FALL ARMYWORM DIAGNOSIS

By

BSE 25-22

DEPARTMENT OF NETWORKS

SCHOOL OF COMPUTING AND INFORMATICS TECHNOLOGY

A Project Report Submitted to the School of Computing and Informatics Technology for the Study Leading to a Project in Partial Fulfillment of the Requirements for the Award of the Degree of Bachelor of Science in Software Engineering of Makerere University.

Supervisor

Dr. NTANDA MOSES

Department of Networks

School of Computing and Informatics Technology, Makerere University

mntanda@cit.ac.ug, +256-41-540628, Fax: +256-41-540620

May, 2025.

DECLARATION

We, group BSE 25-22, hereby declare that the work presented is original and has never been submitted for an award to any university or institution of higher learning. We can confirm that where we have done consultations either from published material or the works of others, it has been attributed in this report.

#	Names	Registration Number	Signature
	SSEBATTA ADAM	21/U/0534	
2	BUTERABA LYNATTE JOANITTA	21/U/09727/PS	
3	NABUKENYA MARIAM	21/U/17130/EVE	
4	GAFABUSA WILLY	21/U/13385/EVE	

APPROVAL

This project report titled Fall Armyworm Diagnosis has been submitted for examination with my approval as the supervisor of group BSE25-22

Dr. Ntanda Moses

Department of Networks

School of Computing and Informatics Technology;

College of Computing and Information Sciences,

Makerere University

Signature: ..... Date: .....

Supervisor

DEDICATION

This project is dedicated to all those who have supported, encouraged, and inspired us throughout the development of the Fall Armyworm Diagnosis System.

To our supervisor, Dr. Ntanda Moses, we express our deep gratitude for his continuous guidance, patience, and support. His insight and commitment helped us grow, stay focused, and push through challenges, and his mentorship has been instrumental to our success.

We also dedicate this work to our general supervisor, Dr. Mary Nsabagwa, whose wisdom and direction gave us clarity and purpose at every stage of this project. Her thoughtful feedback and encouragement helped shape our system and reminded us to aim higher.

This project is dedicated to the farmers of Uganda, whose daily efforts to grow and feed communities inspired us to create a tool that can make a real difference. We hope this system becomes a step toward more accessible and effective agricultural solutions.

Finally, we dedicate this work to our families and friends, who stood by us during late nights, tough decisions, and moments of doubt. Your love, support, and belief in us gave us the strength to keep going.

ACKNOWLEDGEMENT

We are deeply grateful to everyone who contributed to the successful completion of our project, the Fall Armyworm Detection. We would like to express our heartfelt thanks to all who supported us throughout this journey.

First, we thank **God** for granting us life, strength, and wisdom. His grace guided us through every step of our academic journey and gave us the determination to reach this point.

We extend our sincere appreciation to our supervisor, **Dr. Ntanda Moses**, for his dedicated mentorship, consistent guidance, and encouragement. His expertise and thoughtful feedback were essential in shaping our ideas and refining our system. We are thankful for the time and attention he gave us, even with his busy schedule.

We also extend our deep appreciation to our general supervisor, **Dr. Mary Nsabagwa**, who played a pivotal role in shaping our project. Her guidance gave us the clarity we needed at each stage, and her thoughtful challenges encouraged us to improve our approach. She consistently directed us toward the right path, and her involvement greatly enriched the development of this system.

We also wish to thank the entire team at the **School of Computing and Informatics Technology** for the knowledge and skills we have gained during our studies, which laid the foundation for this project.

To our **parents**, we are truly thankful for your sacrifices, love, and prayers. Your support both emotional and financial gave us the peace of mind to focus on our studies. You believed in us and cheered us on, even when things got difficult.

To our **friends and classmates**, thank you for your encouragement, honest feedback, and for walking this journey with us. Your presence made the long hours of work lighter and the successes more joyful.

Finally, we are grateful to the **farmers and agricultural experts** who inspired this project with their stories and challenges. Their real-world experiences motivated us to build a system that could make a practical difference.

Without all this support, the Fall Armyworm Detection System would not have come to life. We are truly grateful.

ABSTRACT

The Fall Armyworm Diagnosis System is a mobile-first solution developed to support Ugandan farmers and agricultural officers in identifying, monitoring, and managing Fall Armyworm (FAW) infestations. The system leverages artificial intelligence, geolocation, weather integration, and expert-driven community support to address the persistent challenges of late-stage pest detection, improper pesticide use, and lack of localized outbreak data. It is optimized for rural environments where internet connectivity and technical literacy may be limited.

The platform features a machine learning model based on TensorFlow Lite and MobileNet architecture, capable of analyzing maize leaf images and classifying them into four stages: Healthy, Eggs, Frass, or Larval Damage. A pre-screening module ensures that only valid maize leaf images are processed. Stage-specific treatment recommendations including herbicide names, dosages, and safe application instructions are generated automatically. The system also integrates GPS-based geolocation and reverse geocoding to map detections to districts, enabling the visualization of outbreak patterns through color-coded maps and supporting regional pest surveillance.

In addition to detection, the system provides real-time weather data using the OpenWeatherMap API to help users optimize pesticide application timing. Users can view historical detections, generate downloadable PDF reports, and access a dynamic analytics dashboard that visualizes infestation trends, detection stage distributions, and regional outbreak comparisons. A community support module enables authenticated users to post crop-related concerns, attach images, and receive advice from verified agricultural experts.

This report documents the entire system lifecycle from the definition of system requirements and specifications to architectural design, backend development using Flask and SQL, and mobile interface implementation using Flutter. It also covers inspection, testing, and validation procedures across multiple modules, including input processing, detection logic, map rendering, and performance under rural network conditions.

The outcomes demonstrate that the system offers an effective, user-friendly, and intelligent platform for pest detection and management. By combining machine learning, geospatial data, weather forecasts, and expert knowledge, the Fall Armyworm Diagnosis System enhances farmers' ability to respond to infestations early, apply treatment correctly, and contribute valuable data for national agricultural planning. Its scalable and modular design also offers potential for adaptation to other crop pests and regions.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS/ACRONYM

SA	Ssebatta Adam
GW	Gafabusa Willy
BL	Buteraba Lynatte
NM	Nabukenya Mariam
FAW	Fall Armyworm
APIs	Application Programming Interfaces
ML	Machine Learning
GPS	Global Positioning System
TF	TensorFlow
HTTPS	HyperText Transfer Protocol Secure
FastAPI	Fast Application Programming Interface (a modern web framework for building APIs with Python)
CNN	Convolutional Neural Network
APK	Android Package Kit
VS Code	Visual Studio Code
MySQL	My Structured Query Language
CI/CD	Continuous Integration / Continuous Deployment
e.g.	For example (*exempli gratia*)
SDK	Software Development Kit
App	Application
CORS	Cross-Origin Resource Sharing
ID	Identifier
JSON	JavaScript Object Notation
SDD	Software Design Document
SQL	Structured Query Language
HTTP	HyperText Transfer Protocol
v2	Version 2
GeoJSON	Geographic JavaScript Object Notation
.pdf	Portable Document Format
HEX	Hexadecimal
EAT	East Africa Time
MB	Megabyte

CHAPTER 1: INTRODUCTION

1.1 Background

Fall Armyworm (*Spodoptera frugiperda*) is a highly destructive pest that poses a significant threat to maize production in Uganda and across Sub-Saharan Africa[1]. Since its emergence in the region, the pest has caused severe yield losses, threatening food security and the livelihoods of millions of smallholder farmers. One of the major challenges in managing Fall Armyworm is the lack of timely and accurate detection at various stages of infestation, leading to delayed responses and inappropriate pesticide application.

This project presents the design and implementation of a Fall Armyworm Disease Detection**,** a comprehensive mobile application aimed at empowering farmers and agricultural authorities in Uganda to detect, monitor, and manage Fall Armyworm infestations effectively. The system leverages modern technologies including machine learning, geospatial analysis, and mobile computing to provide accurate stage-wise detection, district-level disease mapping, treatment recommendations, and expert support.

1.2 Scope of the Project

The scope of the project covers the complete life cycle of infestation detection, from image capture and classification of the pest stage (eggs, frass, larval damage), to location tracking, data visualization, analytics, weather integration, and community interaction. The system includes a mobile-based user interface backed by a Flask-based REST API, with Firebase and it is targeted towards smallholder maize farmers, agricultural extension officers, and government authorities across Uganda.

