KISAN MITRA: SMART FARMING ASSISSTANT

MAJOR PROJECT REPORT

SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE
AWARD OF THE DEGREE OF

BACHELOR OF TECHNOLOGY

(Computer Science and Engineering)

Submitted By:

Diya Baweja (2203425)

Gurjot Kaur (2203433)

Rahul Sachdeva (2203536)

Submitted To.:

Prof. Hardeep Singh Kang

Project Guide

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GURU NANAK DEV ENGINEERING COLLEGE

LUDHIANA, 141006

May, 2025

List of Figures

Contents

1 Introduction

1.1 Introduction to Project

With a substantial part ofthe AGRICULTURE according to the plant, agriculture forms backbone of the economy of India for the population. But it is not always easyfor the farmers as they struggle with plant pests, deals weather circumstances, uncertainty, andpoor availability of information and decision making that result into crop production loss and decreased yield. The proliferation of intelligent services Intoday' s world of rapid technological development, the ex-plo of intelligent form the reasonable goal of many information systems.gent systems which canhelp the farmers to take right decision. With the rapid development of machine learning technology, automated plantdisease detection has been.a promising solution. However, existing models struggle with real-world deployment due to issues like high computational complexity, overfitting to benchmark datasets, and lack of generalization to diverse environmental conditions.

Kisan Mitra is a smart farming assistant application with an intention to empower farmers with the use oflisted smart farming services- machinelearning (ML), image processing, and live weather data. The system enables the farmer to take pictures of thediseased crop and get an instant disease diagnosis classification with propertherapeutics. It also makes use of weather APIs inorder to provide advance warning ofoutbreaks of disease or adverse weather. This holisticapproach makes the link between ancient knowledge and technology modernization, leading toward agricultureand improving crop productivity.

1.2 Project Category

The project falls under the **Research** category with elements of **Application-Based Project**. It encompasses practical implementation using real-world datasets, machine learning techniques, and API integrations for mobile-based deployment.

- **Application-Based**: It is deployed as a mobile app for real-world usage by farmers.
- **Research-Based**: It involves training and deploying ML models for image classification and knowledge distillation.

1.3 Problem Formulation

- Despite the advancements in agriculture, many farmers still rely on traditional and manual methods to identify plant diseases. Misdiagnosis, delays in treatment, and lack of access to expert guidance continue to affect crop health. Moreover, the unpredictable nature of weather significantly contributes to disease outbreaks, which are often unnoticed until significant damage has occurred.

- The core problem is the lack of an intelligent, accessible, and real-time solution that can diagnose plant diseases and suggest remedies while also factoring in climatic conditions. This project addresses the following formulated problem statement:

  "To develop a mobile-based smart farming assistant that accurately classifies plant diseases from leaf images using machine learning and provides contextual recommendations and weather-based alerts to farmers."

1.4 Identification/Recognition of Need

- The farmers need to diagnose the disease quickly and accurately to prevent crop losses.

- There are no professional agricultural diagnosis tools available in the countryside.

- There is a requirement for inexpensive and user-friendly digital toolsthat are designed for use by non-technical users.

- The rise of climate change has made disease outbreaks more frequent, and warnings critical.

- Existing applications either focus on generic recommendations or only on weather alerts without contextual intelligence.

Recognizing these factors highlights a strong necessity for a smart assistant that can support decision-making and enhance yield by proactively assisting farmers.

1.5 Existing System

Several plant disease identification apps and agricultural platforms exist; however, they suffer from various limitations:

- **Less Accuracy**: Several applications do not use advanced image processing tools.ML models resulting in false classification.
- **Generic Recommendations**:User recommendations are not alwayspersonalized toward the detected disease or local context.
- **Real Time Alerting Unavailability**:There are no systems currently in use that use weatherAPIs or out geolocationdata to issue pre-visions.
- **No Offline Support**: The majority of the platform won't work without a very good internetconnection, hence.Maximum Time Frame Zero: We all know that the preparation for UPSC an exam requires time of more than 1 year is even short.
- **Lack of Integration**: Most systems treat disease identification, recommendation, and alert generation as separate services rather than an integrated pipeline.
- **Mobile Device:** ARM-based smartphone or tablet with a minimum of 2 GB RAM.
- **Server/Cloud:** A cloud-based server or on-premises machine capable of hosting the Node.js backend and TensorFlow ML inference engine.

1.6 Objectives

The main objectives of the *Kisan Mitra* project are:

1. To develop an application system supporting intelligent plant disease identification by analysing plant leaf images using machine learning techniques.
2. To provide a machine learning based recommendation system that suggests targeted remedies and preventive measures based on classified disease.
3. To provide alerts to farmers in textual and graphical form using weather and location monitoring about potential disease outbreaks or climate problems.

1.7 Proposed System

The proposed system—*Kisan Mitra*—is a mobile-based smart farming assistant built using a Flutter frontend, a Node.js backend, and a machine learning module powered by TensorFlow.The system ensures timely information delivery, minimizes the need for manual diagnosis, and reduces dependency on agricultural experts in remote areas.
Key components of the system include:

- **Plant Disease Detection Module**: Utilizes a knowledge-distilled machine learning pipeline (teacher-student architecture) to classify diseases from plant leaf images.
- **Recommendation Engine**: Maps identified diseases to a structured database containing AI-powered recommendations, including treatment protocols, pesticide usage guidelines, and preventive measures, while also addressing farming-related queries.
- **Alert Generation**: Fetches real-time weather and location data to trigger alerts for potential disease risks or climate stress.
- **Mobile Interface**: Offers an intuitive interface for farmers to interact with the system, upload images, view results, and receive alerts in both textual and graphical formats.

1.8 Unique Features of the Proposed System

Kisan Mitra introduces several novel features that differentiate it from existing solutions:

- **Teacher-Student ML Architecture**: A hybrid learning strategy that ensures high accuracy while maintaining lightweight inference on mobile devices.
- **End-to-End Integration**: Unified pipeline from disease detection to treatment and alerting within a single platform.
- **Real-Time Weather-Based Alerts**: Contextual notifications based on climate and geolocation data enhance farmer readiness.
- **Modular Design**: Easy integration of new crops, disease types, and environmental factors without major redesign.
- **User-Centric Dashboard**: Clean, minimal, and highly accessible UI designed for tech-novice farmers.

2 Requirement Analysis and System Specification

2.1 Feasibility Study

Before initiating the development of *Kisan Mitra: Smart Farming Assistant*, a detailed feasibility study was conducted to evaluate the practicality and viability of the project. The assessment focused on three major aspects: technical, economic, and operational feasibility.

2.1.1 Technical Feasibility

The project leverages existing, proven technologies and frameworks such as:

- Flutter for cross-platform mobile app development.
- Node.js for backend REST API development.
- TensorFlow/TensorFlow Lite for machine learning model training and deployment.
- OpenWeatherMap API and Geolocation services for environmental data integration.

These tools are well-supported, documented, and compatible with the mobile-first deployment strategy. Moreover, the use of a student-teacher knowledge distillation approach ensures that the ML model is both accurate and lightweight, allowing real-time inference on mobile devices with minimal resource consumption. Since the team possesses strong proficiency in these tools, the technical risk is considered low.

2.1.2 Economic Feasibility

The system's potential scalability and cost-effectiveness are key components of its economic viability. Important factors include:

- Development is done with open-source tools like Flutter, Node.js, and Tensor-Flow, which lowers licensing costs.
- Firebase's free tier, which is appropriate for prototypes, makes it suitable for cloud hosting.Vercel or other free hosting services can be used for deployment, guaranteeing low development costs.

- Minimal hardware requirements ensure the app runs on budget smartphones commonly used by farmers.
- Scalability permits expansion to a larger user base without a notable increase in infrastructure costs; minimal hardware requirements guarantee that the app operates on low-cost smartphones frequently used by farmers.

2.1.3 Operational Feasibility

With end users—farmers—in mind, the system is designed to ensure:

- Real-time weather updates are provided via the OpenWeatherMap API, guaranteeing dependable and effective system operation.
- The most recent agricultural news is retrieved via the Newsdata.io API, guaranteeing users timely and pertinent information;
- The most recent agricultural updates are delivered via a news reports API, guaranteeing users remain informed with the least amount of operational complexity.

According to preliminary prototype testing, the system can be swiftly embraced with little training. As a result, even in isolated and underserved areas, the project is operationally very feasible.

2.2 Software Requirement Specification

The behavior, features, and limitations of the intended system are all thoroughly described in a Software Requirement Specification (SRS) document. The SRS for Kisan Mitra: Smart Farming Assistant is organized in accordance with accepted software engineering practices and comprises the subsequent subsections:

2.2.1 Data Requirements

Data is at the heart of how the Kisan Mitra system works. It powers the machine learning) model, the recommendation engine, as well as weather-dependent alert creation.

- **Image Data for ML Model**:
  - Image Data for ML Model: High-quality labeled leaf images from crops such as tomato, potato, appleand more.
  - Categories will consist of healthy leaves and leaves with disease (early blight,late blight, leaf mold).
  - Dataset Download (385.16 MB) Dataset Details - Collectedfrom sources such as PlantVillage, government agriculturalrepository,and curated open datasets.
  - Preprocessing processes areresize, augmentation (rotation, flipping, brightness adjustment and format conversion.
- **Recommendation Data**:
  - Data is at the heart of how the Kisan Mitra system works. It powers the machine learning) model, the recommendation engine, as wellas weather-dependent alert creation.
  - Each entry contains:
    * Disease name
    * Recommended pesticide/fungicide
    * Organic alternatives (if applicable)
    * Preventive strategies
    * Dosage and safety instructions

- **Environmental and User Data**:
  - Current weather and weather forecast all around the world, weather and time for millions oflocations all over the world fetchedfrom OpenWeatherMap API.
  - Geolocation-coordinates of users need to be used“ for context basedweather alerting.”
  - User profile information: name, crops, location (securely stored),contact information..

2.2.2 Functional Requirements

Functional requirements tellus what the system needs to do. The main featuresare

- **Image Upload and Disease Detection**:
  - User can take photo or upload the image of a plant leaf.
  - Preprocessed image is sent to the trained ML model.
  - Model classifies the image into a disease class or “Healthy”..
- **Recommendation Generation**:
  - System queries the database once a disease is detected.
  - AI-powered tailored suggestions including remedy, chemical treatment, organic solution, and best practices are displayed.
- **Weather and Location-Based Alerts**:
  - Weather data fetched based on GPS coordinates.
  - If conditions are favorable for disease outbreak, alerts are generated and displayed.
- **User Registration and Profile Management**:
  - Users can register with basic details and crop information.
  - Profile can be updated and used to personalize recommendations and alerts.
- **Backend Operations**:

- Backend APIs manage communication between app, model, and database.

2.2.3 Performance Requirements

Performance targets include:

- Model Inference Time: approximately 44 ms on average, with each image taking less than 100 ms.
- Mobile RAM Usage: Low-end Android devices use less than 200MB of mobile RAM.
- Less than two to three seconds is the API response time.
- App Load Time: App load times are less than 5 seconds for the initial load and less than 2 seconds for subsequent loads.
- Battery Efficiency:When not required, no demanding background operations are performed.

2.2.4 Dependability Requirements

- Accuracy: knowledge distillation yielded a 98
- Robustness: Manages partially visible and noisy images.
- Error Handling: Error events are handled with friendly user interface messages.

2.2.5 Maintainability Requirements

- A front-end, back-end, and machine learning codebase that is modular.
- Branching techniques and Git are used for version control
- APIs that can be expanded for upcoming voice control and chatbot integrations.

2.2.6 Security Requirements

- Password-based authentication for secure login.
- Bcrypt for encrypting sensitive user data.
- Authentication for secure API access.

2.2.7 Look and Feel Requirements

- Simple interface with minimal learning curve.
- Large fonts, readable contrast themes for all screen types.

2.3 SDLC Model to Be Used

Selected Model: **Agile Development Model**

Agile methodology emphasizes flexibility, collaboration, and iterative delivery, which is perfectly suited for the evolving, modular nature of the Kisan Mitra application.

