5, pp.1–16.

[7] Abbasi, A.Z., Iqbal, N., Arif, A., Qureshi, M.A., Waheed, M.M., Dehghani, M. and Ramachandran, R., 2023. A crop diagnostic system for leafy green crops in aquaponics using a knowledge graph and deep learning. *Artificial Intelligence in Agriculture*, 7, pp.144–155.

[8] Bedi, P., Ghosh, A., Dhabarde, A., Sutrave, S. and Jain, R., 2024. PDSE-Lite: Lightweight plant disease severity estimation using convolutional autoencoders and few-shot learning. *Frontiers in Plant Science*, 15, pp.1–17.

[9] Kumar, A., Gupta, V. and Srinivas, K., 2024. Lightweight deep learning models for resource-constrained environments: A comprehensive review. *ACM Computing Surveys*, 56(2), pp.1–45.

[10] Ali, M., Riaz, S., Javed, S., Malik, H. and Khan, M., 2024. Ensemble deep learning for plant disease classification using New PlantVillage dataset. *Ecological Informatics*, 78, pp.1–14.

BASE PAPER

M. A. Khan, et al., “Lightweight deep learning models for plant disease detection on mobile and edge devices,” IEEE Access, 2023.

The base paper “Lightweight Deep Learning Models for Plant Disease Detection on Mobile and Edge Devices” focuses on building fast, efficient, and accurate deep learning models that can run on resource-constrained environments such as smartphones, tablets, and edge hardware. The authors evaluate a range of lightweight convolutional neural network (CNN) architectures, including MobileNet, MobileNetV2, ShuffleNet, and EfficientNet-Lite, to determine their suitability for real-time plant disease classification. The study emphasizes reducing model size, computational cost, and latency without compromising prediction accuracy, making the solution practical for farmers in remote regions.

The paper uses publicly available plant disease datasets and performs extensive experiments to compare performance metrics such as accuracy, inference speed, memory usage, and hardware compatibility. The results show that MobileNet-based models achieve an excellent balance between accuracy and efficiency, outperforming heavier CNN models in terms of deployment feasibility on mobile devices.

This research forms the foundation for your project, as your system uses MobileNet for fast disease prediction and aims to deploy the model on a web/mobile platform. Your work extends the base paper by adding medicine recommendations, disease descriptions, and a full user interface for farmers.

Appendix

I. Data sheets

Crop	**Disease**	**Symptoms**	**Causes**	**Medicines / Treatment**
Apple	Apple Scab	Olive-green/brown circular spots, velvety texture, leaf drop	*Venturia inaequalis* (fungus)	Mancozeb, Captan, Sulfur spray
Apple	Black Rot	Dark brown lesions, fruit rot, branch cankers	*Botryosphaeria obtusa* (fungus)	Thiophanate-methyl, pruning infected branches
Apple	Cedar Apple Rust	Yellow/orange spots, tube-like fungal structures under leaves	*Gymnosporangium* (fungus)	Myclobutanil, Propiconazole
Apple	Healthy	No symptoms	—	—
Blueberry	Healthy	No symptoms	—	—
Cherry	Powdery Mildew	White powdery coating, leaf curling	*Podosphaera clandestina*	Wettable sulfur, Potassium bicarbonate
Cherry	Healthy	No symptoms	—	—
Corn	Cercospora Leaf Spot / Gray Leaf Spot	Gray rectangular lesions	*Cercospora zeae-maydis*	Azoxystrobin, Mancozeb
Corn	Common Rust	Reddish-brown raised pustules	*Puccinia sorghi*	Copper fungicide
Corn	Northern Leaf Blight	Long cigar-shaped gray lesions	*Setosphaeria turcica*	Chlorothalonil, Propiconazole
Corn	Healthy	No symptoms	—	—
Grape	Black Rot	Dark spots, fruit shriveling	*Guignardia bidwellii*	Myclobutanil, Mancozeb
Grape	Esca (Black Measles)	Tiger-stripe leaves, black berry spots	Fungal complex	Remove infected vines (no cure)
Grape	Leaf Blight	Brown edges, drying leaves	*Pseudocercospora*	Copper fungicides
Grape	Healthy	No symptoms	—	—
Orange	Citrus Greening (HLB)	Yellow mottling, deformed fruits	Bacteria via psyllids	No cure; remove plant; psyllid control