The system includes a mobile application for field use by farmers and experts, and a Flask-powered backend API integrated with modern technologies such as:

- **MobileNet-based TensorFlow Lite models** for real-time image classification
- **SQL database** for storing detection and user data
- **Geolocation APIs** for district-level outbreak mapping
- **OpenWeatherMap API** for weather-informed spraying advice
- **Firebase** for user authentication and cloud-based storage
- **Community-driven expert support** for knowledge sharing

1.3 Project Objectives

The main objectives of the project are:

- To provide a mobile-based platform for farmers to diagnose Fall Armyworm infestation on maize leaves through automated image analysis.
- To ensure detections are reliable by verifying input images and linking results with relevant treatment advice.
- To enable geolocation-based tracking of infestation patterns across districts for improved pest surveillance.
- To support farmers with weather-informed guidance to improve the timing and effectiveness of pesticide application.
- To offer a communication channel where farmers can share problems and receive advice from agricultural experts and peers.
- To allow users to monitor historical detections and access regional trends through interactive reports and visual dashboards.
- To assist agricultural authorities and researchers in collecting real-time field data for planning interventions and policymaking.

1.4 Target Users

The system is primarily designed for:

- **Smallholder maize farmers** in rural and semi-urban Uganda
- **Agricultural extension workers** who assist and train farmers in pest management
- **District agricultural officers** for planning and policy response
- **Researchers and NGOs** focused on pest control and food security

1.5 Expected Benefits

The system provides several benefits to end users and the broader agricultural ecosystem:

- **Timely and accurate pest detection**, enabling early intervention and reducing yield losses.
- **Improved pesticide application practices** by recommending treatments based on infestation stage and weather conditions.
- **Empowered decision-making** through visual dashboards, historical trend analysis, and data-driven insights.

- **Stronger community support** and expert engagement via the interactive knowledge-sharing platform.
- **Digitized reporting and traceability**, allowing farmers and stakeholders to document infestations and evaluate progress.
- **Scalability and reusability**, allowing the system to be adapted for future crop pest and disease detection needs.

1.6 Overview of the document

This report provides a comprehensive account of the system development process, from problem identification to system deployment. It details the system specifications, software and hardware architecture, implementation strategies, testing and validation results, as well as challenges encountered and recommendations for future improvements.

The document is structured into the following chapters:

- **Chapter 1:** Introduces the problem background, scope of the system, and document structure.
- **Chapter 2:** Specifies the functional and non-functional requirements of the system, including detailed input/output specifications, functionality, limitations, and default settings.
- **Chapter 3:** Describes the design and implementation of the system, the architecture, technologies used, user interface, and programming practices.
- **Chapter 4:** Discusses the testing process, inspection results, and performance evaluation of the system.
- **Chapter 5:** Outlines the system installation process and user acceptance testing.
- **Chapter 6:** Covers performance expectations, servicing, maintenance strategies, and how future updates can be managed.
- **Chapter 7:** Summarizes the findings and proposes recommendations for future work and system enhancement.

CHAPTER 2: SYSTEM SPECIFICATIONS

This chapter presents the technical specifications of the Fall Armyworm Disease Detection System. It details the system's input and output mechanisms, functionality, limitations, safety considerations, version control processes, default settings, special operational requirements, and error-handling strategies. These specifications guided the development and evaluation of the system.

2.1 Version of requirements and Version Control

During the system's development lifecycle, the requirements evolved through stakeholder feedback, testing iterations, and continuous model evaluation. To manage these changes systematically and ensure traceability, the development team adopted Git for version control, hosted on GitHub to support collaborative development.

A semantic versioning strategy was used, where updates were recorded as minor or patch-level changes based on the scope of improvement. The system underwent several iterations, each introducing new features while preserving the overall system structure.

- **Version 1.0**: Initial prototype featuring basic image upload and classification using a TensorFlow Lite model. Detection was limited to gallery uploads, and there was no integration with location or analytics modules.
- **Version 1.1**: Introduced GPS-based geolocation tracking, enabling the mapping of detection results to specific districts in Uganda.
- **Version 1.2**: Added an analytics dashboard to display temporal trends and district-wise infestation data. This version also integrated weather forecasting to assist with treatment timing.
- **Version 1.3**: Implemented a community module that allows users to post crop issues and receive expert guidance. Detection history tracking and refined treatment recommendations were also included.

All version changes were logged in a changelog file, and distinct Git branches (e.g., main, geo-feature, analytics-enhancements) were used to separate and manage development across modules. This approach allowed the team to incrementally improve the system while maintaining a stable core.

2.2 Input

The system accepts several types of inputs from the user, device sensors, and third-party APIs. These inputs are essential for the detection, classification, and contextual analysis of Fall Armyworm infestation.

1. **Image (Maize Leaf):**
   - Users can upload maize leaf images using the mobile device's camera, select from the gallery, or scan in real time.
   - The system first performs pre-screening to confirm the presence of a maize leaf. Only valid images proceed to detection.
   - The image is then analyzed by the model to determine the infestation stage.
2. **Geolocation (GPS Coordinates):**
   - Automatically captured by the mobile device.
   - Used to identify the Ugandan district where the image was captured.
   - Coordinates are validated to ensure they fall within Uganda’s geographic boundaries.
3. **User Metadata:**
   - Captured data includes the user's identification number, date and time of detection, device ID, and district.
4. **Text Input for Community Support:**
   - Farmers can describe their crop problems and share related images or questions for expert assistance.
5. **Weather Data API:**
   - The system fetches local weather conditions (e.g., rainfall, temperature, humidity) to inform treatment timing. This provides a good timing for spraying conditions.

2.3 Output

The system generates several outputs that assist users in identifying infestations, understanding their severity, and applying appropriate interventions. Outputs vary depending on the detection stage and the presence or absence of infestation.

1. **Detection Results:**
    - The system returns the specific stage of Fall Armyworm infestation, which may include: *Healthy*, *Eggs*, *Frass*, or *Larval Damage*.
    - If the uploaded, captured, or scanned image is determined not to contain a maize leaf, the system terminates the analysis process and prompts the user to submit a valid maize leaf image.
    - Each stage corresponds to specific symptoms on the maize leaf. For example:
    - **Eggs** indicate early infestation and appear as tiny white/yellow clusters.
    - **Frass** refers to pest droppings, suggesting active larval feeding.
    - **Larval Damage** represents visible damage such as holes and shredding of leaves.
    - A confidence score is provided to indicate the model's certainty about the classification.
    - Recommendations are provided on how to control the disease on a given detected stage.
    - Downloading of the diagnosis Results.

2. **Treatment Recommendations:**
    - Based on the detected stage, the system provides herbicide suggestions and application guidelines.
    - Recommendations vary by stage, ranging from early-stage bio-pesticide applications to stronger synthetic chemical treatments for severe damage.
    - Herbicide instructions include names, mixing ratios, timing of application (e.g., early morning), and safety precautions.

3. **Geospatial Visualization:**
    - Detections are displayed on an interactive map.
    - The map includes colored markers to represent infestation severity and allows users to filter data by date, district, and infestation type.

4. **Analytical Reports:**
    - Users can view dynamic graphs showing:
    - Temporal trends (daily, weekly, monthly)
    - District-wise infestation intensity
    - Comparisons between time periods
    - Classification stage distribution

5. **Detection Report Generation:**

- After each detection, users have the option to generate a downloadable report.
- The report includes details such as detection stage, GPS location, timestamp, image preview, and recommended treatment.
- If the system detects no infestation (i.e., the image is classified as **Healthy**), a "clean report" is generated to confirm the absence of the Fall Armyworm.

6. **Community Responses:**

- When farmers share issues in the community section, expert feedback is returned as structured responses.
- This supports collaborative problem-solving and learning.

2.4 Functionality

The Fall Armyworm Disease Detection System is designed as a comprehensive and intelligent decision-support tool for maize farmers. It integrates several critical functionalities aimed at early detection, accurate diagnosis, actionable treatment recommendations, and effective tracking of Fall Armyworm infestations. Each function plays a unique role in ensuring that farmers can manage the pest problem effectively, improve their crop health, and reduce crop loss.

Below is a detailed explanation of each core system functionality:

2.4.1 Infestation Detection

This is the core functionality of the system, responsible for identifying whether a submitted maize leaf image shows signs of Fall Armyworm infestation, and if so, classifying the **stage of the infestation**.

- The detection is powered by a **TensorFlow Lite deep learning model** deployed in the backend, optimized for mobile usage. The model is trained on thousands of annotated maize leaf images across various infestation stages.
- The model first uses a **binary classifier** to detect whether the image contains a maize leaf or not (pre-screening stage).
- If confirmed, the image is passed to a **multi-class classifier** based on the **MobileNet architecture**. This classifier is trained to distinguish between:

- *Healthy*
- *Eggs*
- *Frass*
- *Larval Damage*
- The model outputs a prediction label along with a **confidence score** that indicates the level of certainty of the classification.
- This detection result is used to trigger stage-specific actions such as treatment advice, alert generation, and report generation.