2.3.1 Key Reasons for Choosing Agile

1. **Rapid Iterative Development**:
   - ML model, UI, and backend developed in parallel.
   - Each sprint delivers a working application subset.
2. **Modular Feature Releases**:
   - Sprint 1: Disease Detection
   - Sprint 2: Recommendation System
   - Sprint 3: Weather Alert System
3. **Cross-Functional Collaboration**:
   - Developers, data scientists, and designers work together.
4. **Adaptive to Change**:
   - New crops, diseases, or UI modules added easily.

2.3.2 Agile Implementation Plan

- **Sprint 1** (Week 1–2): UI Design + ML model setup.
- **Sprint 2** (Week 3–4): Image upload + classification.
- **Sprint 3** (Week 5–6): Recommendation database + display.

- **Sprint 4** (Week 7–8): Weather API + alert integration.
- **Sprint 5** (Week 9–10): Profile management + multi-language support.
- **Sprint 6** (Week 11–12): Testing, optimization, final deployment.

Each sprint ends with:

- Sprint review meetings (demo to stakeholders).
- Sprint retrospective for identifying improvements.
- Backlog grooming for reprioritizing upcoming tasks.

Conclusion: The Agile model's adaptability and responsiveness ensure that *Kisan Mitra* evolves based on user needs and real-world challenges. This iterative cycle is ideal for delivering a robust, user-focused agricultural assistant application.

3 System Design

3.1 Design Approach

The project uses an Object-Oriented Design (OOD) methodology because of its scalability, modularity, and reusability. Object-oriented design enables each module to be modeled as a distinct entity with clearly defined responsibilities because Kisan Mitra is made up of various interacting components, including the image classifier, recommendation engine, weather alert system, and user interface.Additionally, OOD works well with the ML pipeline, the Node.js backend, and Flutter (which makes use of the object-oriented language Dart).
This approach is chosen for the following reasons:

- **Modularity and Encapsulation**: Classes and objects contain related functionalities, minimizing the effect of changes made to one module on another.
- **Reusability and Maintainability**: Components such as the ML model wrapper and alert generator can be reused and easily maintained.
- **Ease of Integration**: Well-defined interfaces make it easier to manage component integration..
- **Scalability**: New features can be incorporated seamlessly by extending existing classes, ensuring scalability.

3.2 Detail Design

The detailed design of *Kisan Mitra* revolves around meeting three main objectives:

1. Disease Detection
2. Recommendation System
3. Weather-Based Alert System

Each objective is handled via individual but integrated modules that communicate using REST APIs and share access to a centralized database.

3.2.1 Disease Detection Module

- **Input**: Image of a plant leaf.
- **Process**:
  - Image is preprocessed (resized, normalized).
  - Forwarded to ML model for classification.
  - Model uses a lightweight CNN trained using teacher-student distillation.
- **Output**: Predicted disease name and confidence score.

Below is a high-level workflow diagram illustrating the model training process:

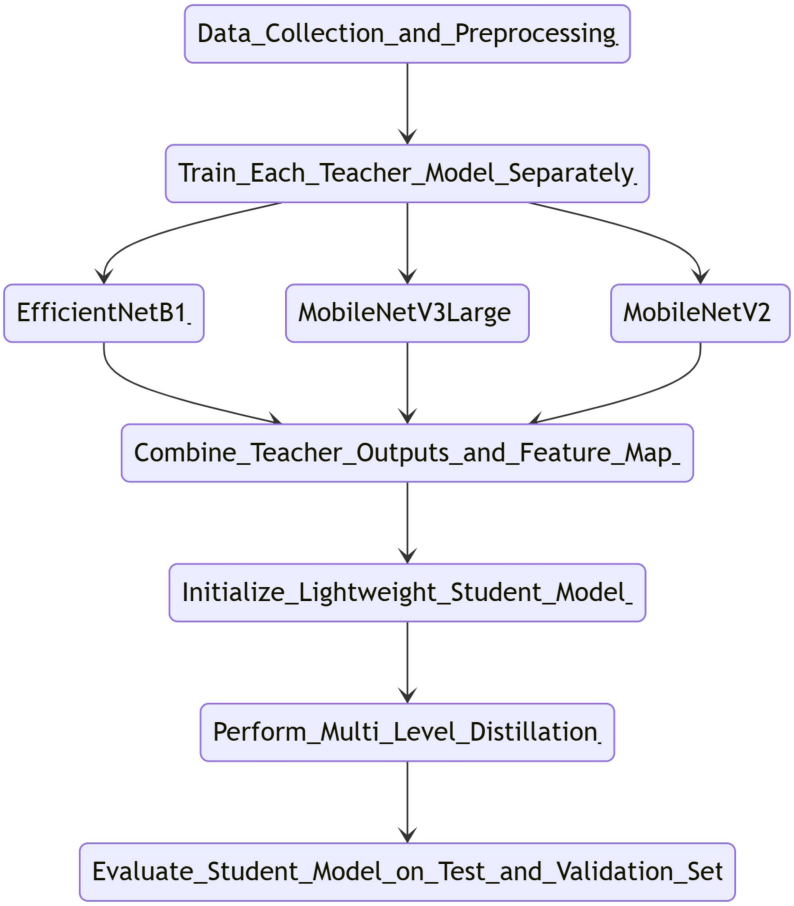


Figure 1: Model Training Workflow

3.2.2 Data Collection & Preprocessing

- **Dataset Integration:**
  - combining pictures from several sources (like PlantVillage) into a single direction.
  - Oversampling is done to balance classes and create combined class names to prevent conflicts.
- **Image Augmentation:**
  - Model generalization is enhanced by methods like flipping, rotation, and color correction.

3.2.3 Teacher Models

- **Architectures Used:** EfficientNetB1, MobileNetV3Large, and MobileNetV2 are the architectures utilized.
- **Feature Extraction:** To extract complementary features from the dataset, each teacher model is trained separately.
- **Ensemble Fusion:** To direct the student model, the teacher models' outputs and intermediate feature maps are dynamically weighted (using learnable weights) and fused.

3.2.4 Student Model

- **Architecture:** Squeeze-and-excitation blocks and depthwise separable convolutions are used in this lightweight, customized Convolutional Neural Network (CNN).
- **Distillation Strategy:**
  - **Output-Level Distillation:** Aligns the student's softened predictions (using a high temperature) with the teacher ensemble's output.
  - **Feature-Level Distillation:** Matches intermediate feature maps between the student and the fused teacher features.

Benefits: This multi-level approach guarantees that the student model maintains a small form factor for mobile deployment while retaining the rich semantic information that the teacher ensemble has learned.

3.2.5 Evaluation & Inference

- **Metrics:** F1 score, recall, accuracy, precision, inference latency, and model size.
- **Deployment Considerations:** The student model is perfect for mobile deployment because it is tuned for low latency and minimal resource consumption.

3.2.6 Recommendation Engine

- Retrieves treatment information associated with the disease that has been identified.
- Outcomes include dosage, preventative measures, and chemical and organic treatments.
- Provides recommendations driven by AI and answers questions about farming.

3.2.7 Weather and Alert System

- Using location services, the app retrieves current weather information.
- The backend compares weather information with conditions that can cause disease.
- If high risk is identified, alerts are generated and displayed.

3.2.8 Integration Logic

- The application manages image uploading, user interaction, and result visualization.
- The database, ML model, and frontend are coordinated by the backend.
- Disease forecasts and alert logs are kept for later review.

3.3 System Deployment Architecture

Below is the high-level architecture diagram for system deployment:

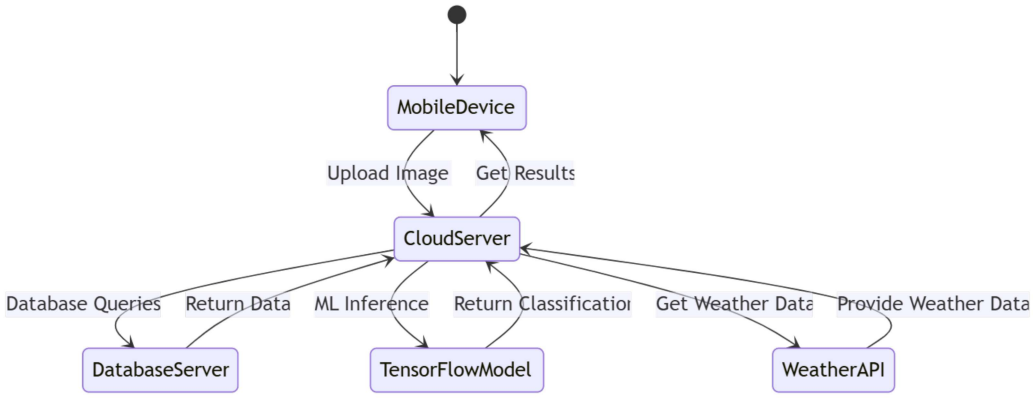


Figure 2: System Deployment Architecture

- **Mobile Device (Flutter):** Provides a frontend interface for user interactions. Currently, the design is in place without active functionalities.
- **Cloud Server (Node.js):** Manages API requests and orchestrates interactions between the mobile app, ML model, and weather service.
- **Database Server:** Stores user profiles, logs, and the recommendations database.
- **Weather API:** Supplies real-time climate data for alert generation.

3.4 Database Design (E-R Diagram)

The following Entity-Relationship (ER) diagram represents the database schema:

Users Table

- **id (integer, primary key)**: Unique identifier for each user.
- **name (varchar)**: Stores the name of the user.
- **phone (varchar)**: Contact number of the user.
- **profile_image (varchar)**: URL or path to the user's profile picture.
- **location (varchar)**: User's general location.
- **city (varchar)**: User's city.
- **state (varchar)**: User's state.

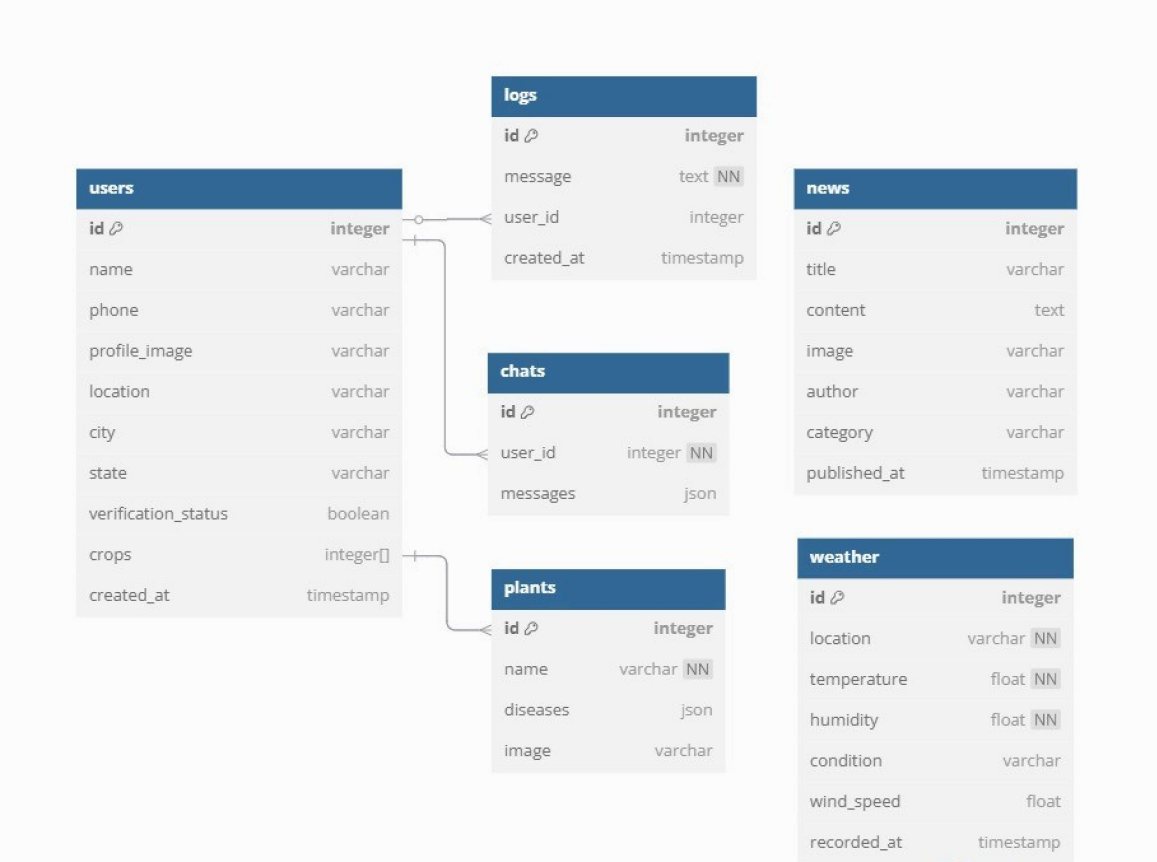


Figure 3: E-R Diagram for Database

- **verification_status (boolean)**: Indicates whether the user is verified.
- **crops (integer[])**: Array of crops associated with the user.
- **created_at (timestamp)**: Timestamp of user account creation.