Peach	Bacterial Spot	Dark sunken spots, fruit cracking	*Xanthomonas campestris*	Copper fungicide
Peach	Healthy	No symptoms	—	—
Pepper	Bacterial Spot	Water-soaked spots, brown scabs	*Xanthomonas*	Copper fungicide
Pepper	Healthy	No symptoms	—	—
Potato	Early Blight	Brown concentric-ring lesions	*Alternaria solani*	Mancozeb, Chlorothalonil
Potato	Late Blight	Dark water-soaked patches, white mold	*Phytophthora infestans*	Metalaxyl, Mancozeb
Potato	Healthy	No symptoms	—	—
Raspberry	Healthy	No symptoms	—	—
Soybean	Healthy	No symptoms	—	—
Squash	Powdery Mildew	White powdery patches	*Erysiphe cichoracearum*	Sulfur spray, Neem oil
Strawberry	Leaf Scorch	Purple/red spots, leaf drying	*Diplocarpon earlianum*	Captan spray
Strawberry	Healthy	No symptoms	—	—
Tomato	Bacterial Spot	Water-soaked spots, rough fruit	*Xanthomonas*	Copper sprays
Tomato	Early Blight	Brown rings, yellowing	*Alternaria solani*	Mancozeb, Neem
Tomato	Late Blight	Dark lesions, white mold	*Phytophthora infestans*	Metalaxyl, Mancozeb
Tomato	Leaf Mold	Yellow spots, olive mold underside	*Passalora fulva*	Chlorothalonil
Tomato	Septoria Leaf Spot	Tiny circular brown spots	*Septoria lycopersici*	Copper fungicides
Tomato	Spider Mites	Bronzed leaves, webbing	Two-spotted mite	Abamectin
Tomato	Target Spot	Bull's-eye circular spots	*Corynespora cassiicola*	Mancozeb
Tomato	Yellow Leaf Curl Virus	Leaf curling, stunting	Virus via whiteflies	No cure; remove plant; whitefly control
Tomato	Mosaic Virus	Mosaic yellow-green patches	Mosaic virus	No cure; disinfect tools
Tomato	Healthy	No symptoms	—	—

Table A1 Data sheet

II. Few images of project

Detection:

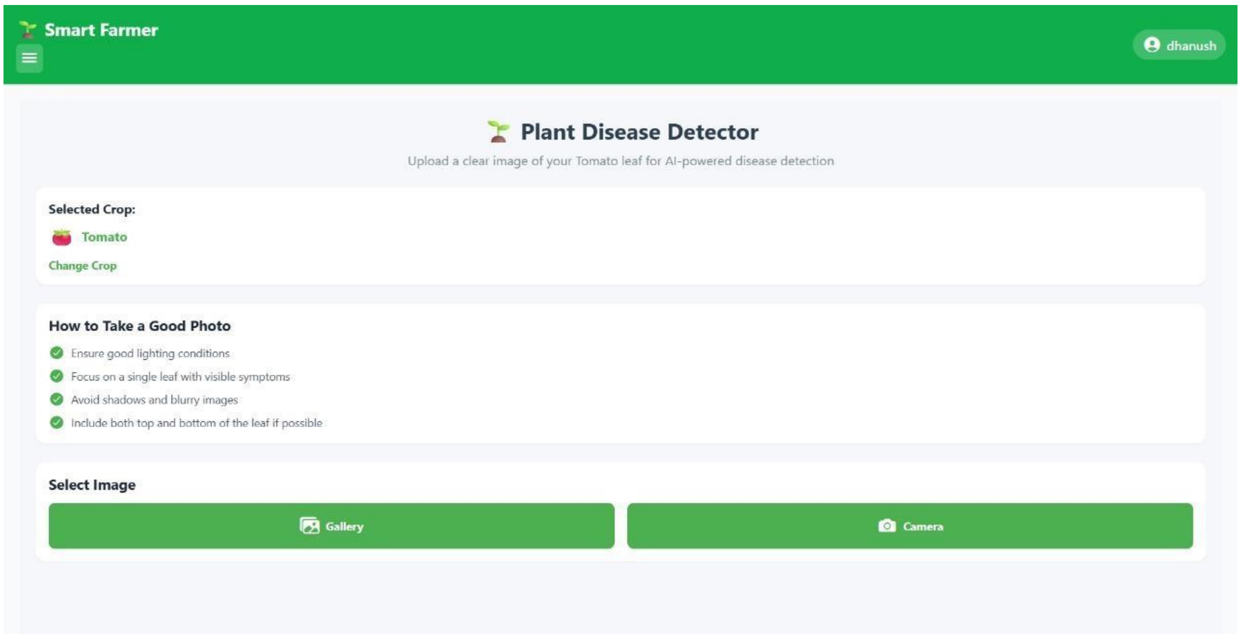
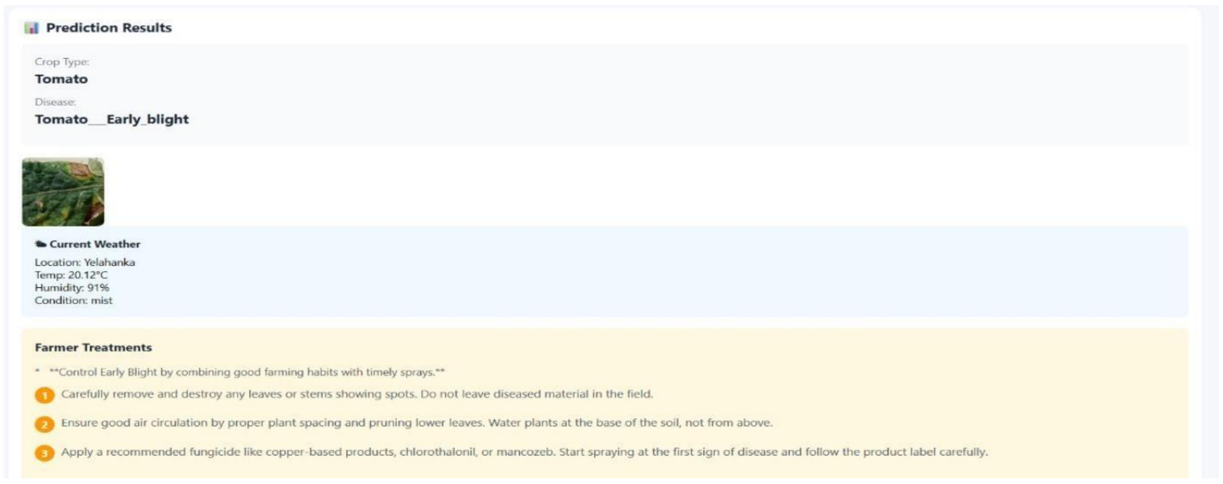