This functionality ensures **early detection** of infestations, which is critical for timely intervention and reduction of crop losses.

2.4.2 Pre-Screening Validation

Before any image is submitted for classification, the system first validates whether the image actually contains a maize leaf. This pre-screening process is essential to ensure the accuracy and relevance of the classification results.

- The pre-screening classifier is a lightweight convolutional neural network (CNN) model trained to distinguish maize leaves from other non-relevant inputs (e.g., hands, background soil, other crops).
- If the image fails the validation, the system immediately alerts the user and prompts them to retake or upload a new image.

This step helps:

- Prevent misclassification and false positives
- Improve system accuracy and confidence
- Save processing resources by rejecting irrelevant inputs early

2.4.3. Location Mapping

Understanding where an infestation is detected is crucial for tracking the spread of the Fall Armyworm pest across regions.

- Upon image submission, the system captures real-time GPS coordinates from the user's mobile device.

- These coordinates are automatically reverse-geocoded into a district name using geospatial APIs.
- The coordinates are validated to ensure that they fall within Uganda's borders. If they don’t, the system blocks further processing and notifies the user.

Location mapping allows:

- Aggregation of infestation data at the district level
- Generation of geospatial maps showing pest distribution
- Support for targeted intervention by district-level agricultural authorities

2.4.4. Analytics and Reporting

The system incorporates a powerful analytics engine that processes all stored detection data to generate real-time insights for both farmers and policymakers.

Key analytical features include:

- **Time-series graphs** showing daily, weekly, and monthly trends in infestation rates
- **District-level summaries** that display the number and severity of detections by location
- **Comparison tools** that highlight percentage increases or decreases in infestations over selected periods
- **Stage distribution charts** showing the breakdown of detections across stages (e.g., 40% Larval Damage, 30% Frass)

Additionally, after each detection, the system enables the user to generate a downloadable report containing:

- Detection result and confidence score
- Leaf image and GPS location
- Stage-specific treatment recommendations
- Weather data and timestamp

These tools empower users to make data-driven decisions and keep digital records of their farm health status.

2.4.5. Weather-Based Recommendations

Weather plays a critical role in the effectiveness of pest control, especially in pesticide application. Rain, for instance, can wash away chemicals, making treatment ineffective or even harmful.

To address this:

- The system integrates with the **OpenWeatherMap API** to retrieve weather data such as rainfall probability, humidity, and temperature, based on the user's district.
- The treatment recommendation engine takes weather into account and advises farmers when to spray and when to wait.

For example:

- If rain is expected within 6 hours, the system may suggest delaying pesticide application.
- It can also advise on spraying in the early morning or late evening when conditions are cooler and evaporation is reduced.

This functionality improves pesticide efficiency, reduces waste, and protects the environment from unnecessary chemical runoff.

2.4.6. Community Support

In addition to automated detection, the system also provides a community-based knowledge-sharing platform where farmers can engage with experts and each other.

- Users can post text-based queries describing symptoms or unusual occurrences in their crops.
- They can attach images of their plants or affected fields.
- Responses are provided by trained agricultural extension workers or system administrators.
- All responses are stored and made publicly viewable to benefit other farmers with similar issues.

This module serves as a support network, allowing for problem-solving in cases where the model may not cover specific or rare issues (e.g., chemical resistance, soil-related problems).

2.4.7. Detection History

To help farmers and researchers track progress over time, the system maintains a full detection history log for each user.

This includes:

- All previously uploaded images
- Detection results with timestamps
- Geographical locations of each detection
- Generated reports
- Notes or follow-ups on treatment outcomes

Farmers can use this history to:

- Compare infestation levels over time
- Show progress in pest management to agricultural inspectors
- Monitor effectiveness of applied treatments

2.5 Limitations and safety

While the system has been designed for robustness and reliability, certain limitations were identified during implementation:

- **Dependency on Internet Connection:**
  Most features (disease detection, weather integration, map display, community module) require an active connection.
- **Image Quality Sensitivity:**
  Blurry, low-light, or obstructed images may reduce detection accuracy.
- **Geographical Limitation:**
  The system currently validates GPS data only within Uganda's national boundaries.

Safety Measures:

- Input validation is enforced to ensure quality and accuracy of detection.
- Role-based access control protects sensitive features.
- District validation prevents the inclusion of unsupported geographic locations.

2.6 Default settings

- **Default Display Settings:**
- The analytics dashboard shows the past 30 days by default.
- The map zoom is centered to show all Ugandan districts.

Default Weather Integration:

- The system uses OpenWeatherMap API for real-time local forecasts.

2.7 Special requirements

To maintain data integrity, system availability, and user confidentiality, the following measures were implemented:

Security:

- Encrypted communication via HTTPS
- Secure authentication for all system users

Data Backup and Redundancy:

- Weekly remote backups are scheduled.
- Versioned backups ensure data recovery in case of failure.

Code Protection and Rate Limiting:

- API keys are securely stored and not hardcoded.
- Requests to the backend are rate-limited to prevent abuse.

Controlled Update Environment:

- A staging environment is used for testing new features before live deployment.

2.8 Errors and alarms

The system is designed to handle errors gracefully, providing clear feedback to the user in the following cases:

Error Condition	System Response
No maize leaf detected in image	"Please upload a clear image of a maize leaf."
Invalid or missing GPS coordinates	"Location not detected. Please enable GPS and try again."
Poor model confidence (<60%)	"Uncertain result. Please capture another image."
Internet disconnected	"Network error. Please check your internet connection."
Out-of-boundary coordinates	"Detection is limited to Uganda. Please retake from the valid area."

CHAPTER 3: DESIGN OUTPUT

3.1 Implementation (Coding and Compilation)

The Fall Armyworm Detection System was developed using a carefully chosen set of tools and technologies to support mobile development, backend services, machine learning integration, data management, version control, and deployment. Each component played a key role in the smooth and efficient development, testing, and deployment of the system.

Mobile Application: Flutter was used to develop the mobile application, with the goal of delivering a fast and responsive Android experience. The app allows users to capture or upload maize leaf images, submit them for classification, and receive infestation diagnosis and treatment recommendations. The interface also provides access to weather information and a community support board. During development, Android Studio was used as the primary IDE for running emulators and testing APK builds on multiple screen sizes and device versions to ensure stability across Android environments.

REST API Backend Server: Flask, a Python-based microframework, served as the backbone of the backend. It was structured in a FastAPI-like modular architecture to expose well-organized RESTful endpoints for image classification, geolocation, weather data retrieval, report generation, and community post management. The backend also handled all logic associated with preprocessing images, validating maize leaves before classification, and responding with confidence scores and advisory content. The Flask server communicated with a MySQL database for persistent data storage and logged every detection for possible reporting or future analysis.

Machine Learning Model: The backend hosts two TensorFlow Lite models, a binary CNN model for maize leaf validation (pre-screening) and a MobileNet-based multi-class classifier for predicting infestation stages. These models were trained on large, labeled datasets using Google Colab and later optimized for fast inference with TensorFlow Lite. Model files were served directly through the Flask backend, and image data from users was processed and analyzed in real time upon upload.

Database: A structured MySQL relational database was used to store detection results, user information, weather lookups, image metadata, and community board messages. The schema

supports foreign keys and indexing for quick query response, ensuring that data can be accessed efficiently when generating reports or performing analytics.

External APIs: The system integrates with third-party services to provide enriched and context-aware functionality. OpenWeatherMap's API supplies real-time weather data based on the user's district, informing treatment timing. Reverse geocoding APIs are used to translate GPS coordinates collected from the mobile device into recognizable district names in Uganda, which are used for location-tagging detections.

Code Editors and Development Tools: Two main code editors were used throughout the development cycle. Visual Studio Code (VS Code) served as the primary environment for backend development in Python and for managing project files, version control integration, and environment variables. Android Studio was used to manage the Flutter mobile application, enabling real-time previews, emulator testing, and APK generation. Both editors supported plugins for linting, debugging, and Git version control.

API Testing: To verify that REST endpoints were functioning as expected, Postman was used extensively during development and testing. It allowed the team to simulate API requests for uploading images, retrieving classification responses, checking weather data integrations, and testing authentication headers. Each endpoint was tested with various payloads to ensure reliable and secure interaction between frontend and backend components.

Hosting and Deployment: The Flask backend, machine learning models, and MySQL database were deployed to the cloud using Render, a modern cloud hosting platform known for its simplicity and developer-friendly configuration. Render was selected because it not only supports Python-based applications seamlessly but also provides a preconfigured deployment pipeline. This means the application can be continuously deployed directly from the GitHub repository without setting up external CI/CD tools. On every push to the production branch, Render automatically rebuilds and redeploys the backend. It also manages HTTPS encryption, environment variable management, and server scaling. This streamlined setup reduced the operational overhead and allowed the team to focus more on feature development rather than server management.