Logs Table

- **id (integer, primary key)**: Unique identifier for each log entry.
- **message (text, NOT NULL)**: Log message details.
- **user_id (integer, foreign key)**: References the user who created the log.
- **created_at (timestamp)**: Timestamp of log creation.

Chats Table

- **id (integer, primary key)**: Unique identifier for each chat entry.

- **user_id (integer, foreign key, NOT NULL)**: References the user associated with the chat.
- **messages (json)**: Stores chat messages in JSON format.

Plants Table

- **id (integer, primary key)**: Unique identifier for each plant.
- **name (varchar, NOT NULL)**: Name of the plant.
- **diseases (json)**: List of diseases affecting the plant.
- **image (varchar)**: URL or path to an image of the plant.

News Table

- **id (integer, primary key)**: Unique identifier for each news article.
- **title (varchar)**: Title of the news.
- **content (text)**: Main content of the news.
- **image (varchar)**: URL or path to an image related to the news.
- **author (varchar)**: Name of the news author.
- **category (varchar)**: Category of the news (e.g., agriculture, weather).
- **published_at (timestamp)**: Timestamp of when the news was published.

Weather Table

- **id (integer, primary key)**: Unique identifier for each weather record.
- **location (varchar, NOT NULL)**: Location where the weather data was recorded.
- **temperature (float, NOT NULL)**: Temperature at the recorded location.
- **humidity (float, NOT NULL)**: Humidity level at the location.

- **condition (varchar)**: Weather condition (e.g., sunny, rainy).
- **wind_speed (float)**: Wind speed at the location.
- **recorded_at (timestamp)**: Timestamp of weather data recording.

3.5 Notification Feature

To provide alerts to farmers in textual and graphical form using weather and location monitoring about potential disease outbreaks or climate problems.

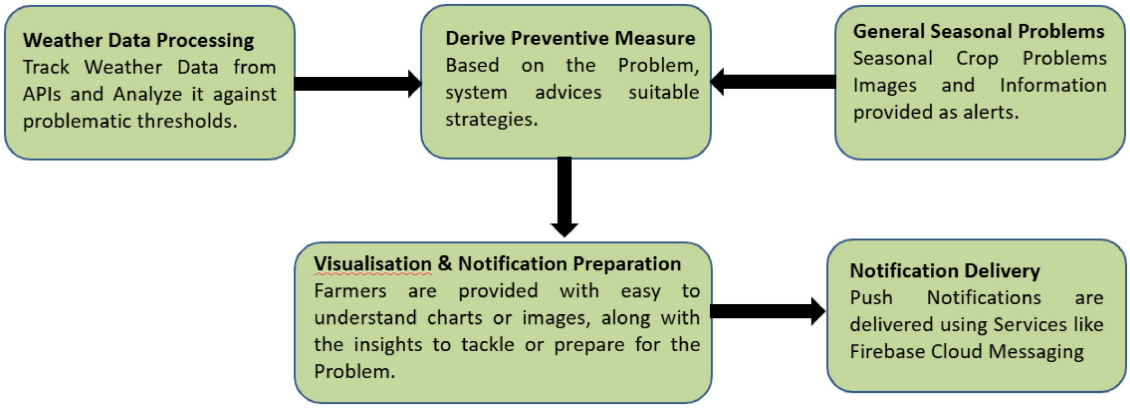


Figure 4: Notification Feature

3.6 User Interface Design

Kisan Mitra's Flutter-built user interface design places a high value on modularity, accessibility, and simplicity.

3.6.1 Design Principles Followed

- Simple Design: Easy to use for tech-inexperienced farmers.
- Accessibility features include multilingual options, color coding, and large buttons.
- Responsiveness: Adjusts to different screen sizes.
- Consistency: consistent color schemes, fonts, and padding

3.6.2 Key Screens and Their Roles

- **Welcome and Onboarding Screens**: Explain the usability and features of the app.
- **Login and Email Verification**: streamlined user authentication.
- **Home Dashboard**: news, crop guide, diagnosis tool, weather information, and crop status.
- **Digital Assistant Interface**: Chatbot for agricultural assistance.
- **Crop Guide**: Information on diseases, growth stages, and climate requirements.
- **Detection Result Screen**:Accurate and recommended disease prediction.
- **Settings Page**: PPrivacy Policy, Contact Support, FAQ, and Profile Management.

3.6.3 Design Tools Used

- Flutter DevTools for troubleshooting.
- An emulator for Android Studio for testing.
- Figma (optional) for UI design wireframing.

3.7 Methodology

The development followed the Agile Software Development Methodology.

3.7.1 Agile Workflow Overview

- Sprints divided into 2-week cycles.
- Daily standups to monitor progress.
- Backlog tracking through Trello.
- Sprint reviews for continuous feedback.

3.7.2 Sprint Timeline and Milestones

- **Sprint 1**: Project setup, Flutter UI prototyping.
- **Sprint 2**: Backend setup, Weather API integration.
- **Sprint 3**: ML model training and TFLite deployment.
- **Sprint 4**: Image diagnosis and result integration.
- **Sprint 5**: Crop guide, assistant UI, weather alert integration.
- **Sprint 6**: Testing, optimization, and final deployment.

3.7.3 Version Control and CI/CD

- Git and GitHub used for version control.
- Branching strategy:
  - **main**: Stable production branch.
  - **dev**: Active development.
  - **feature**: Individual feature branches.
- Firebase Hosting (optional) and APK testing for deployment.

3.7.4 Tools and Technologies Recap

- Frontend: Flutter (Dart)
- Backend: Node.js, Express.js
- Machine Learning: Python, TensorFlow, TensorFlow Lite
- Database: MongoDB
- APIs: OpenWeatherMap, Google Geolocation
- IDEs: Visual Studio Code, Android Studio
- Version Control: Git and GitHub

3.7.5 Testing and Evaluation

- Unit Testing for frontend and backend components.
- Integration Testing to verify component connectivity.
- Model evaluation using metrics like accuracy, precision, and recall.
- User Acceptance Testing (UAT) to gather real-world feedback.

Conclusion:

The design and development process of *Kisan Mitra* combined object-oriented design, modular architecture, and Agile methodology to build a scalable, user-friendly, and robust smart farming assistant.

4 Implementation and Testing

4.1 Introduction to Languages, IDEs, Tools and Technologies Used

Front-end and back-end technologies, cloud-based tools, machine learning frameworks, and APIs are all used in the creation of Kisan Mitra. To accommodate users with different levels of technical expertise, the implementation is guided by the principles of performance, cross-platform compatibility, and modularity.

4.1.1 Programming Languages

1. **Dart**:
    - Using the Flutter framework, Dart is used for front-end development.
    - It was selected because of its robust community, rich widget library, and support for reactive user interfaces.
    - Both iOS and Android to be deployed using a single codebase.
2. **JavaScript**:
    - Used for backend development with Node.js and Express.
    - Powers REST APIs and handles routing, logic, and database communication.
3. **Python**:
    - Employed for machine learning model development.
    - Supports the training and evaluation of the plant disease classifier using TensorFlow and Keras.

4.1.2 IDEs Used

1. **Visual Studio Code**:
    - Used primarily for backend and Flutter development.

- Extensions like Flutter, ESLint, REST Client, and GitLens enhanced productivity.

2. **Android Studio**:
   - Used for testing and deploying the mobile application on Android devices.
   - Provides an emulator for UI testing and APK generation.

4.1.3 Tools and Frameworks

- Flutter: Framework for building native-like UI for both Android and iOS platforms from a single codebase.
- Node.js + Express.js: Backend REST API development.
- MongoDB: NoSQL database to store user information, image logs.
- TensorFlow/Keras: Deep learning frameworks for building and training the teacher-student CNN model.
- TensorFlow Lite: For mobile deployment of ML models.
- Postman: For API testing and debugging.
- Firebase: Optional hosting, storage, and push notifications.
- OpenWeatherMap API: For real-time weather data integration.
- Git & GitHub: For version control and collaboration.

4.2 Algorithm / Pseudocode Used

The core algorithms implemented in *Kisan Mitra* are described below.

4.2.1 High-Level Algorithm for Disease Detection

Algorithm: Disease Detection using Knowledge Distillation

- **Input**: Leaf Image I
- **Output**: Disease Label D and Confidence Score C

Steps:

1. Preprocess Image I
    - Resize to 224x224 pixels
    - Normalize pixel values
    - Convert to Tensor
2. Forward Image I to Ensemble Teacher Models
    - T1 = EfficientNetB1
    - T2 = MobileNetV3Large
    - T3 = MobileNetV2
3. Extract outputs and feature maps.
4. Fuse Teacher Outputs
    - Weighted fusion of softmax probabilities and feature representations.
5. Train Student Model (Lightweight CNN)
    - KL-Divergence for soft output matching.
    - L2 Loss for feature map alignment.
6. Evaluate Model using Accuracy, F1-Score, Precision, Recall.
7. Deploy Trained Student Model to Mobile using TensorFlow Lite.
8. On User Upload:
    - Forward image to model
    - Get predicted class and probability
    - Return result to UI

4.2.2 Farming Assistant Response Generation (Rule-Based + ML Hybrid)

Algorithm: Farming Assistant Query Response

- **Input**: User Query Q
- **Output**: Answer A

Steps:

1. Preprocess Q (lowercase, remove stop words, tokenize)
2. Intent Classification
    - Rule-based matching
    - ML-based prediction fallback (e.g., logistic regression or BERT)
3. Entity Extraction (crop names, weather conditions)
4. Match Intent to Response
5. Return Response A

4.2.3 Crop Suggestion Algorithm (Season + Location Based)

Algorithm: Crop Suggestion Based on Season & Region

- **Input**: Location L, Current Month M
- **Output**: List of Suitable Crops C_list

Steps:

1. Fetch seasonal crop data.
2. Filter crops by region and season.
3. Rank crops by demand/yield.
4. Return C_list.

4.2.4 Alert Generation Algorithm (Weather-Based)

Algorithm: Alert Generation

- **Input**: User's Geolocation (Latitude, Longitude)

Steps:

1. Fetch weather data from OpenWeatherMap API.
2. Parse temperature, humidity, condition.
3. If humidity exceeds threshold and condition is Rainy/Cloudy:
    - Cross-check with disease risk database.
    - Generate warning alert.
4. Store alert log and notify user.

4.3 Testing Techniques

Testing ensures the system functions reliably under different scenarios.

4.3.1 Unit Testing

- Testing individual components like:
    - Image preprocessing
    - Disease classification
    - Recommendation querying
    - API endpoints
- Tools used: Jest, Flutter Test framework

4.3.2 Integration Testing

- Validated end-to-end flow:
    - Image Upload → Classification → Recommendation
    - Location Access → Weather API → Alert generation

4.3.3 System Testing

- Full workflow testing in real network and mobile device conditions.

4.3.4 Performance Testing

- KPIs measured:
  - Inference time: ∼44ms
  - API response: $\leq$ 3 seconds
  - App start time: ∼1.5 seconds

4.3.5 Model Testing

- Test set evaluation:
  - Accuracy: 98.08%
  - Precision, Recall, F1-Score: ∼98%
  - Confusion matrix used to analyze misclassifications.