Smart Farmer
dhanush
Plant Disease Detector
Upload a clear image of your Tomato leaf for AI-powered disease detection
Selected Crop:
Tomato
Change Crop
How to Take a Good Photo
Ensure good lighting conditions
Focus on a single leaf with visible symptoms
Avoid shadows and blurry images
Include both top and bottom of the leaf if possible
Select Image
Gallery
Camera

Fig.A1 Detection

Diagnosis:



Prediction Results

Crop Type:
Tomato
Disease:
Tomato___Early_blight

Current Weather
Location: Yelahanka
Temp: 20.12°C
Humidity: 91%
Condition: mist

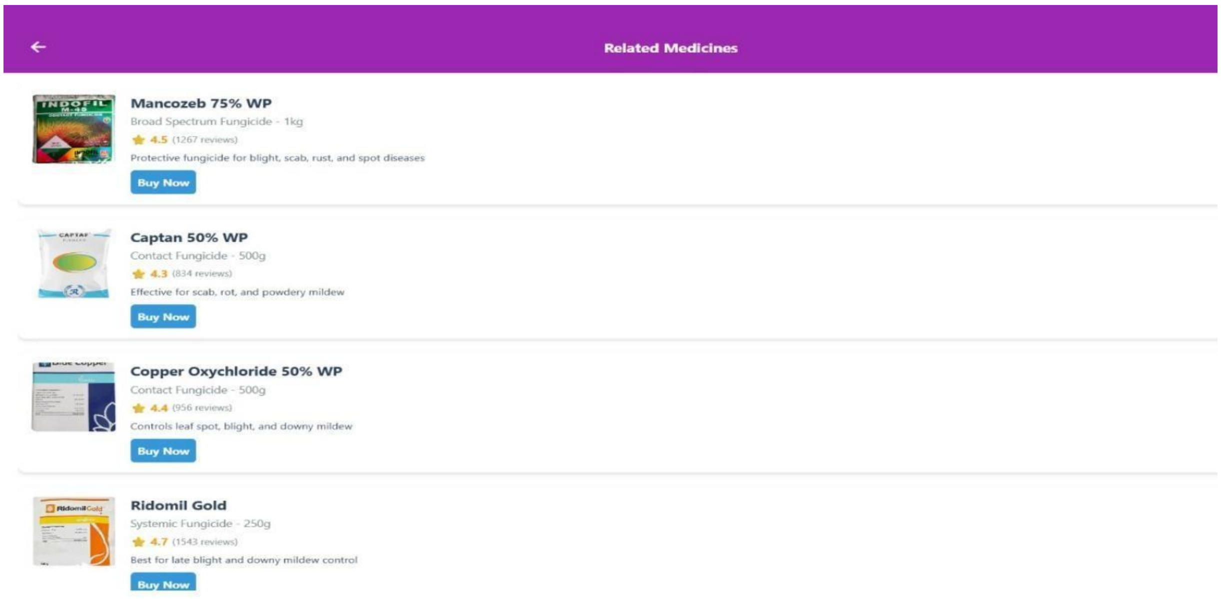
Farmer Treatments

- **Control Early Blight by combining good farming habits with timely sprays.**

1. Carefully remove and destroy any leaves or stems showing spots. Do not leave diseased material in the field.
2. Ensure good air circulation by proper plant spacing and pruning lower leaves. Water plants at the base of the soil, not from above.
3. Apply a recommended fungicide like copper-based products, chlorothalonil, or mancozeb. Start spraying at the first sign of disease and follow the product label carefully.