Version Control: Git was used for local version control, enabling the development team to track changes, manage branches, and resolve conflicts collaboratively. GitHub served as the remote repository, where all source code for both the mobile application and the backend was maintained. GitHub also acted as the integration point for continuous deployment to the Render platform. Pull requests, commit messages, and issue tracking were used to ensure organized development and team collaboration.

Operating Environment: The mobile application runs on Android smartphones with version 8.0 (Oreo) or higher, as these are commonly used by farmers in rural Uganda. The backend was developed and tested locally on machines running Windows 11 and Ubuntu 22.04 LTS and deployed to the cloud via Render's Linux-based environment. A reliable internet connection is required for users to upload images, retrieve predictions, and access real-time weather and expert advice. Microsoft Word was used for writing documentation and preparing the final system report.

3.1.1 Notes on Development Anomalies

During the implementation of the Fall Armyworm Detection System, the development team encountered several anomalies and challenges that affected the workflow, system behavior, and scope of functionality.

Dataset Limitations: One of the most significant early challenges was finding a suitable dataset that included all desired infestation stages of the Fall Armyworm: Healthy, Eggs, Frass, and Larval Damage. Most publicly available datasets focused only on binary classification (e.g., diseased vs. healthy) and lacked sufficient images for intermediate stages. Since the system was specifically designed to detect and recommend treatment based on infestation stage, this limitation was critical. The team had to manually compile and clean images from different sources and apply their own labeling process, which was time-intensive and delayed the training phase.

Flutter SDK and Gradle Incompatibilities: While developing the mobile application in Flutter, the team encountered build issues caused by incompatibilities between the Flutter SDK version and the Gradle build system. These inconsistencies led to repeated errors during project compilation, particularly when using newer or auto-updated plugins that conflicted with older Gradle versions. Resolving this required overriding default build configurations by explicitly forcing the SDK to use a specific version of Gradle. This was done by adjusting the project's

Gradle wrapper and build scripts to lock in compatible versions, allowing the application to build successfully and maintain stability during development.

API Integration with Mobile App: Establishing communication between the Flutter frontend and the Flask backend introduced several bugs during integration. Problems such as CORS errors, authentication failures, incorrect request formatting, and header mismatches caused API calls to fail. These issues required thorough debugging and careful adjustment of both frontend request logic and backend endpoint handling to ensure secure and reliable communication.

Loss of Detection Results on Hosting: After deploying the backend to Render, an unexpected problem occurred where detection results were not being saved on the server. While users could view real-time predictions, no permanent records were stored, meaning that data would be lost after restarting the server or closing the app. This issue was traced to the lack of configured persistent storage during early deployment. To resolve it, the team upgraded the hosting plan and paid for external database storage, which added unanticipated costs to the project budget.

Changing Feature Requirements: As development progressed, some initial features had to be revised or removed entirely due to time constraints, technical feasibility, or shifting project priorities. For example, the feature for listing all past detections was removed in favor of providing downloadable reports. Similarly, ideas like push notifications and multilingual support were dropped to focus development time on more impactful components such as weather-based treatment advice and expert community engagement. These changes required refactoring or removing previously written modules and adjusting the user flow accordingly.

3.1.2 Module and Integration Details

The Fall Armyworm Detection and Monitoring System is built upon several core functional modules that work in synergy to ensure accuracy, user-friendliness, and relevance for smallholder farmers. Each of these modules plays a specific role in the data pipeline, from image capture to disease detection, to report generation and expert feedback. Below is a comprehensive explanation of each module and its role in the system.

Image Capture and Processing

The heart of the system lies in its image classification capability, which is responsible for identifying signs of Fall Armyworm infestation on maize leaves. When a user submits an image, either captured via the phone camera or selected from gallery, it is first preprocessed by

resizing and normalizing it to meet the input requirements of the TensorFlow Lite model. The machine learning model, which is a modified version of MobileNet, has been trained to classify the image into one of four categories: Eggs, Frass, Larval Damage, or Healthy (no infestation). The MobileNet architecture was chosen for its efficiency and suitability for deployment on mobile and edge devices without sacrificing accuracy. Once the prediction is made, the app returns a result that includes the classification label, a confidence score (e.g., 93.6%), symptoms and a tailored treatment recommendation. This result is immediately presented to the user through a clear and responsive interface, helping them make timely decisions on pest control interventions.

FAW Detection and Classification

Every time a detection is performed within the application, the system generates and displays a detailed result screen summarizing the event. This detection summary includes the original image submitted by the user, the identified infestation stage with a model confidence score, the characteristic symptoms associated with that stage, and tailored treatment recommendations. The results are returned immediately after analysis and are presented in a visually structured format.

Below this, two interactive buttons allow users to either “Confirm & See Treatment,” which leads to detailed advice on how to manage the infestation, or “Download Report,” which generates a PDF version of the detection record that can be stored locally. Additionally, a voice output feature is embedded within the result screen, enabling users to hear the detection results and treatment advice read aloud.

All detection records are stored in a structured SQL database hosted in the cloud. To perform a detection or retrieve results, users must be online.

Mapping FAW Infestations

To contextualize infestation data geographically, the system includes a mapping module. This module retrieves GPS coordinates (latitude and longitude) from the user’s smartphone using the native location services. These coordinates are then sent to the Google Map API, which returns the district and region corresponding to the user’s location. This spatial tagging of detection results is crucial for analyzing disease patterns over time and identifying regional

outbreaks. All geolocation data is stored in the backend SQL database along with the detection records for historical tracking and visualization.

Community Support

To support collaborative learning and expert guidance, the system includes a community support feature where users can post questions, images, and descriptions of pest-related issues. This acts like a lightweight discussion forum built within the app, allowing users to get feedback from both peers and agricultural experts. Each user post is associated with metadata such as user identity (which can be anonymized), postdate, and optional image attachments. This system is powered by Firebase's real-time database and authentication mechanisms, ensuring low-latency message delivery and organized thread management. It fosters a sense of community and enables knowledge sharing, which is particularly valuable in rural settings where access to agricultural professionals may be limited.

Analytics and Reporting

The analytics and reporting module plays a crucial role in empowering farmers with clear and actionable insights derived from their detection history. When a farmer submits a maize leaf image and it is processed by the system, the results are not only stored but also aggregated and visualized in ways that help the farmer make better decisions. Upon accessing the analytics section of the app, the farmer is presented with a summary of their detection activity, including the Total detections, Districts affected, infestation rate and recent trend. But this is more than just a count; the app interprets this data to highlight how severe the infestation is becoming over time, whether their plants are mostly healthy, or if there's a recurring pattern of damage, like repeated larval-stage infestations.

What makes this module particularly powerful is its ability to personalize the data based on location, period and district. By using filters built into the interface, a farmer can choose to view detection trends from a specific date range. They can even filter by the type of detection for example, viewing only detections classified as "Eggs" or "Healthy" to assess urgent needs.

Visual representations make this data easier to understand at a glance. A pie chart shows how the detections are distributed across the various infestation stages, offering a quick snapshot of how widespread or controlled the issue is. A line chart displays detection trends over time,

helping farmers observe whether the infestation is worsening, stabilizing, or declining. Furthermore, horizontal bar charts provide two more views, one that compares how different infestation types are distributed in the farmer's data, and another that reveals how infestation severity varies by district. Each of these visualizations is interactive, updating in real time as the farmer adjusts the filters for date, location, or detection type.

This module does not require the farmer to have any technical background; the goal is to present data in a form that is intuitive and immediately helpful. All this information is stored securely in a centralized SQL database and is fetched dynamically when the user opens the analytics view. Since we are no longer supporting offline access, the farmer must have an internet connection to access these insights. However, the real-time nature of this setup ensures that farmers are always working with the most up-to-date information, potentially saving crops and improving yields through data-driven action.

Integration Details

Mobile App (Flutter) to Backend (Flask): The Flutter mobile application communicates with the Flask-based REST API backend to perform key functions such as image classification, detection result retrieval, treatment recommendation, weather forecasting, and report generation. API calls are securely transmitted via HTTPS. Detection data and metadata are stored in a MySQL database, and endpoints were tested using Postman to ensure reliability and consistency across the mobile-backend interface.

Flask Application Server: The backend is implemented using Flask, a Python-based microframework. It serves all RESTful API endpoints consumed by the mobile frontend. These endpoints handle image submission for classification, detection result logging, user registration and login, report generation, and access to advisory content. Its lightweight design enables fast response times, which is critical for users in low-connectivity rural areas.