4.3.6 User Acceptance Testing (UAT)

- Conducted with farmers and agriculture students.
- Feedback incorporated to refine UI, text, and alerts.

4.4 Test Cases Designed for the Project Work

A few representative test cases are summarized below:

- **TC-01**: Upload healthy leaf image → Output: "Healthy" (Passed)
- **TC-02**: Upload diseased leaf image → Output: "Late Blight", 97% confidence (Passed)
- **TC-03**: Get treatment for disease → Correct remedies shown (Passed)
- **TC-04**: Invalid image upload → Error or "Unrecognized" (Passed)

- **TC-05**: Weather alert generation → Alert shown on high humidity (Passed)
- **TC-06**: User registration → Profile created (Passed)
- **TC-07**: API error simulation → "Service Unavailable" message (Passed)
- **TC-08**: App offline behavior → "Offline mode active" (Passed)

Conclusion:

The implementation of *Kisan Mitra* integrates advanced machine learning, modular development practices, and a farmer-friendly UI to solve real-world agricultural challenges. Comprehensive testing across functionality, performance, usability, and model evaluation ensures system robustness and field readiness.

5 Results and Discussions

5.1 Brief Description of Various Modules of the System

1. **Image Upload Module**
    - Allows the user to capture or upload a plant leaf image.
    - Accepts images in JPG/PNG format.
    - Triggers disease classification on submission.
2. **Disease Detection Module**
    - The uploaded image is analyzed using a lightweight CNN model.
    - Results show the disease name and confidence level (e.g., "Early Blight – 97% confidence").
    - If the image is not clear, appropriate warnings are displayed.
3. **Recommendation Module**
    - Fetches tailored remedies based on the diagnosed disease.
    - Recommendations include:
        - Chemical treatments (dosage, application method)
        - Organic alternatives
        - Preventive measures (irrigation, spacing, spraying frequency)
4. **Alert System Module**
    - Uses location data to fetch real-time weather.
    - Evaluates disease risk based on humidity, rainfall, and temperature.
    - Push notifications and in-app alerts are shown if disease-prone conditions are detected.
5. **User Profile and History Module**
    - Users can register and save their data (name, phone, crops grown).
    - Displays history of past uploads, predictions, and alerts received.

5.2 Snapshots of System

The *Kisan Mitra* mobile application features a thoughtfully designed interface aimed at enhancing the digital experience for farmers. Each screen is developed with usability, clarity, and purpose in mind.

5.2.1 Welcome Screen

- Acts as the splash screen when the app is first opened.
- Displays app logo and tagline.

5.2.2 Navigation and Access Screens

- **Navigation Bar (Top and Bottom)**:
  - Top Bar: Displays page title and quick access options.
  - Bottom Bar: Standard navigation tabs (Home, Crops, Assistant, News, Settings).
- **Language Selection Screen**:
  - Allows users to choose their preferred language (e.g., English, Hindi, Punjabi).
- **Login/Signup Screen**:
  - Mobile number-based authentication with minimal layout.

5.2.3 Core Application Screens

- **Home Screen**:
  - Weather updates, crop status, image diagnosis shortcut, news, and crop guide links.
- **Settings Screen**:
  - Profile management, FAQs, support contact, privacy policy, and language change options.

5.2.4 Educational and Support Modules

- **Crop Guide Page**:
  - Information on common crops and best practices.
- **Specific Crop Guide Screen**:
  - Detailed growth stages, diseases, fertilization, and irrigation advice.
- **Digital Assistant Welcome Screen**:
  - Entry point for the chat-based assistant.
- **Chat Interface**:
  - Real-time Q&A between user and assistant.
- **Chat History Screen**:
  - Review past interactions with the assistant.

5.2.5 Diagnosis and Alerts

- **Image Detection Results Screen**:
  - Displays disease classification, confidence, and remedy suggestions.
- **Your Crops Selection Screen**:
  - Manage and track the health status of chosen crops.
- **Weather Alerts Screen**:
  - Real-time disease outbreak risk based on current weather.

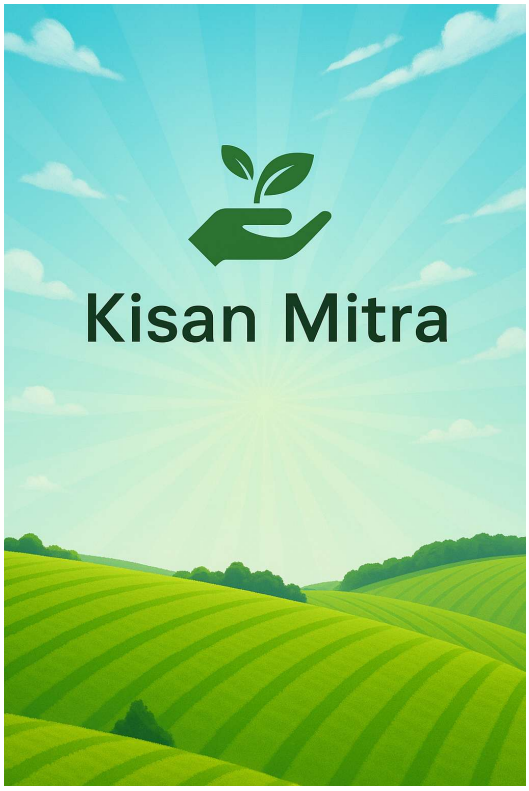
5.2.6 Informational and Utility Pages

- **News Screen**:
  - Agricultural news, categorized by topics like policy, technology, weather.
- **Other Screens**:

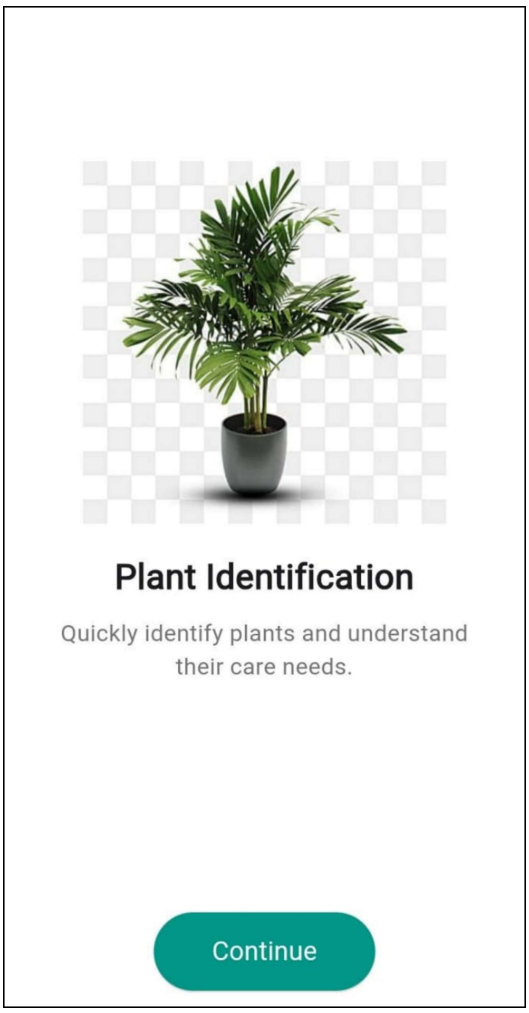
- Profile Page
- FAQ Page
- Terms & Conditions
- Contact Us Page

5.3 Mobile App UI

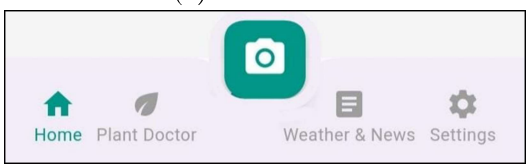
The frontend, built using Flutter, is fully designed to provide a seamless user experience.