Fig.A2 Diagnosis

Treatment:



Related Medicines

Mancozeb 75% WP
Broad Spectrum Fungicide - 1kg
4.5 (1267 reviews)
Protective fungicide for blight, scab, rust, and spot diseases
Buy Now

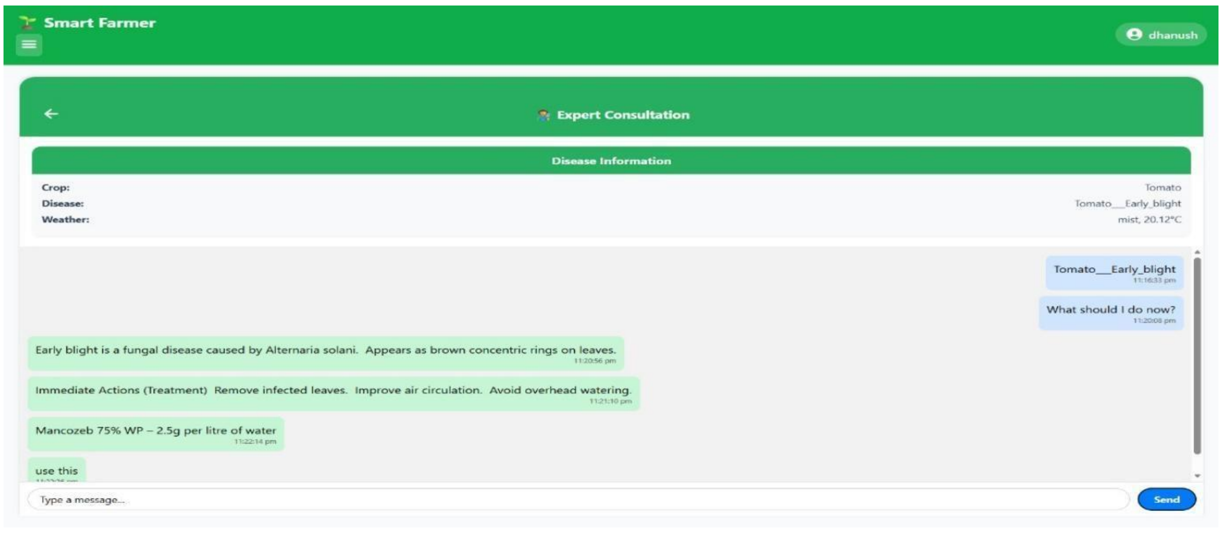
Captan 50% WP
Contact Fungicide - 500g
4.3 (834 reviews)
Effective for scab, rot, and powdery mildew
Buy Now

Copper Oxychloride 50% WP
Contact Fungicide - 500g
4.4 (956 reviews)
Controls leaf spot, blight, and downy mildew
Buy Now

Ridomil Gold
Systemic Fungicide - 250g
4.7 (1543 reviews)
Best for late blight and downy mildew control
Buy Now

Fig.A3 Treatment

Consultation:



Smart Farmer
dhanush

Expert Consultation

Disease Information

Crop:	Tomato
Disease:	Tomato___Early_blight
Weather:	mist, 20.12°C

Tomato___Early_blight
11:16:33 pm

What should I do now?
11:20:08 pm

Early blight is a fungal disease caused by Alternaria solani. Appears as brown concentric rings on leaves.
11:20:56 pm

Immediate Actions (Treatment) Remove infected leaves. Improve air circulation. Avoid overhead watering.
11:21:10 pm

Mancozeb 75% WP – 2.5g per litre of water
11:22:14 pm

use this

Type a message...
Send

Fig.A4 Consultation

III. Any auxiliary documents

<u>1. API DOCUMENTATION:</u>

This appendix provides detailed documentation of all APIs integrated into the system, including authentication (Supabase), disease prediction (FastAPI on Render), medicines retrieval, weather information, Gemini AI responses, and chat functionality. Each API description includes the endpoint, method, purpose, request format, and response format.

Authentication API (Supabase)

1. User Sign Up

Endpoint:

POST https://<project-id>.supabase.co/auth/v1/signup

Purpose:

Creates a new user account.

Request Body:

```
{
  "email": "user@example.com",
  "password": "password123"
}
```

Response:

```
{
  "id": "user_id",
  "email": "user@example.com"
}
```

2. User Login

Endpoint:

POST https://<project-id>.supabase.co/auth/v1/token?grant_type=password

Purpose:

Authenticates existing users and returns an access token.

Request Body:

```
{
  "email": "user@example.com",
  "password": "password123"
}
```

Response:

```
{
  "access_token": "<jwt_token>",
  "user": {
    "id": "user_id",
    "email": "user@example.com"
  }
}
```

Disease Prediction API (FastAPI on Render)

3. Upload Leaf Image for Prediction

Endpoint:

POST https://your-backend.onrender.com/predict

Purpose:

Receives a crop leaf image and returns the predicted disease with confidence score.