Asynchronous Operations with FastAPI Logic: While Flask handles core routing, asynchronous operations such as real-time weather data retrieval and image pre-screening are handled using FastAPI-like patterns. These ensure non-blocking I/O when fetching data from third-party services, leading to a smoother user experience on the mobile app.

SQL Database Integration: Detection results, classification logs, image paths, timestamps, GPS coordinates, and weather information are stored in a cloud-hosted MySQL database.

Firebase Integration: Firebase is selectively integrated to handle user authentication and community post management. It manages sign-in and sign-up flows, and stores posts from the app's knowledge-sharing community. However, Firebase is not used for detection storage or model processing. This scoped use reduces overhead and leverages Firebase's strength in real-time data and identity management.

Weather API Integration: The backend connects to the OpenWeatherMap API to retrieve localized weather data based on the user's district (determined from GPS). This data is included in detection results and used to influence treatment recommendations. For example, if the system forecasts rain, it recommends delaying pesticide application.

Image Pre-screening Module to backend: Before classification begins, the backend uses a lightweight CNN model to verify that the uploaded image contains a maize leaf. This filter prevents irrelevant inputs from being processed and conserves computational resources. If the image does not pass this stage, an appropriate error is returned to the frontend.

TensorFlow Lite Inference Engine: Once validated, the backend uses the TensorFlow Lite runtime to load a MobileNetV2-based model for multi-class classification. The input image is resized, normalized, and passed through the model, which predicts one of four stages: Healthy, Eggs, Frass, or Larval Damage, along with a confidence score.

Integration with Hosting and Storage: The backend is hosted on Render, which provides automated builds, deployment from GitHub, and environment variable management. Persistent data storage was configured separately to ensure that detection results are saved across sessions. Initially, detections were not stored due to lack of database configuration on Render, which was later resolved by upgrading the hosting tier and paying for persistent storage.

Firebase Analytics Integration: The mobile app integrates Firebase Analytics to monitor user interactions, detect behavior patterns, and assess feature engagement in real time. A centralized AnalyticsService class is responsible for logging key events such as image uploads, detection attempts, treatment card views, navigation flows, and button clicks. This service also tracks app open times, screen views, and user-specific metadata (e.g., login method, user ID). Events

are enriched with timestamps and contextual information, allowing the development team to analyze app usage trends and performance over time.

3.1.3 Model Training and Deployment

The development and deployment of the Fall Armyworm detection model was a multi-stage process designed to ensure both high accuracy and operational efficiency on mobile platforms. The entire pipeline from dataset preparation through model training and optimization to real-world deployment was carefully engineered to work seamlessly with the mobile application. This section outlines how the machine learning system was built to provide timely and accurate classifications of maize leaf infestations, empowering farmers with actionable insights directly from their smartphones.

The process began with the preparation of a comprehensive dataset composed of thousands of images of maize leaves, each annotated according to four infestation categories: Healthy, Eggs, Frass, and Larval Damage. The dataset was sourced from various open-access agricultural image repositories and research datasets that already contained labeled images. This pre-annotated data ensured consistency and saved significant time in the preparation process. To ensure effective model training and validation, the dataset was partitioned into three subsets: 70% for training, 15% for validation, and 15% for testing. This stratification ensured that the model was exposed to enough examples to learn effectively while still being evaluated on unseen data for performance assessment.

To improve the model's ability to generalize to real-world variations such as differences in lighting, camera angle, or leaf orientation several data augmentation techniques were applied. These included random rotations, shifts, zooming, adjustments in brightness and contrast, and both horizontal and vertical flips. Such augmentation simulated natural variability in the environment and helped the model become robust to imperfect image capture conditions that are common in farming scenarios.

The model architecture was centered around MobileNetV2, a lightweight convolutional neural network specifically designed for efficient performance on mobile and edge devices. MobileNetV2 was selected because of its proven balance between accuracy and computational efficiency, allowing it to run smoothly even on devices with limited resources. It also supports integration with TensorFlow Lite (TFLite), a framework for deploying machine learning models in resource-constrained environments. The architecture took RGB images resized to

224x224 pixels as input and processed them through the MobileNetV2 backbone, culminating in a softmax layer with four output neurons each representing one of the four defined infestation classes.

To ensure that only relevant images were classified by the main model, a lightweight binary convolutional neural network was also trained as a pre-screening filter. This model was responsible for validating whether the uploaded image actually depicted a maize leaf. If the image failed this check for example, if it showed an unrelated crop, or a blurry background it was rejected and the user was prompted to submit a clearer or more relevant photo. This two-step classification pipeline improved the reliability and trustworthiness of the system's outputs.

The training of both the main detection model and the pre-screening model was conducted using the TensorFlow framework in a GPU-enabled environment provided by Kaggle. The categorical cross-entropy loss function was used because the task was a multi-class classification problem. The Adam optimizer, known for its adaptive learning rate and fast convergence, was employed during training. The models were trained with a batch size of 8 over 50 epochs, although early stopping was implemented to prevent overfitting.

Once a satisfactory level of accuracy was achieved, the trained MobileNetV2 model was converted into the TFLite format using the tflite_convert tool. This conversion made it suitable for mobile deployment. Further optimization steps were applied to enhance the model's performance in production. Quantization was used to convert the model weights from float32 to int8, reducing its size and increasing inference speed without significantly compromising accuracy.

Finally, the optimized TFLite model was integrated into the backend server, which was built using the Flask web framework. The server was configured to accept image uploads from the mobile app, preprocess them, and then run inference through both the pre-screening and classification models using the tflite_runtime library. The resulting prediction, indicating the infestation stage along with the associated confidence score was returned to the client as a JSON response. This backend service completed the loop by enabling real-time feedback to the user, turning their mobile phone into a powerful diagnostic tool for managing Fall Armyworm infestation in maize crops.

3.2 Device Interfaces and Equipment Used

Table 1: Device interfaces and equipment used

Device Type	**Purpose**
Mobile Phones	Primary devices for interacting with the application. Users can capture and upload maize leaves, view results, download reports.
Laptops	For development, debugging, version control.

The system is designed primarily for use on mobile phones, which serve as the sole user-facing interface. Farmers interact with the application entirely through their smartphones, capturing maize leaf images, submitting them for analysis, receiving diagnosis results, downloading PDF treatment reports, checking localized weather forecasts, and engaging with the expert community forum.

The minimum device requirements include a functioning smartphone camera and GPS capability. The camera is used to capture images of maize leaves for classification, while the GPS sensor enables the application to determine the user's district. This geolocation data is essential for reverse geocoding, weather integration, and district-specific analytics.

On the development side, laptops played a critical role throughout the project lifecycle. All programming, debugging, model integration, version control, and deployment activities were carried out on laptops running Windows 11 and 10. These machines were used to develop both the Flutter mobile application and the Flask backend server. They also facilitated local testing, interaction with Android emulators, pushing commits to GitHub, and deploying the backend to the Render cloud hosting platform.

3.3 Hardware and Software Environment

The application was developed and tested using standard office laptops running Windows 10 and Windows 11 operating systems. These machines were equipped with modern processors and at least 8 GB of RAM, allowing smooth execution of both frontend (Flutter) and backend (Flask) development environments. Android Studio and Visual Studio Code were used as the primary IDEs for building, testing, and deploying the application.

All machine learning model training was performed using Kaggle's cloud environment, which provided access to GPUs for accelerated training of the maize leaf classification models. The use of GPU-based compute in Kaggle allowed the team to efficiently train and evaluate the MobileNetV2 and pre-screening CNN models on annotated image datasets.

The mobile application is designed for Android smartphones, with a minimum supported version of Android 8.0 (Oreo). Testing was conducted on a variety of devices including Tecno, Samsung, and Infinix phones to ensure broad compatibility and performance across screen sizes and device capabilities.

A reliable internet connection is required for full application functionality. Network connectivity is necessary for uploading images to the backend, receiving classification results, retrieving real-time weather forecasts, and interacting with the expert community. During testing, a minimum of 3G network performance was assumed to simulate real-world usage scenarios in rural Uganda.

Documentation and final report writing were completed using Microsoft Word. Early collaboration and note-sharing were managed via Google Docs. GitHub Issues and pull requests were used for basic task tracking, version control, and coordinating changes across the development team.

3.4 Documentation

The Fall Armyworm Detection System is accompanied by a comprehensive set of documentation designed to support developers, and end-users (primarily farmers) in understanding, deploying, and using the system effectively.

Software Design Document (SDD): The Software Design Document outlines the technical structure of the system, including detailed diagrams and architecture descriptions. It features an Entity-Relationship Diagram (ERD) that maps the structure of the SQL database, showing relationships between users, detections, and treatment records. A System Architecture Diagram illustrates the interaction between the Flutter mobile frontend, Flask REST API backend, TensorFlow Lite model layer, Firebase Authentication, MySQL database, and third-party APIs such as OpenWeatherMap. The SDD also includes a Functional Module Overview highlighting key components such as Image Classification, Community Interaction, Weather Integration, and Report Generation.