(a) UI Screen 1

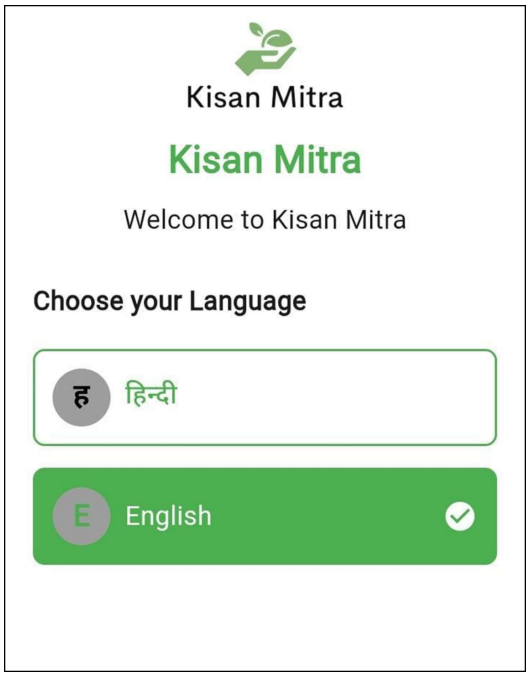


(b) UI Screen 2

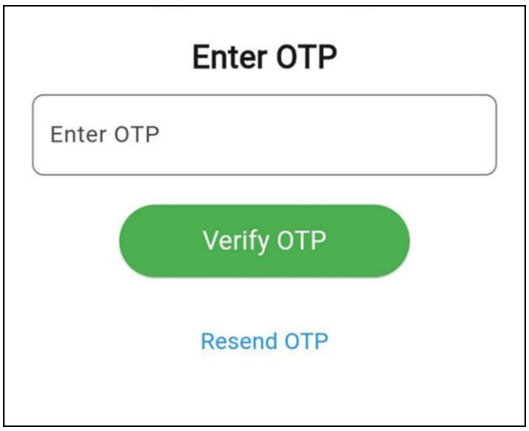


(c) UI Screen 3

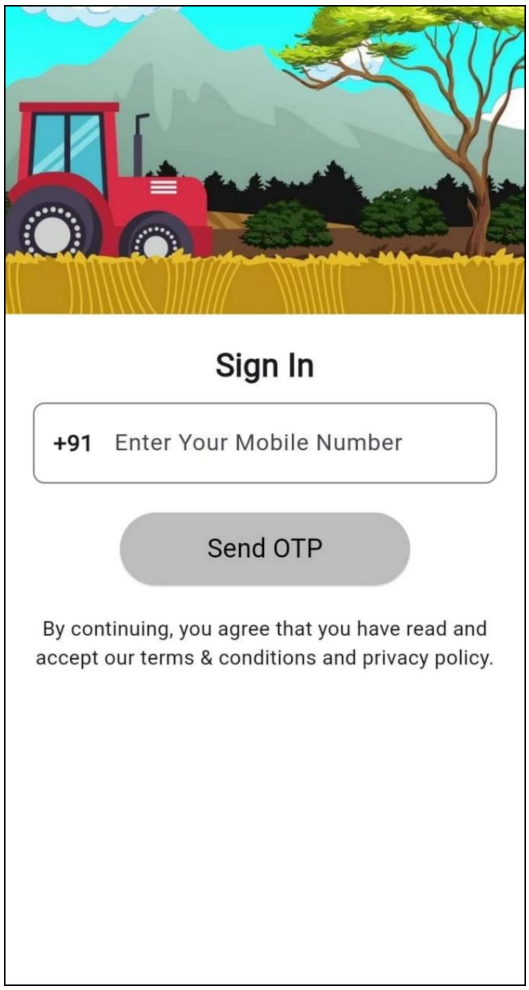
(d) UI Screen 4



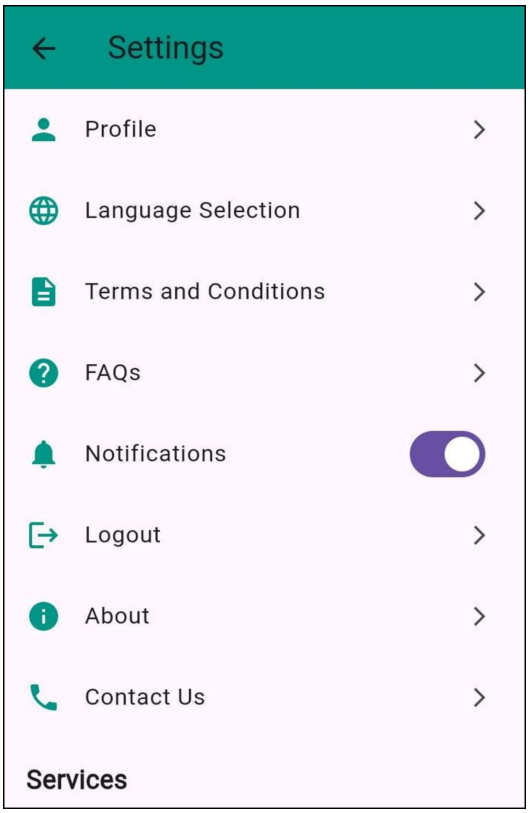
(a) UI Screen 5



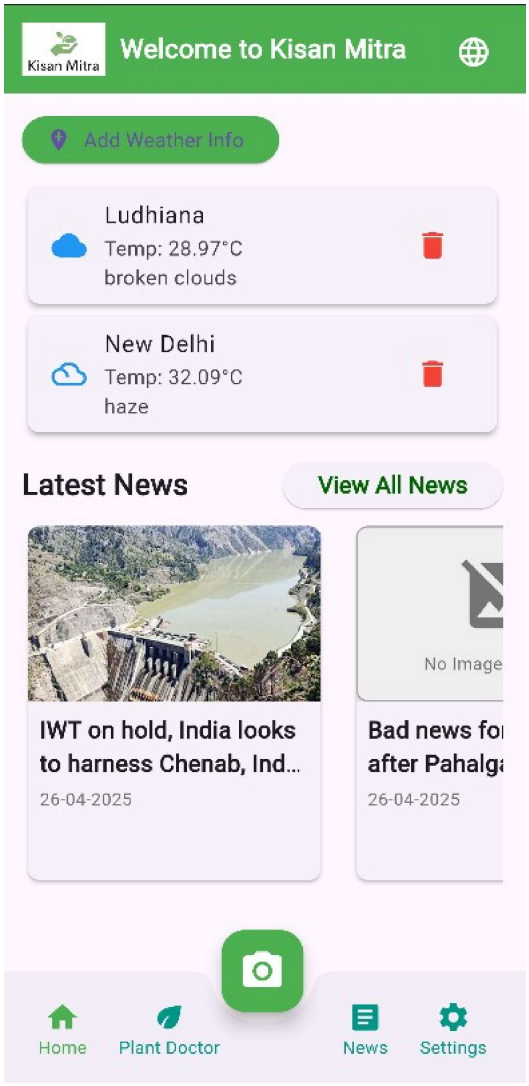
(b) UI Screen 6



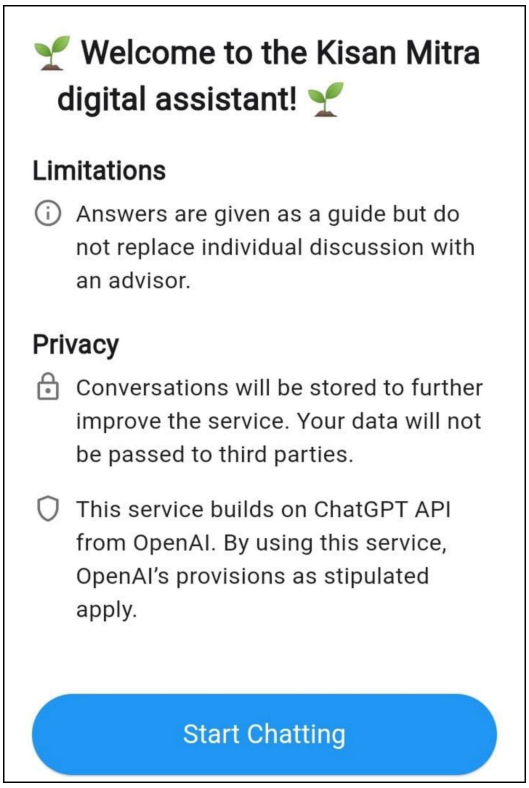
(c) UI Screen 7



(d) UI Screen 8



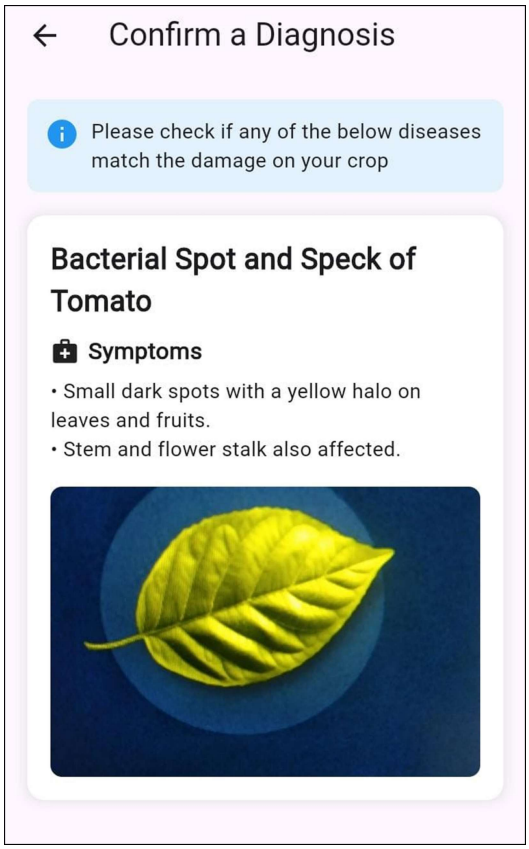
(a) UI Screen 9



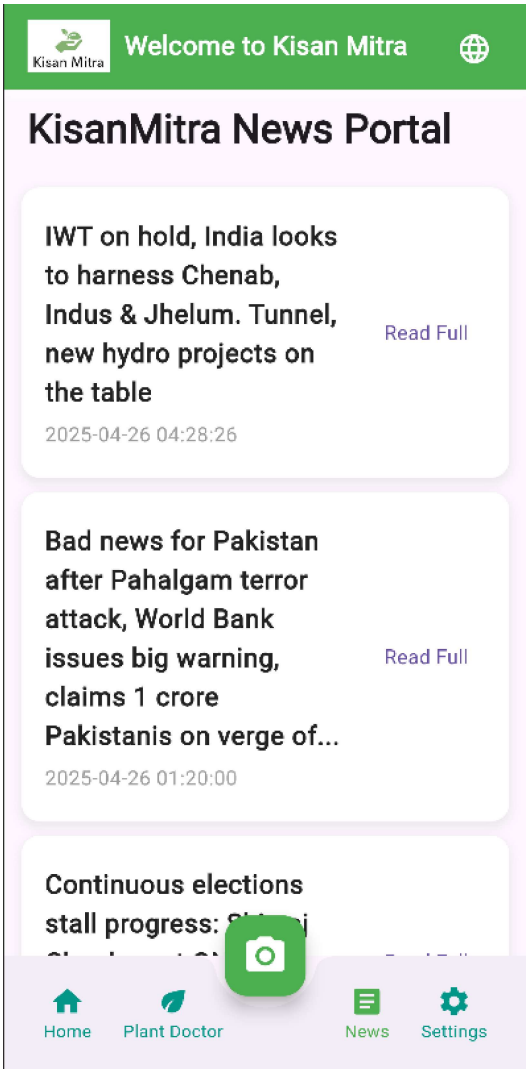
(b) UI Screen 10

(c) UI Screen 11

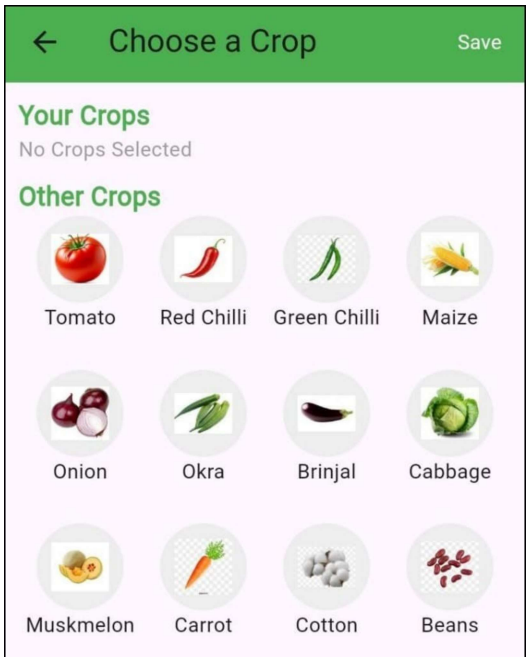
(d) UI Screen 12



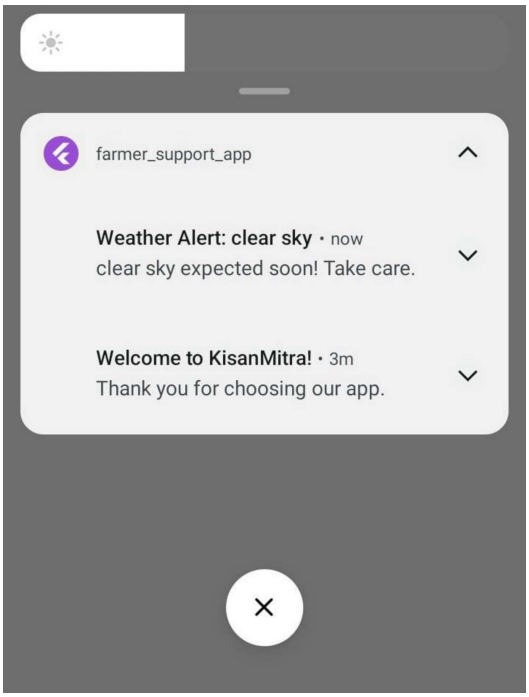
(a) UI Screen 13



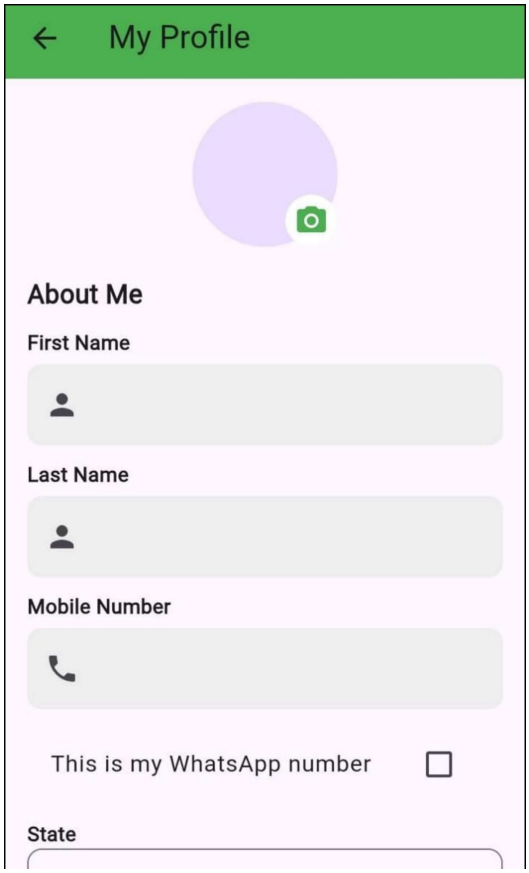
(b) UI Screen 14



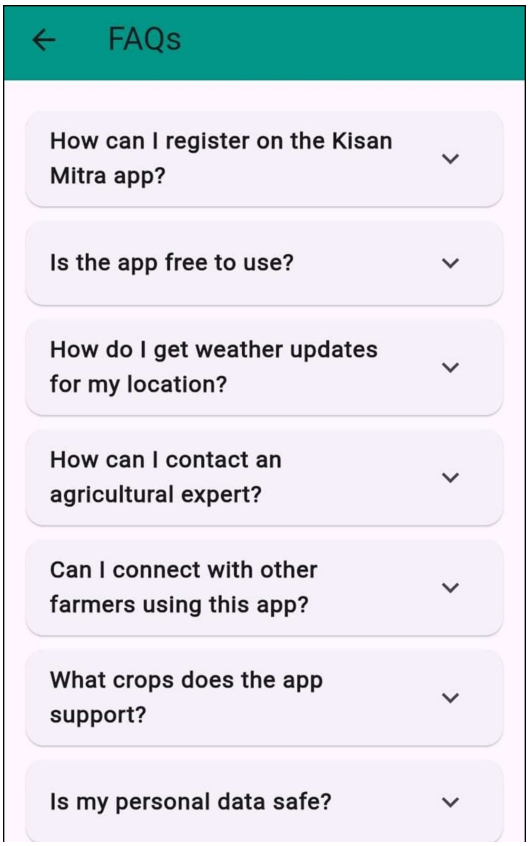
(c) UI Screen 15



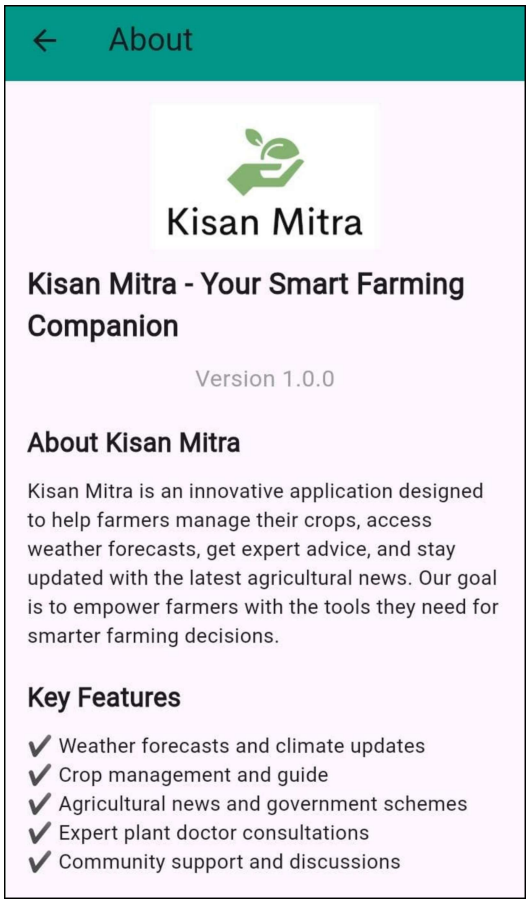
(d) UI Screen 16



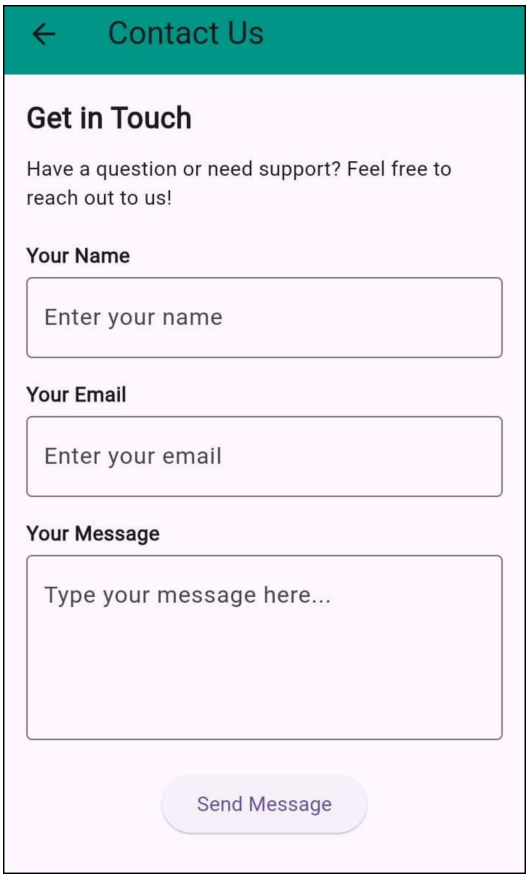
(a) UI Screen 17



(b) UI Screen 18



(c) UI Screen 19



(d) UI Screen 20

Figure 9: App User Interface Screens

5.4 Model Performance Visualizations Results

We tested the model on two unseen leaf images' test sets. The results indicate robust performance with high accuracy and minimal misclassifications across classes, as shown in the following visualizations.