Request (Multipart Form Data):

- image: Leaf image file
- crop_type: Selected crop (e.g., "tomato")

Responses:

Low Confidence (<70%):

```
{
  "status": "low_confidence",
  "message": "Please try again"
}
```

Crop Mismatch:

```
{
  "status": "crop_mismatch",
  "message": "Please select correct crop type"
}
```

Successful Prediction:

```
{
  "status": "success",
  "disease": "Tomato Early Blight",
  "confidence": 92.5,
  "crop": "tomato"
}
```

Medicines API (Backend Database)

4. Get Medicines for Detected Disease

Endpoint:

GET https://your-backend.onrender.com/medicine/<disease_name>

Purpose:

Fetches recommended medicines stored in the backend.

Response:

```
{
  "disease": "Tomato Early Blight",
  "medicines": [
    "Trifloxystrobin + Tebuconazole",
    "Chlorothalonil",
    "Mancozeb"
  ]
}
```

Weather API

5. Weather Details for Selected Crop Area

Endpoint:

GET https://api.weatherapi.com/v1/current.json?key=<API_KEY>&q=<location>

Purpose:

Provides real-time weather conditions that affect crop diseases.

Response:

```
{
  "location": {
    "name": "Bangalore"
  },
  "current": {
    "temp_c": 29.5,
    "humidity": 76,
    "condition": {
      "text": "Partly cloudy"
    }
  }
}
```

Gemini API for Precaution & Treatment

6. Generate Precaution/Treatment Suggestions

Endpoint:

POST https://generativelanguage.googleapis.com/v1beta/models/gemini-pro:generateContent?key=<API_KEY>

Purpose:

Generates dynamic precaution or treatment steps for the detected disease.

Request Example:

```
{
  "contents": [
    {
      "parts": [
        {
          "text": "Give treatment steps for Tomato Early Blight"
        }
      ]
    }
  ]
}
```

Response Example:

```
{
  "output": "1. Remove infected leaves...\n2. Apply fungicide..."
}
```

Chat API

7. Get Messages with Expert

Endpoint:

GET https://your-backend.onrender.com/chat/<user_id>

Purpose:

Fetches chat history for the logged-in user.

Response:

```
{
  "messages": [
    {
      "sender": "expert",
      "message": "Please upload the image clearly."
    }
  ]
}
```

8. Send Message to Expert

Endpoint:

POST https://your-backend.onrender.com/chat/send

Purpose:

Sends a new user message to the expert.

Request:

```
{
  "user_id": "12345",
  "message": "What treatment should I follow?"
}
```

Response:

```
{
  "status": "sent"
}
```

2.User Guide / How to Use the Web Application:

This appendix provides a simple and clear guide for users on how to operate the **AI-Based Plant Disease Prediction Web Application**, built using React for the frontend and FastAPI for the backend.

Accessing the Website

- Open the website in any modern browser (Google Chrome recommended).
- Ensure that you have a stable internet connection.

User Authentication

2.1 Sign Up (New Users)

1. Click **"Sign Up"** on the login page.
2. Enter:
    - Email address
    - Password
3. Click **Create Account**.
4. Authentication is securely managed using Supabase.

2.2 Login (Existing Users)

1. Enter your registered email and password.
2. Click **Login** to access the homepage.

Selecting the Crop

- After logging in, the homepage displays a **crop selection** interface.
- Available crop options include:
    - Tomato
    - Potato
    - (Plus any additional crops added in your UI)

Select the crop you want to analyze and click **Continue**.

Uploading the Leaf Image

On the image upload page:

1. Click **Upload Image**.
2. Select a leaf image from your device.
3. The system automatically checks:
   - Whether the image contains a leaf
   - Whether the image is clear enough
4. Click **Submit** to send the image for prediction.

Backend Validation (No Confidence Shown to User)

The backend performs validation and returns one of the following:

1. Invalid Prediction (<70% Confidence)

Displayed message:

"Please try again"

This means the model is not confident enough.

2. Crop Mismatch

Displayed message:

"Please select correct crop type"

Occurs when the selected crop does not match the detected crop.

3. Valid Prediction

If prediction is accepted, the website shows:

- Detected disease name
- Current weather data (via Weather API)
- Buttons:
  - **Precaution**
  - **Treatment**

Precaution & Treatment (Gemini API)

When the user clicks **Precaution** or **Treatment**:

- A prompt is sent to the Gemini API.
- The API returns:
  - Step-by-step precaution advice
  - Recommended treatment procedures

The generated information is displayed neatly on the page.

View Medicines

- The system provides medicine recommendations for each detected disease.
- On **Find Medicines**, the user can:
  - View medicines retrieved from the FastAPI backend
  - Search for medicines
- Medicines are stored in a JSON/Database on the backend.

Chat With Expert

The **Chat page** allows real-time text communication with an agricultural expert.

Features include:

- Typing and sending messages
- Viewing previously sent messages
- Backend storage of all conversations per user

Navigation Menu

At the bottom-left corner of the website, the navigation menu includes:

- **Home** – Crop selection and disease detection
- **Chat** – Communication with expert
- **Find Medicines** – View and search medicines
- **Logout**

Logout

- Clicking **Logout** redirects the user to the Login page
- The session ends immediately.