User Manuals: A dedicated user manual has been developed for farmers, who are the primary end-users of the system. This guide provides clear, step-by-step instructions on how to install the mobile application, capture and upload maize leaf images for analysis, interpret detection results, access weather-based treatment recommendations, and download PDF reports. It also includes guidance on signing up, logging in, and participating in the community forum.

README File: The GitHub repository for the project includes a comprehensive README.md file that serves as an onboarding guide for developers and stakeholders. It provides an overview of the system's purpose, and outlines core features such as image classification, weather integration, treatment guidance, report generation, and community support. The README clearly describes the system architecture (Flutter frontend, Flask backend, TensorFlow Lite model, and OpenWeatherMap API), technical specifications (version control with Git and GitHub, GPS validation, HTTPS security), and setup instructions for running the Flutter app. It also lists contributors, code statistics, and development branches, serving as both a technical reference and collaboration entry point.

Source Code Repository: The entire codebase for the system, including the Flutter mobile app, Flask backend server, model inference scripts, and Firebase integration logic, is maintained in a public GitHub repository. Version control is managed using Git, and branches are used to organize features and hotfixes. Commits, pull requests, and changelogs provide a detailed development history, supporting transparency and traceability.

Table 2: Design details

Topics	**Design output**	
Good programming practice *Efforts made to meet the recommendations for good programming practice...*	Source code is... ☑ Modulized ☑ Encapsulated ☐ Functionally divided ☐ Strictly compiled ☑ Fail-safe mechanisms implemented	Source code contains... ☐ Revision notes ☑ Comments ☑ Meaningfull names ☑ Readable source code ☑ Printable source code
Windows programming *If implementing Windows applications...*	☑ Interface implemented using standard Windows elements ☐ Interface implemented using self-developed Windows elements ☐ Application manages single/multiple running instances	
Dynamic testing *Step-by-step testing made dynamically during the implementation...*	☐ All statements have been executed at least once ☑ All functions have been executed at least once ☐ All case segments have been executed at least once ☐ All loops have been executed to their boundaries ☐ Some parts were not subject to dynamic test	

3.5 System Architecture

The Fall Armyworm detection system is built upon a client-server architecture that embraces modularity through loosely coupled components. This design approach enhances the system's flexibility and maintainability, allowing each layer to operate independently while still communicating effectively with the others. The architecture is composed of several interconnected layers, each with a distinct responsibility, working together to deliver a smooth and responsive user experience on mobile devices.

The presentation layer, also referred to as the frontend, is developed using Flutter and is optimized for mobile phone deployment. This layer is responsible for all user interactions, including image capture for disease detection, displaying the analysis results, and providing access to herbicides and expert insights. Additionally, users can use this interface to engage with a community forum, where they may ask questions or offer advice. The frontend

communicates with the backend server through structured HTTP requests, ensuring seamless data exchange and real-time responsiveness.

The application layer, commonly known as the backend, is implemented using Flask, a lightweight and versatile Python-based web framework. Although built with Flask, the structure is influenced by FastAPI to offer a clear and scalable routing system. This layer orchestrates the core logic of the system, such as managing API endpoints for image submission, retrieving weather updates, mapping geographic coordinates to specific locations, and handling communication between users. Furthermore, it handles the preprocessing and postprocessing of image analysis results and manages connections to the database layer. The backend also integrates a machine learning model for image classification, which is discussed in the next layer.

The machine learning model operates as part of the backend logic and plays a critical role in detecting signs of Fall Armyworm infestation. The system first runs a lightweight convolutional neural network (CNN) as a pre-screening step to confirm whether the submitted image is of a maize leaf. Upon successful validation, the image is passed to a more advanced model based on MobileNet architecture, optimized with TensorFlow Lite to suit mobile performance requirements. This classifier is capable of distinguishing between multiple infestation stages, such as Healthy, Eggs, Frass, and Larval Damage. By incorporating this two-step verification and classification mechanism, the system ensures that results are both accurate and efficient.

The data layer is implemented using a MySQL relational database. This layer serves as the central repository for all persistent information related to the system, including image metadata, diagnosis results, GPS data, weather logs. Its schema is carefully designed to be scalable and to support fast data retrieval, ensuring that both individual farmers and system administrators can access historical data as needed without delays.

Finally, the architecture integrates external APIs to enrich the system's functionality. The Open Weather API provides up-to-date weather forecasts such as temperature, and chances of rainfall, which are crucial for giving context-aware treatment advice. Additionally, a reverse geocoding API is employed to translate raw GPS coordinates into human-readable district names. This geographic localization helps the system provide region-specific insights, such as weather-based spraying warnings and monitoring regional disease patterns.

Together, these layers form a cohesive architecture that supports a robust and farmer-centric disease detection system tailored specifically for mobile users in agricultural communities.

CHAPTER 4: INSPECTION AND TESTING

4.1 Introduction

The inspection and testing of the Fall Armyworm Detection System were meticulously planned and executed in accordance with a predefined test plan. The depth and scope of the testing align with system requirements, system acceptance criteria, risk levels, functional complexity, and the intended use of the mobile application. The overall goal was to ensure that the system is reliable, accurate, and ready for deployment among smallholder farmers and agricultural officers in Uganda.

Design Output Inspection

Table 3: Design output inspection

Item	**Details**
Inspection Performed	Results from the Design Output section inspected.
Date / Initials	03/04/2025 / GW and BL
Comments	All UI flows matched user requirements. Minor color adjustments needed.

Documentation Inspection

Table 4: Documentation inspection

Item	**Details**
Inspection Performed	Documentation inspected for accuracy and completeness.
Date / Initials	05/04/2025 / NM
Comments	API documentation and user guide aligned with project scope. Add data schema diagram.

Software Development Environment Inspection

Table 5: Software development environment inspection

Item	**Details**
Inspection Performed	Environment elements inspected (IDE, Flask framework, Firebase).
Date / Initials	06/04/2025 / SA
Comments	Python and Flask properly configured. Firebase authentication working as expected.

Final Inspection Result

Table 6: Final inspection result

Item	Details
Approval	Approved after final review.
Date / Initials	01/04/2025 / All Team Members
Comments	Inspection passed with minor revisions noted above. All critical issues resolved.

4.2 Test plan and performance

The test plan for the system was developed during the implementation phase and detailed all components subject to testing. The plan specified what was to be tested, the criteria for success, testing procedures, expected outcomes, and actual results. Each test case was validated and recorded to ensure traceability and compliance with the system's functional and non-functional requirements.

4.2.1 Test objectives

The objectives of testing were to:

- Verify accurate image classification of Fall Armyworm stages that are eggs, larva and frass.
- Validate all Flask REST API endpoints.
- Ensure integration between components (camera, detection model, map, Firebase).
- Evaluate app responsiveness and performance in low-connectivity rural settings.
- Confirm system usability for non-technical users (e.g., farmers, extension workers).

4.2.2 Scope and Relevance of tests

- The tests covered all core modules: image capture and classification, treatment recommendation engine, weather data retrieval, community interactions, and geospatial analytics. The scope was designed to simulate real-world usage scenarios reflecting the complexity of field operations and the volume of user interactions expected during infestations

The tests covered:

Image Capture & Detection (Camera, Pre-screening, MobileNet-based classification)

Geolocation & Mapping (GPS tracking, district tagging)

Treatment Advice Engine (Recommendation generation based on stage and weather)

Analytics & Plotting (Time-based trends, stage distributions)

Community Interaction (Posting, commenting, viewing expert replies)

Backend Endpoints (REST API for detections, reports, weather, map view)
Frontend Integration (UI rendering, navigation, Firebase auth)

4.3 Anomalous conditions

Given that the application operates in Android and cloud environments (Firebase, Flask backend), several anomalous conditions were identified and addressed:

- Network interruptions during data sync
- Misalignment between predicted and actual detection due to poor image quality
- Inconsistencies in behavior across different Android OS versions

These discrepancies were documented, and where possible, corrective actions were implemented.

4.3.1 Precautionary steps taken

To mitigate potential issues:

- Added **image capture guidelines** to improve quality (tips feature).
- Required **location access prompt** before detection begins.
- Used **Firebase token refresh routines** for stable authentication.
- Extensive image quality guidelines were incorporated into the app to improve detection accuracy.
- The system was tested on various Android OS versions to ensure compatibility.
- Known software limitations were documented in the user manual with corresponding workarounds.