5.4.1 Test Set - 1

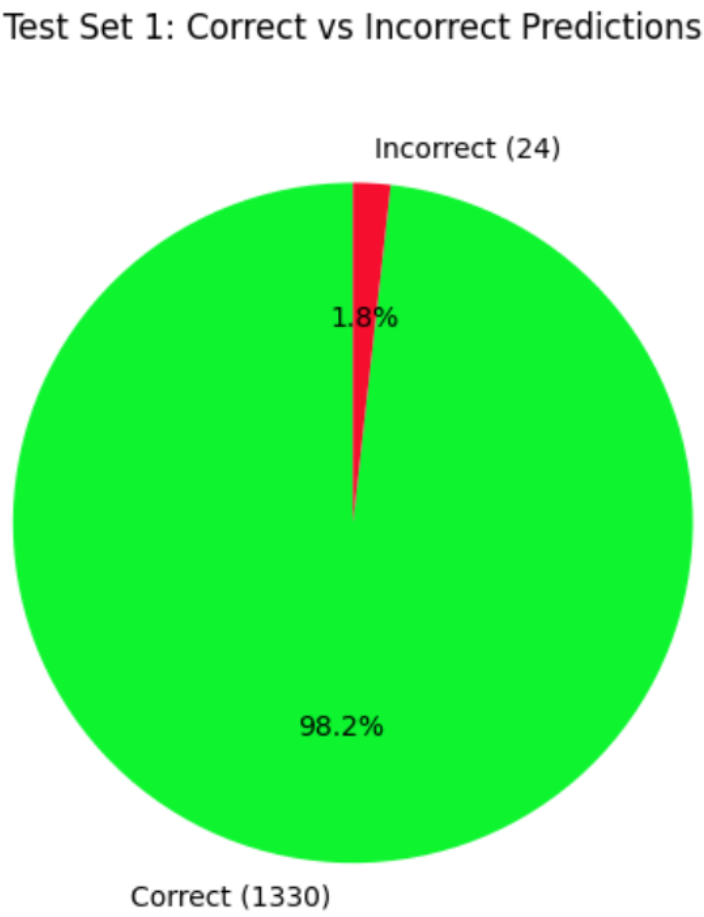


Figure 11: Test Set - 1: Correct vs Incorrect Predictions

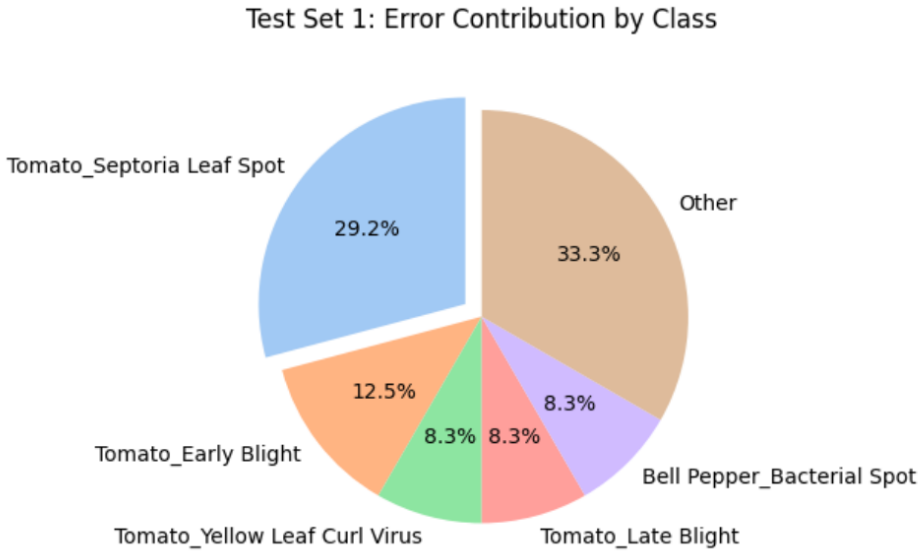


Figure 12: Test Set - 1: Error Contribution By Class

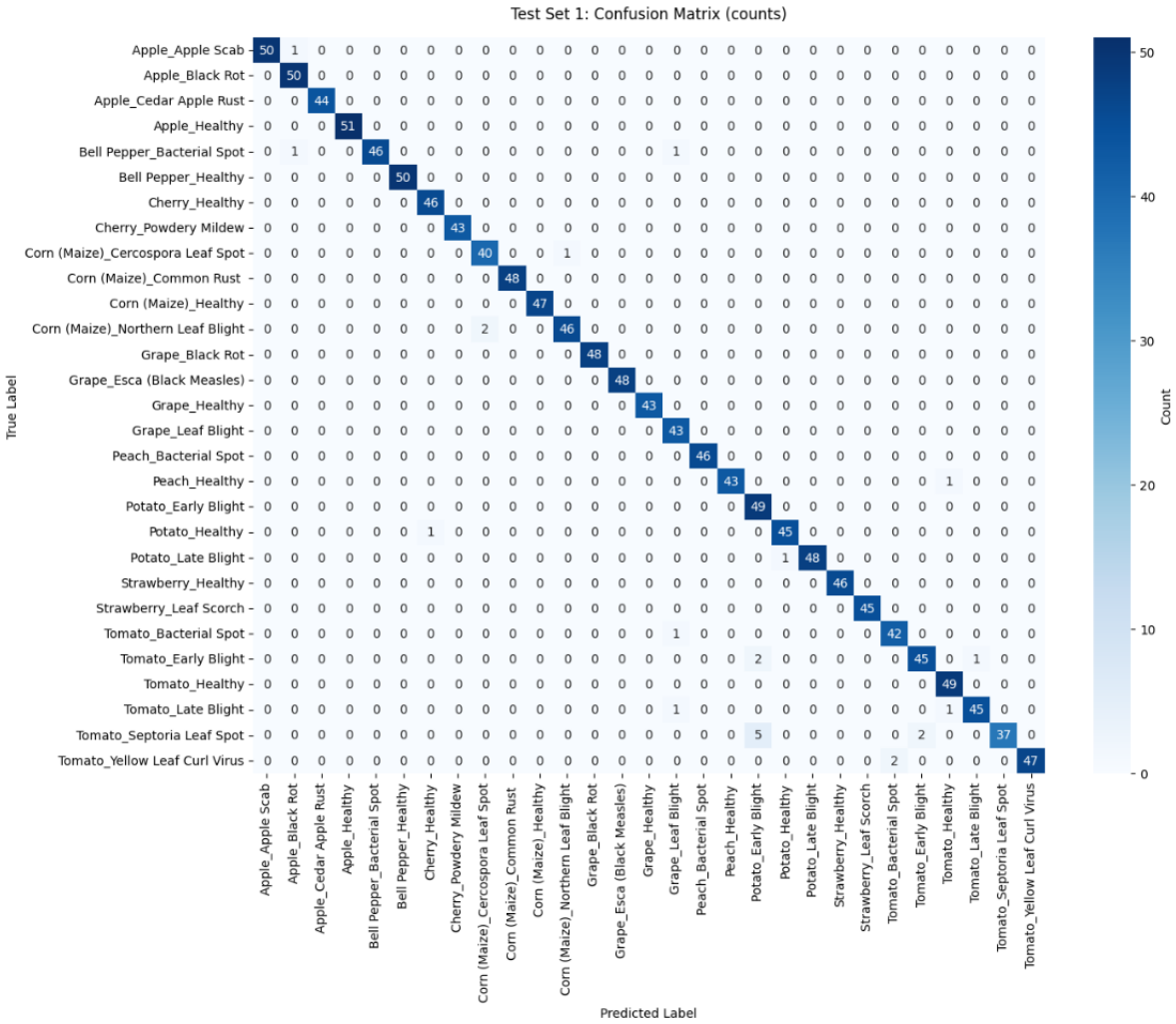


Figure 13: Test Set - 1: Confusion Matrix

Test Set 1: Classification Report

	precision	recall	f1-score	support
Apple_Apple Scab	1.00	0.98	0.99	51
Apple_Black Rot	0.96	1.00	0.98	50
Apple_Cedar Apple Rust	1.00	1.00	1.00	44
Apple_Healthy	1.00	1.00	1.00	51
Bell Pepper_Bacterial Spot	1.00	0.96	0.98	48
Bell Pepper_Healthy	1.00	1.00	1.00	50
Cherry_Healthy	0.98	1.00	0.99	46
Cherry_Powdery Mildew	1.00	1.00	1.00	43
Corn (Maize)_Cercospora Leaf Spot	0.95	0.98	0.96	41
Corn (Maize)_Common Rust	1.00	1.00	1.00	48
Corn (Maize)_Healthy	1.00	1.00	1.00	47
Corn (Maize)_Northern Leaf Blight	0.98	0.96	0.97	48
Grape_Black Rot	1.00	1.00	1.00	48
Grape_Esca (Black Measles)	1.00	1.00	1.00	48
Grape_Healthy	1.00	1.00	1.00	43
Grape_Leaf Blight	0.93	1.00	0.97	43
Peach_Bacterial Spot	1.00	1.00	1.00	46
Peach_Healthy	1.00	0.98	0.99	44
Potato_Early Blight	0.88	1.00	0.93	49
Potato_Healthy	0.98	0.98	0.98	46
Potato_Late Blight	1.00	0.98	0.99	49
Strawberry_Healthy	1.00	1.00	1.00	46
Strawberry_Leaf Scorch	1.00	1.00	1.00	45
Tomato_Bacterial Spot	0.95	0.98	0.97	43
Tomato_Early Blight	0.96	0.94	0.95	48
Tomato_Healthy	0.96	1.00	0.98	49
Tomato_Late Blight	0.98	0.96	0.97	47
Tomato_Septoria Leaf Spot	1.00	0.84	0.91	44
Tomato_Yellow Leaf Curl Virus	1.00	0.96	0.98	49
accuracy			0.98	1354
macro avg	0.98	0.98	0.98	1354
weighted avg	0.98	0.98	0.98	1354