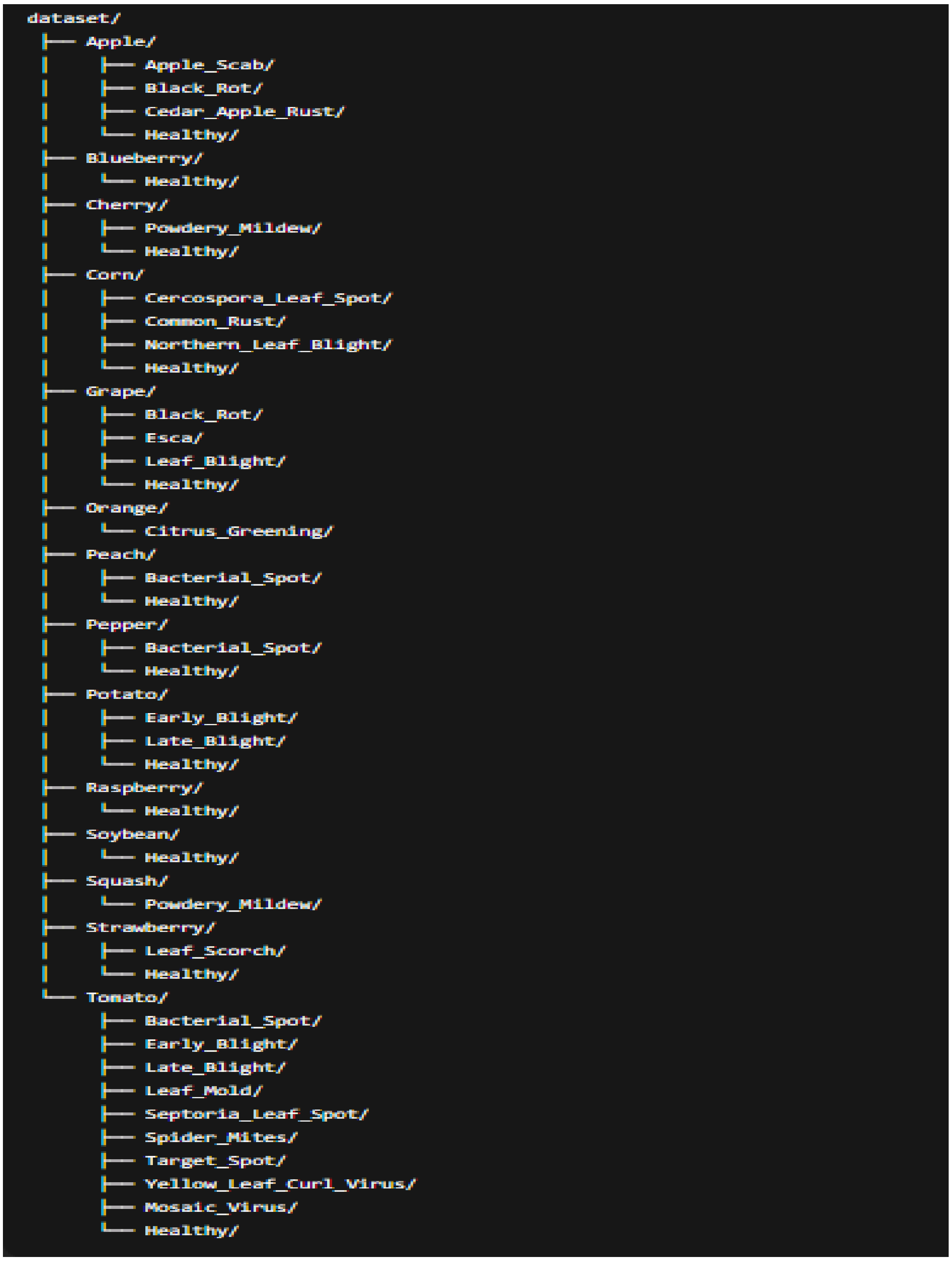
Ending the Session

- Closing the browser window

  or
- Clicking **Logout**

…ends the session securely.