CHAPTER 5: INSTALLATION AND SYSTEM ACCEPTANCE TEST

This chapter provides a comprehensive overview of how we installed and validated the Fall Armyworm Detection System. We documented the systematic installation process for both the mobile application and server components, ensuring all system elements were properly configured in their target environments. Our installation qualification procedures verified that each component functioned correctly after deployment, with attention to the system's ability to operate in environments with varying connectivity levels. The acceptance testing confirmed that the installed system met all specified requirements and was ready for operational use by agricultural extension workers and farmers.

5.1 Input Files

The Fall Armyworm Disease Detection System installation package contains the following key files that are essential for proper system deployment:

Table 7: List of relevant files on the installation media.

File	Purpose	Location
FallArmyworm_v2.0.apk	Android application package for installation on mobile devices	/installation/mobile/
server_deployment.zip	Backend server deployment package including Flask API and Docker containers	/installation/server/
ml_models.tar.gz	Compressed machine learning models (MobileNet architecture) for disease detection	/installation/models/
database_schema.sql	SQL scripts for initializing the MySQL database with reference data	/installation/database/
initial_data.json	Initial reference data including treatment recommendations and district information	/installation/data/
uganda_regions.geojson	GeoJSON files containing boundary data for all Ugandan districts	/installation/maps/

connectivity_settings.conf	Network configuration optimized for Ugandan mobile networks	/installation/config/
deployment_guide.pdf	Comprehensive deployment guide with infrastructure considerations	/installation/docs/

The installation media was organized to support both centralized deployment at the server level and distributed installation on mobile devices used by agricultural extension workers and farmers across Uganda.

5.2 Supplementary Files.

In addition to the core installation files, the following supplementary files were included to support installation, configuration, and usage:

Documentation Files

Table 8: Table showing supplementary files.

File	Description	Location
README.md	Overview with deployment notes and quick start guide	/supplementary/docs/
CHANGELOG.md	Detailed list of changes between versions	/supplementary/docs/
LICENSE.txt	License agreement for the software	/supplementary/legal/
PRIVACY_POLICY.pdf	Data privacy policy for user information	/supplementary/legal/

Example and Reference Files

Table 9: table showing reference files

File	Description	Location
sample_images/	Directory with Fall Armyworm images at different stages	/supplementary/samples/
treatment_guide.pdf	Treatment guides for different infestation stages	/supplementary/reference/
training_materials/	Training materials for system users	/supplementary/training/
support_contacts.csv	Contact information for technical support	/supplementary/support/

Configuration Templates

Table 10: Table showing the configuration templates.

File	Description	Location
server_config.conf	Configuration for central server deployment	/supplementary/config/
mobile_config.conf	Configuration for mobile application deployment	/supplementary/config/
backup_scripts/	Scripts for automated data backup	/supplementary/scripts/
sync_settings.conf	Configuration for data synchronization	/supplementary/config/

These supplementary files ensured that installers and users had access to all necessary information for successful deployment, configuration, and operation of the Fall Armyworm Detection System.

5.3 Installation Qualification.

The installation qualification process ensured that all components of the Fall Armyworm Detection System were correctly installed and configured. We followed a systematic approach to verify each component's installation status.

Table 11: Checklist of the Installation and System Acceptance Test

Topics	Installation summary	Date / Initials
Installation method	The system used a two-pronged installation approach: Mobile application: Manual APK installation for offline distribution. Backend services: Flask API server, machine learning models, and database Each component included verification steps to confirm successful installation.	15/04/2025 / SA, NM
Comments:	We tested the installation method in field environments to ensure viability and proper functioning of all the features.	

Installation media	Installation files were distributed through multiple channels: Mobile application: Direct APK distribution Server components: Configuration files- Documentation: Digital formats (PDF, HTML) and printed quick-start guides. All installation media included integrity verification through SHA-256 checksums.	15/04/2025 / BL
Comments:	Distribution channels were selected based on target user requirements, with particular attention to accessibility for users with limited technical expertise.	
Installed files	The installation process deployed the following file types:- DBK files: Database backup files stored at /var/opt/faw/backup/ on servers and /sd card/Fall Armyworm/backup/ on mobile devices- HEX files: Compiled firmware for optional weather sensors at /opt/faw/firmware/- PWI files: Pre-weighted model files at /opt/faw/models/ and /sd card/Fall Armyworm/models/- Plg files: Plugin modules for additional functionality at /opt/faw/plugins/ and /sd card/Fall Armyworm/plugins/- BAK files: Automatic backup files at /var/opt/faw/backup/auto/ and /sd card/Fall Armyworm/backup/auto/- Opt files: Optimization configuration at /etc/faw/config/ and /sd card/Fall Armyworm/config/- C files: Source code for custom compilation at /usr/src/faw/	16/04/2025 / GW
Comments:	File locations were selected to follow standard conventions for both server and mobile environments, ensuring ease of maintenance and troubleshooting.	

Table 12: Installation Procedure Check

Topics	**Installation procedure**	**Date / Initials**
Authorization Approval of installation in actual environment	The person responsible: Ssebatta Adam	20/04/2025 / SA
Installation test The following installations have been performed and approved	● Tested and approved in the test environment ● Tested and approved in the actual environment ● We fully tested all functionalities according to our test plan and goals.	22/04/2025 / GW, BL, NM
Comments:	Installation testing revealed minor issues with database connectivity that were addressed with configuration adjustments. Mobile application installation was verified on multiple Android versions (8.0-13.0) to ensure compatibility with a wide range of devices.	

The installation qualification process confirmed that the Fall Armyworm Detection System was properly installed across all required environments. All components were verified for correct installation, and the system was ready for operational qualification and user acceptance testing.

CHAPTER 6: PERFORMANCE, SERVICING, MAINTENANCE, AND PHASE OUT.

6.1 Service and Maintenance.

The Fall Armyworm Disease Detection System requires ongoing service and maintenance to ensure continued effectiveness, with particular attention to the evolving nature of the pest, changes in agricultural practices, and advancements in technology.

Support Structure

We have established a single-tiered support structure for the system, focused on accessibility and responsiveness at the community level:

Primary Support: Support will be provided by trained agricultural extension workers, who are equipped to address basic usage questions, perform simple troubleshooting tasks and collect and document information on more complex issues for potential future escalation if needed. This approach ensures that users receive timely and context-appropriate assistance from personnel familiar with both the application and the local agricultural environment.

Maintenance Schedule

The system follows a structured maintenance schedule:

- **Regular Updates**: Monthly security patches and bug fixes
- **Quarterly Feature Updates**: New functionality and improvements
- **Annual Major Updates**: Significant enhancements and architectural changes

Scheduled Maintenance:

- Central servers: Weekly maintenance window (Sundays, 01:00-03:00 EAT)
- Database optimization: Monthly
- Model retraining: Quarterly with new image data
- Offline database updates: Distributed monthly

Documentation Management

All service and maintenance activities are documented in multiple formats:

- Electronic Records: Maintained in the project repository
- Change Logs: Detailed records of all modifications

- Knowledge Base: Accessible through the application

Future Updates

The system was designed to accommodate future updates through:

- Modular Architecture: Components can be updated independently
- Differential Updates: Only changed components are distributed
- Offline Update Packages: Updates can be distributed via physical media
- Phased Rollout: Updates are deployed progressively to allow for monitoring

The roadmap for future updates was developed in consultation with agricultural stakeholders to ensure alignment with evolving needs.

6.2 Performance and Maintenance.

The Fall Armyworm Detection System must meet specific performance requirements and follow established maintenance procedures to ensure reliable operation.

Performance Requirements

The system meets the following performance criteria:

Response Time:

- Image detection processing: Maximum 5 seconds on mid-range smartphones
- UI navigation: Maximum 1 second response time.
- Data synchronization: Maximum 2 minutes for daily updates

Availability:

- Mobile application: 99.5% availability on devices
- Central servers: 99.9% uptime.