Figure 14: Test Set - 1: Classification Report

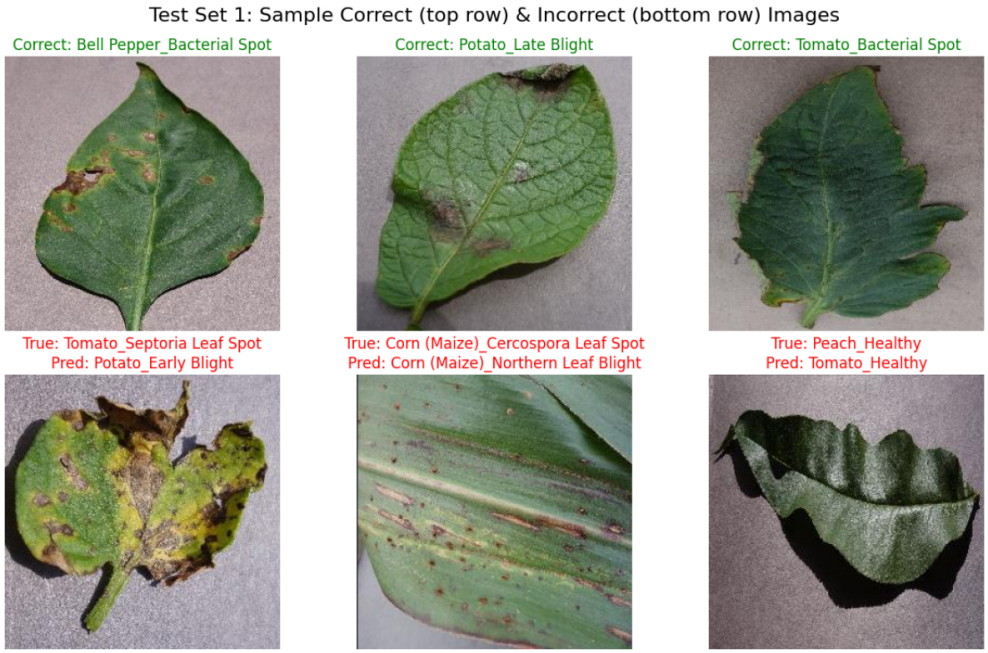


Figure 15: Test Set - 1: Correct and Incorrect Samples

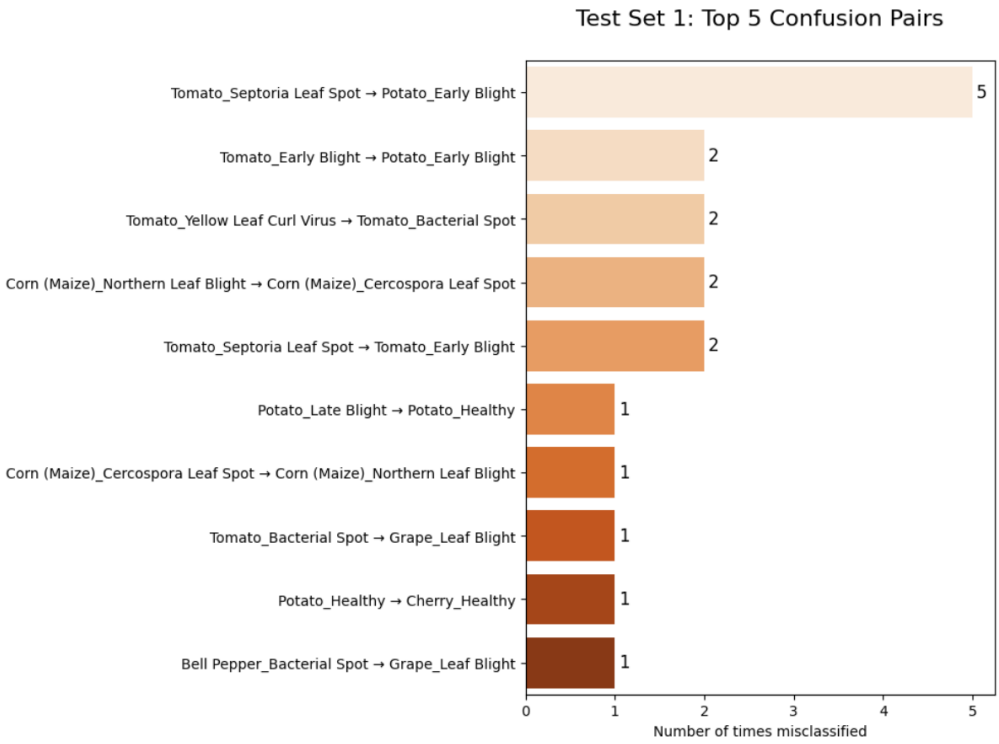


Figure 16: Test Set - 1: Top 5 Confusion Pairs

5.4.2 Test Set - 2

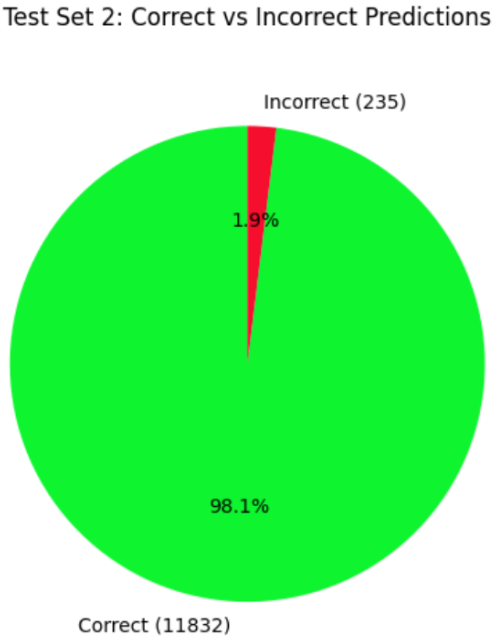


Figure 17: Test Set - 2: Correct vs Incorrect Predictions

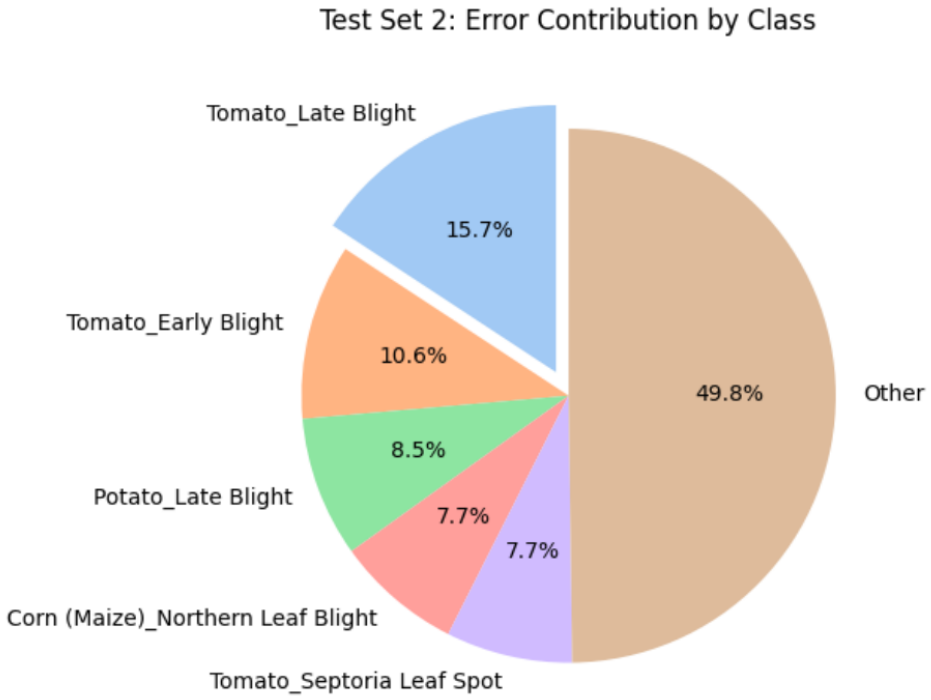


Figure 18: Test Set - 2: Error Contribution By Class

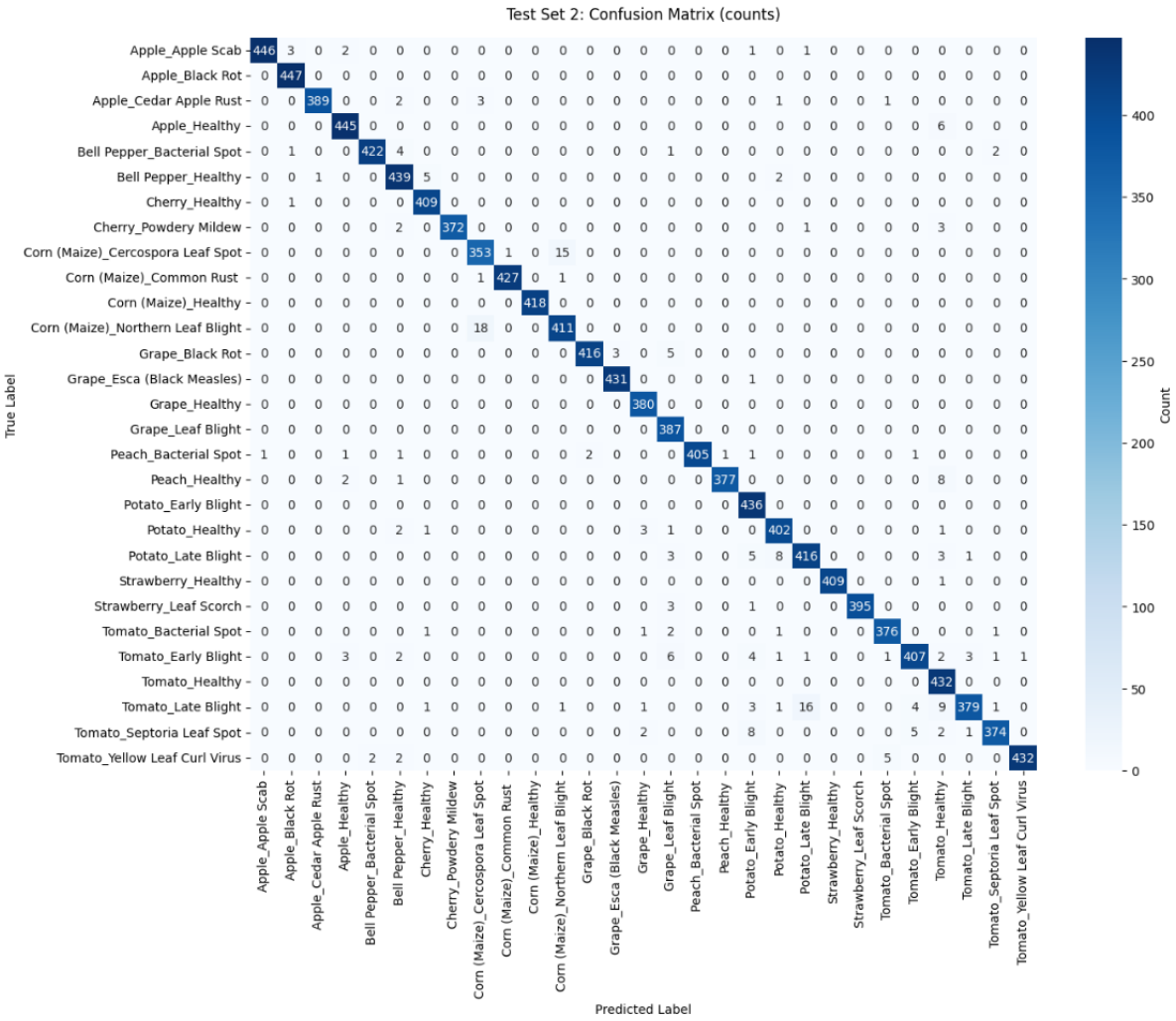


Figure 19: Test Set - 2: Confusion Matrix

Test Set 2: Classification Report

	precision	recall	f1-score	support
Apple_Apple Scab	1.00	0.98	0.99	453
Apple_Black Rot	0.99	1.00	0.99	447
Apple_Cedar Apple Rust	1.00	0.98	0.99	396
Apple_Healthy	0.98	0.99	0.98	451
Bell Pepper_Bacterial Spot	1.00	0.98	0.99	430
Bell Pepper_Healthy	0.96	0.98	0.97	447
Cherry_Healthy	0.98	1.00	0.99	410
Cherry_Powdery Mildew	1.00	0.98	0.99	378
Corn (Maize)_Cercospora Leaf Spot	0.94	0.96	0.95	369
Corn (Maize)_Common Rust	1.00	1.00	1.00	429
Corn (Maize)_Healthy	1.00	1.00	1.00	418
Corn (Maize)_Northern Leaf Blight	0.96	0.96	0.96	429
Grape_Black Rot	1.00	0.98	0.99	424
Grape_Esca (Black Measles)	0.99	1.00	1.00	432
Grape_Healthy	0.98	1.00	0.99	380
Grape_Leaf Blight	0.95	1.00	0.97	387
Peach_Bacterial Spot	1.00	0.98	0.99	413
Peach_Healthy	1.00	0.97	0.98	388
Potato_Early Blight	0.95	1.00	0.97	436
Potato_Healthy	0.97	0.98	0.97	410
Potato_Late Blight	0.96	0.95	0.96	436
Strawberry_Healthy	1.00	1.00	1.00	410
Strawberry_Leaf Scorch	1.00	0.99	0.99	399
Tomato_Bacterial Spot	0.98	0.98	0.98	382
Tomato_Early Blight	0.98	0.94	0.96	432
Tomato_Healthy	0.93	1.00	0.96	432
Tomato_Late Blight	0.99	0.91	0.95	416
Tomato_Septoria Leaf Spot	0.99	0.95	0.97	392
Tomato_Yellow Leaf Curl Virus	1.00	0.98	0.99	441
accuracy			0.98	12067
macro avg	0.98	0.98	0.98	12067
weighted avg	0.98	0.98	0.98	12067

Figure 20: Test Set - 2: Classification Report

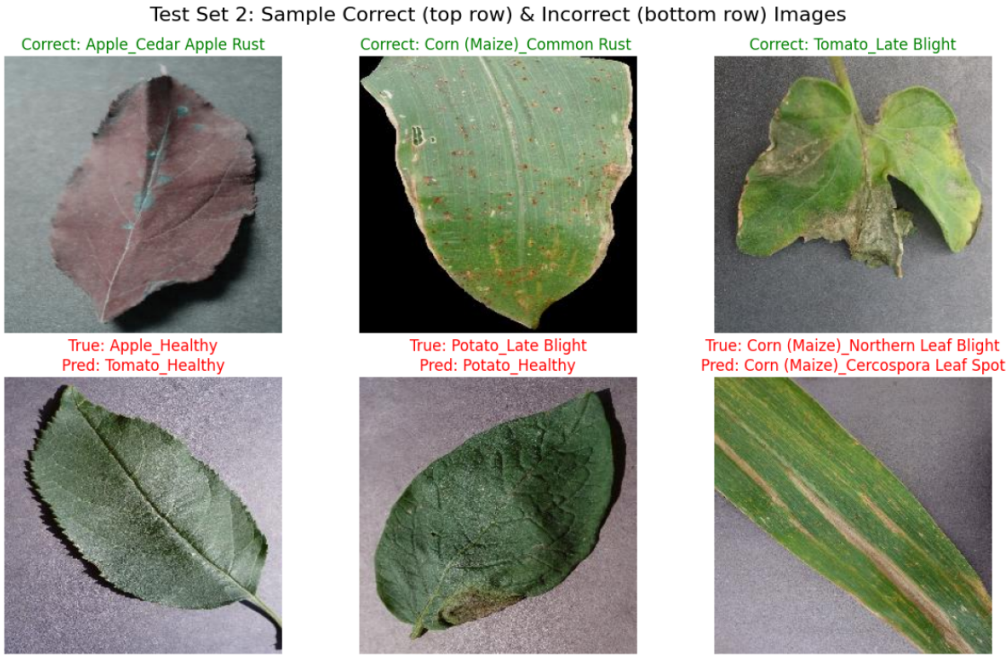


Figure 21: Test Set - 2: Correct and Incorrect Samples

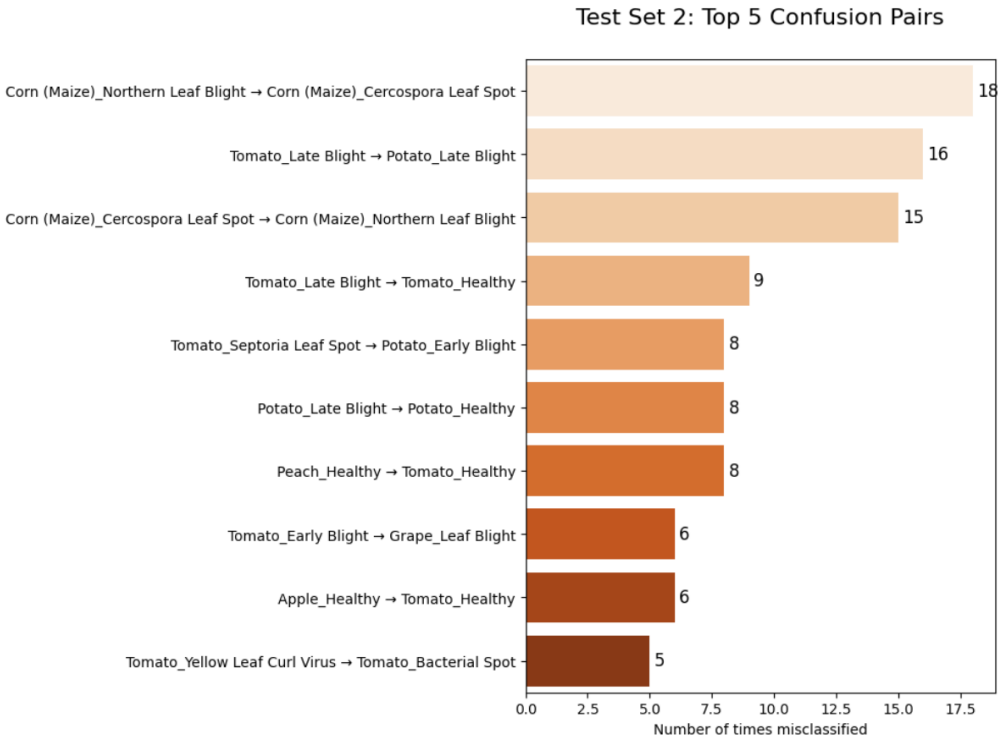


Figure 22: Test Set - 2: Top 5 Confusion Pairs

5.5 Recommendations Result

Figure 23: Sample Recommendations Screen

6 Conclusion and Future Scope

6.1 Conclusion

Kisan Mitra is an innovative mobile-based smart farming assistant developed to empower farmers through the integration of machine learning, mobile technology, and environmental intelligence. The system successfully addresses three primary agricultural challenges: early disease detection, targeted treatment recommendations, and weather-based disease risk alerts.

Through the use of teacher-student model distillation, an optimal trade-off between high classification accuracy and efficient deployment on low-resource mobile devices has been achieved. The intuitive Flutter-based mobile interface makes the application accessible to users with minimal digital literacy, while the backend infrastructure ensures fast, secure, and scalable operations.