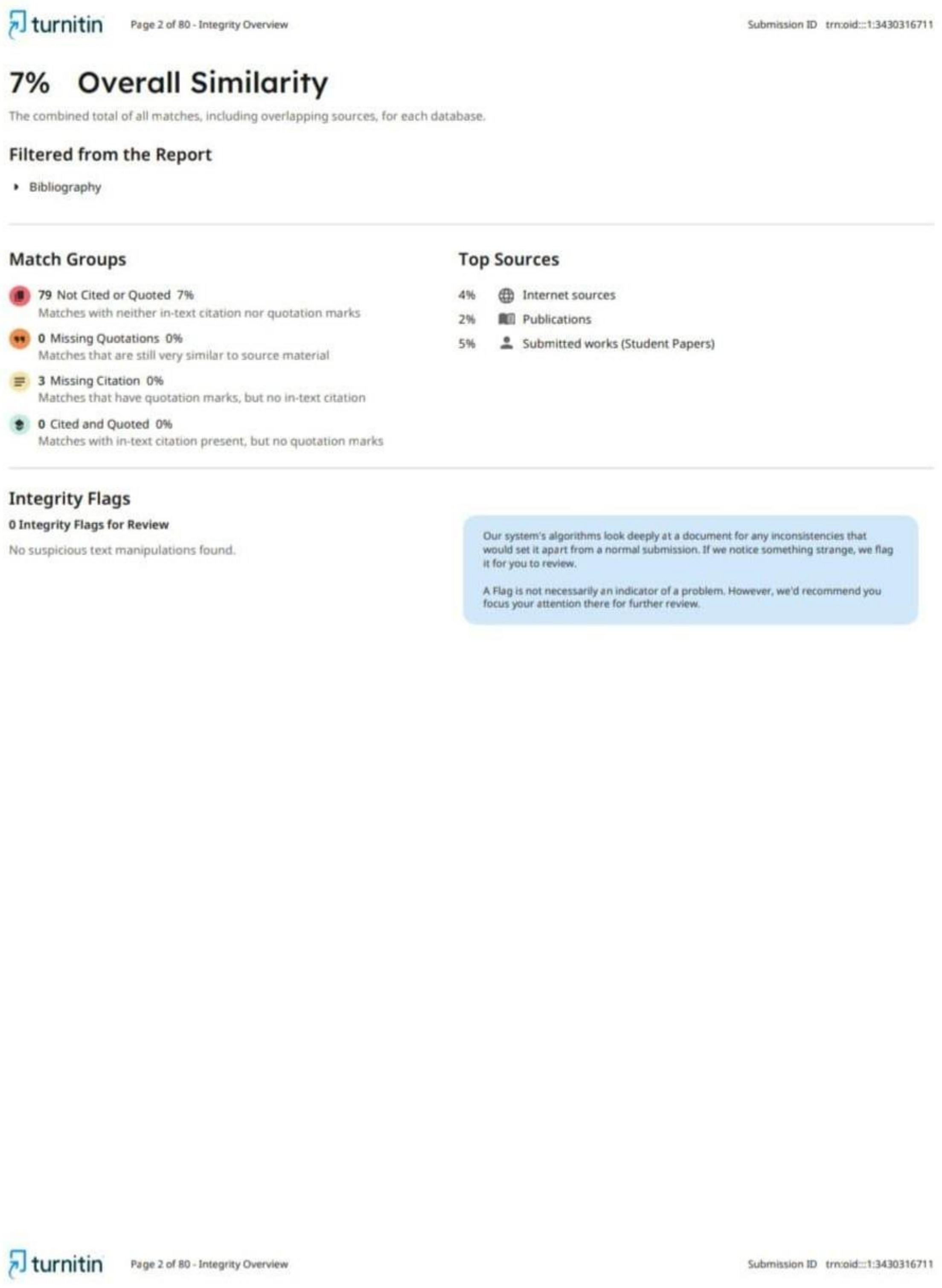
IV. Dataset: “New Plant Diseases Dataset”



```
dataset/
├── Apple/
│   ├── Apple_Scab/
│   ├── Black_Rot/
│   ├── Cedar_Apple_Rust/
│   └── Healthy/
├── Blueberry/
│   └── Healthy/
├── Cherry/
│   ├── Powdery_Mildew/
│   └── Healthy/
├── Corn/
│   ├── Cercospora_Leaf_Spot/
│   ├── Common_Rust/
│   ├── Northern_Leaf_Blight/
│   └── Healthy/
├── Grape/
│   ├── Black_Rot/
│   ├── Esca/
│   ├── Leaf_Blight/
│   └── Healthy/
├── Orange/
│   └── Citrus_Greening/
├── Peach/
│   ├── Bacterial_Spot/
│   └── Healthy/
├── Pepper/
│   ├── Bacterial_Spot/
│   └── Healthy/
├── Potato/
│   ├── Early_Blight/
│   ├── Late_Blight/
│   └── Healthy/
├── Raspberry/
│   └── Healthy/
├── Soybean/
│   └── Healthy/
├── Squash/
│   └── Powdery_Mildew/
├── Strawberry/
│   ├── Leaf_Scorch/
│   └── Healthy/
└── Tomato/
    ├── Bacterial_Spot/
    ├── Early_Blight/
    ├── Late_Blight/
    ├── Leaf_Mold/
    ├── Septoria_Leaf_Spot/
    ├── Spider_Mites/
    ├── Target_Spot/
    ├── Yellow_Leaf_Curl_Virus/
    ├── Mosaic_Virus/
    └── Healthy/
```

Fig.A5 Dataset

V. Similarity Report



turnitin Page 2 of 80 - Integrity Overview Submission ID trn:oid:::1:3430316711

7% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Filtered from the Report

- Bibliography

Match Groups

- **79** Not Cited or Quoted 7%
  Matches with neither in-text citation nor quotation marks
- **0** Missing Quotations 0%
  Matches that are still very similar to source material
- **3** Missing Citation 0%
  Matches that have quotation marks, but no in-text citation
- **0** Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

Top Sources

4%	Internet sources
2%	Publications
5%	Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

No suspicious text manipulations found.