The application enhances the decision-making ability of farmers by providing real-time, personalized, and data-driven agricultural assistance, ultimately contributing to reduced crop loss, improved productivity, and sustainable farming practices.

6.2 Future Scope

While the current system lays a strong foundation, there are several areas where *Kisan Mitra* can be expanded and enhanced in future iterations:

- **Multi-language and Voice Support**:
  - Introduce support for regional languages and voice commands for better accessibility.
- **Soil Health and Crop Advisory**:
  - Extend ML integration for soil type detection and crop suitability suggestions.
- **Offline Mode Enhancements**:
  - Implement complete offline functionality with on-device inference and delayed sync.
- **AI Chatbot Integration**:
  - Provide conversational responses for farmers to ask agricultural queries directly.
- **Integration with Government Schemes and Pricing APIs**:
  - Notify users about subsidies, mandi prices, and crop insurance programs.
- **Crowdsourced Disease Reporting**:
  - Enable users to report disease outbreaks in their area and share geotagged data anonymously.
- **Drone or IoT Integration**:
  - Incorporate drone imagery or IoT soil sensors for precision agriculture support.

By continuously evolving, *Kisan Mitra* has the potential to become a complete digital assistant for farmers, driving forward the vision of smart and sustainable agriculture in India and beyond.

6.3 References/Bibliography

References

[1] A. Tan and Q. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," in *Proceedings of the 36th International Conference on Machine Learning (ICML)*, Long Beach, California, 2019. [Online]. Available: `https://arxiv.org/abs/1905.11946`

[2] A. Howard *et al.*, "Searching for MobileNetV3," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 1314–1324. [Online]. Available: `https://ieeexplore.ieee.org/document/9008835`

[3] OpenWeatherMap API, "Weather API Documentation," OpenWeather Ltd. [Online]. Available: `https://openweathermap.org/api`

[4] T. Garudadri *et al.*, "Knowledge Distillation Facilitates Lightweight and Efficient Plant Disease Detection Models," *Plant Phenomics*, vol. 2023, Article ID 0062, 2023. [Online]. Available: `https://spj.science.org/doi/10.34133/plantphenomics.0062`

[5] TensorFlow, "TensorFlow Lite — ML for Mobile and Edge Devices," Google AI. [Online]. Available: `https://www.tensorflow.org/lite`

[6] Flutter, "Build apps for any screen," Google Developers. [Online]. Available: `https://flutter.dev/`

[7] Node.js Documentation, "Node.js v20.0.0 Documentation," Node.js Foundation. [Online]. Available: `https://nodejs.org/en/docs`

[8] T. Sharma, "Plant Village Dataset Updated," Kaggle Datasets. [Online]. Available: `https://www.kaggle.com/datasets/tushar5harma/plant-village-dataset-updated`