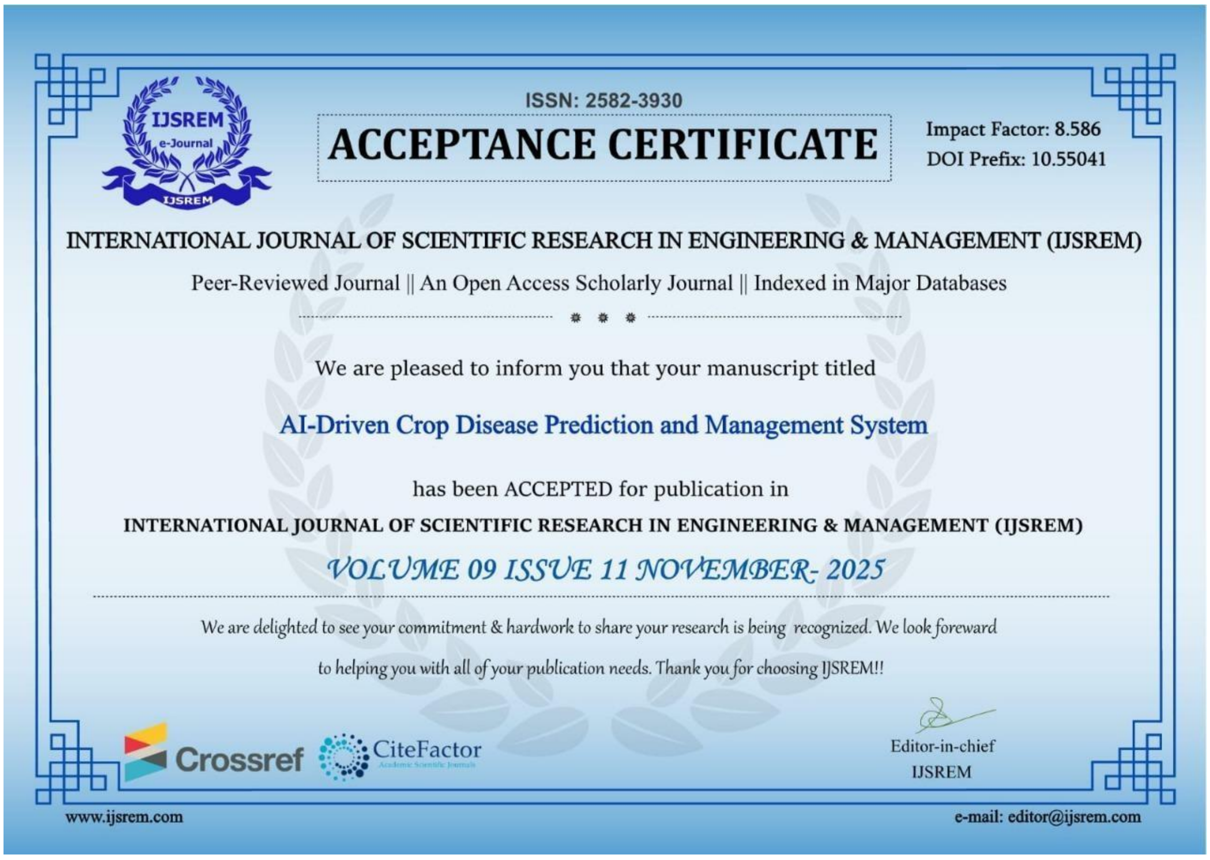
Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

turnitin Page 2 of 80 - Integrity Overview Submission ID trn:oid:::1:3430316711

Fig.A6 Similarity Report

VI. Publications Acceptance letter



ISSN: 2582-3930

ACCEPTANCE CERTIFICATE

Impact Factor: 8.586
DOI Prefix: 10.55041

INTERNATIONAL JOURNAL OF SCIENTIFIC RESEARCH IN ENGINEERING & MANAGEMENT (IJSREM)

Peer-Reviewed Journal || An Open Access Scholarly Journal || Indexed in Major Databases

We are pleased to inform you that your manuscript titled

AI-Driven Crop Disease Prediction and Management System

has been ACCEPTED for publication in

INTERNATIONAL JOURNAL OF SCIENTIFIC RESEARCH IN ENGINEERING & MANAGEMENT (IJSREM)

VOLUME 09 ISSUE 11 NOVEMBER- 2025

We are delighted to see your commitment & hardwork to share your research is being recognized. We look foreward to helping you with all of your publication needs. Thank you for choosing IJSREM!!

Editor-in-chief
IJSREM

www.ijsrem.com

e-mail: editor@ijsrem.com

Fig.A7 Acceptance letter