Resource Utilization:

- Mobile application: Maximum 100MB storage
- Battery consumption: Maximum 50% battery for a full day of field use
- Data usage: Maximum 50MB per day
- Offline storage: Efficient compression for detection history

Maintenance Requirements

The maintenance of the system addresses several key areas:

Preventive Maintenance:

- Regular database optimization to maintain performance
- Proactive monitoring of system resources
- Scheduled backups of all critical data
- Periodic review of security measures

Adaptive Maintenance:

- Updates to accommodate changes in mobile platforms
- Adaptation to new pest patterns as Fall Armyworm evolves
- Adjustments based on user feedback
- Modifications to align with changing agricultural policies

Perfective Maintenance:

- Performance optimization for operation on new devices
- User interface improvements based on usability studies
- Enhancement of detection algorithms with new training data
- Expansion of knowledge base with new treatment options

Corrective Maintenance:

- Bug fixes prioritized based on impact
- Resolution of issues identified through user feedback
- Addressing compatibility issues with new device models
- Correction of any inaccuracies in treatment recommendations

Support Services

The system is backed by comprehensive support services:

Technical Support:

- Email-based support system
- In-person support during field visits
- Detailed troubleshooting guides

Training and Education:

- Initial training conducted for all users
- Refresher training available on demand
- Pictorial guides for users with limited literacy

Monitoring and Reporting:

- Centralized monitoring of system performance
- Usage analytics to identify improvement areas
- Monthly performance reviews

System Changes and Upgrades

Changes to the system are managed through a structured process:

Causes for Changes:

- Feedback from farmers and extension workers
- Evolution of Fall Armyworm behavior
- Changes in available treatments
- Improvements in mobile technology
- Updates to agricultural policies and programs

Change Management Process:

- Change requests collected through multiple channels
- Impact assessment for all proposed changes
- Approval workflow involving stakeholders
- Testing in controlled environments
- Phased deployment starting with pilot users

Upgrade Approach:

- Mobile application: Distribution through Google Play and direct APK
- Central servers: After-hours upgrades with backup systems
- Database: Schema migrations with backward compatibility
- Machine learning models: Parallel deployment for comparison

System Phase Out and Data Migration

When transitioning to new versions or replacement systems, the following approach is used:

Data Migration Strategy:

- Phased data migration to ensure integrity
- Prioritization of critical data
- Local data backup before migration
- Verification procedures after migration

Transition Approach:

- Extended parallel operation period
- Training programs before transition
- Gradual user migration
- Support for both systems throughout the transition

Legacy System Handling:

- Archival of legacy data
- Secure decommissioning
- Knowledge transfer sessions
- Documentation preservation

The phase-out process ensures continuity of service for users while facilitating the adoption of improved capabilities in new system versions.

Table 13: Performance and Maintenance Details

Topics	**Performance and maintenance**	**Date / Initials**
Problem / solution	Detection of system problems follows a structured approach: ● Automated monitoring identifies system-wide issues. ● User reports are collected through support channels. ● Issues are categorized by severity and impact. ● Temporary solutions are distributed when appropriate.	30/04/2025 / SA

	● Permanent fixes are developed and tested before deployment. Recent example: When testing we identified slow image processing on certain device models, a temporary solution reducing image resolution was implemented while a permanent fix optimizing the algorithm was developed and tested before deployment.	
Functional maintenance	The system requires functional maintenance in response to changing conditions: ● Agricultural standards: Updates to treatment recommendations based on regulatory changes ● Seasonal adjustments: Modifications to detection parameters based on seasonal variations. ● Network adaptations: Adjustments to synchronization protocols as connectivity options evolve. ● Interface updates: Refinements based on usability feedback. ● Treatment database: Updates based on availability of new products. Functional maintenance is scheduled quarterly, with critical updates distributed as needed.	30/04/2025 / GW, BL.

Functional expansion and performance improvement	The following enhancements have been identified for future implementation: ● Inclusion of a soil pH sensor integration to determine soil pH levels for comprehensive treatment recommendations that consider soil conditions. ● Integration with SMS services to provide alerts without requiring smartphone access, expanding reach to basic phone users. ● Development of voice-based interaction for users with limited literacy. ● Implementation of offline machine learning model updates to improve detection accuracy without requiring full application updates. These improvements are prioritized based on user feedback and technological feasibility.	1/05/2025 / SA, NM.

The performance and maintenance plan we developed ensures that the Fall Armyworm Detection System remains effective across diverse agricultural regions, adapting to infrastructure limitations while providing consistent service to farmers and extension workers. Regular review meetings with stakeholders allow for continuous refinement based on field experiences and changing conditions.

Our approach to system maintenance recognizes the critical nature of the application in agricultural pest management. By establishing clear procedures for updates, problem resolution, and performance monitoring, we've created a sustainable framework that will support the system throughout its lifecycle.

The phased approach to upgrades and eventual system phase-out demonstrates our commitment to ensuring continuity of service while embracing technological advancements. By planning for the entire system lifecycle from the beginning, we've created a robust solution that can evolve with changing agricultural needs and technological capabilities.

CHAPTER 7: CONCLUSION AND RECOMMENDATIONS

7.1 Conclusion

The Fall Armyworm (*Spodoptera frugiperda*) remains one of the most serious threats to maize production in Uganda and across Sub-Saharan Africa. Its ability to spread rapidly and destroy crops at various stages of growth has endangered food security and the livelihoods of smallholder farmers. In response to this growing challenge, this project set out to design and develop a practical, intelligent, and accessible system to assist farmers in the early detection and management of Fall Armyworm infestations.

The system developed provides a mobile-based platform powered by artificial intelligence, geolocation services, weather integration, and community support tools. It allows farmers to capture and submit maize leaf images, which are validated and classified into infestation stages using a MobileNet-based TensorFlow Lite model. The system also integrates GPS-based location tracking to map the spread of the pest at the district level and connects with weather APIs to guide pesticide application decisions. All detection results are logged and accessible through a simple user interface, along with treatment recommendations tailored to each stage of infestation.

The backend, developed using Flask and connected to a SQL database, efficiently handles user requests, runs the AI model inference, stores data securely, and generates reports. A Firebase integration enables user authentication and cloud storage. Additionally, a community support module enables farmers to interact with experts, creating an inclusive and knowledge-driven environment.

Overall, the system has met its core objectives of improving pest detection accuracy, offering practical treatment advice, and giving farmers a data-informed way to respond to infestations. It also lays the groundwork for agricultural authorities and policymakers to collect field-level data in real-time, helping with broader pest control planning.

7.2 Recommendations

While the Fall Armyworm Detection System has proven functional and impactful during its development and testing phases, there are several recommendations for further improvement, scaling, and deployment:

1. **Expand Model Training with Regional Data**

   The accuracy of AI predictions depends heavily on the diversity and quality of training data. We recommend collecting more annotated images from different regions, lighting conditions, and infestation patterns to improve model generalization across all Ugandan districts and eventually beyond.

2. **Add Multi-language Support**

   Since many rural farmers speak local languages, incorporating translations into Luganda, Runyankore, Luo, and other regional languages would improve accessibility and usability of the system.

3. **Offline Functionality**

   To support users in remote areas with limited internet access, future versions of the app could include offline mode features:

- Local detection using a compressed version of the TFLite model.
- Temporary local data storage and automatic syncing once connected to the internet.

4. **Integration with National Agricultural Systems**

   The system can be linked to platforms managed by the Ministry of Agriculture or local government offices. This would allow district agricultural officers to monitor pest outbreaks in real-time and coordinate responses more effectively.

5. **Broaden Scope to Other Pests and Crops**

   With minor adaptations, the platform can be expanded to detect other common crop pests or diseases such as maize streak virus, leaf blight, or pests in beans, cassava, and bananas.

6. **Improve the Expert Review System**

The community module could benefit from:

- Expert verification tags
- Response ratings by farmers
- A notification system to alert experts of new posts

7. **Include Voice Input and Audio Feedback**

   This would be particularly helpful for users with low literacy levels. Adding voice input for community questions and audio playback for treatment instructions can improve engagement and effectiveness.

8. **Partner with Agrochemical Providers**
   Collaborations with pesticide suppliers could ensure accurate, updated chemical information and potentially offer farmers access to discounted or recommended products directly through the app.

7.3 Final Thoughts

The Fall Armyworm Detection System represents a significant step toward using intelligent systems to solve real-world agricultural problems. It bridges the gap between modern AI capabilities and the urgent needs of rural farmers. While there is still room for enhancement, the system has demonstrated that with the right tools and design, technology can empower even the most underserved communities to make smarter, more informed decisions.

We believe that this project has laid a strong foundation for further innovation and that, with continued development and stakeholder involvement, it can make a lasting impact on Uganda's agricultural resilience.

CHAPTER 8: USER MANUAL

This chapter provides a detailed guide on how to use the Fall Armyworm Detection. The system is built as a mobile application supported by a Flask-based backend API, providing farmers with real-time pest diagnosis, treatment advice, and analytical insights.

The system has been designed for ease of use, particularly for smallholder farmers with minimal technological background. This manual walks through each major feature of the application, how it works, and how the user can interact with it effectively. All features are presented in a logical flow, based on typical user activity.

8.1 Application Launch and Home Screen

Once the application is installed and launched:

- The **Home Screen** appears by default and the app asks user's permissions to access the resources like camera and the location.
- The screen includes:
- A **real-time weather display** for the user's district (requires location permission).
- A **button to upload an image**.
- A **button to scan a plant in real time** using the camera.
- A **bottom navigation bar** for accessing other parts of the app: Home, Community, Coverage, and Analytics

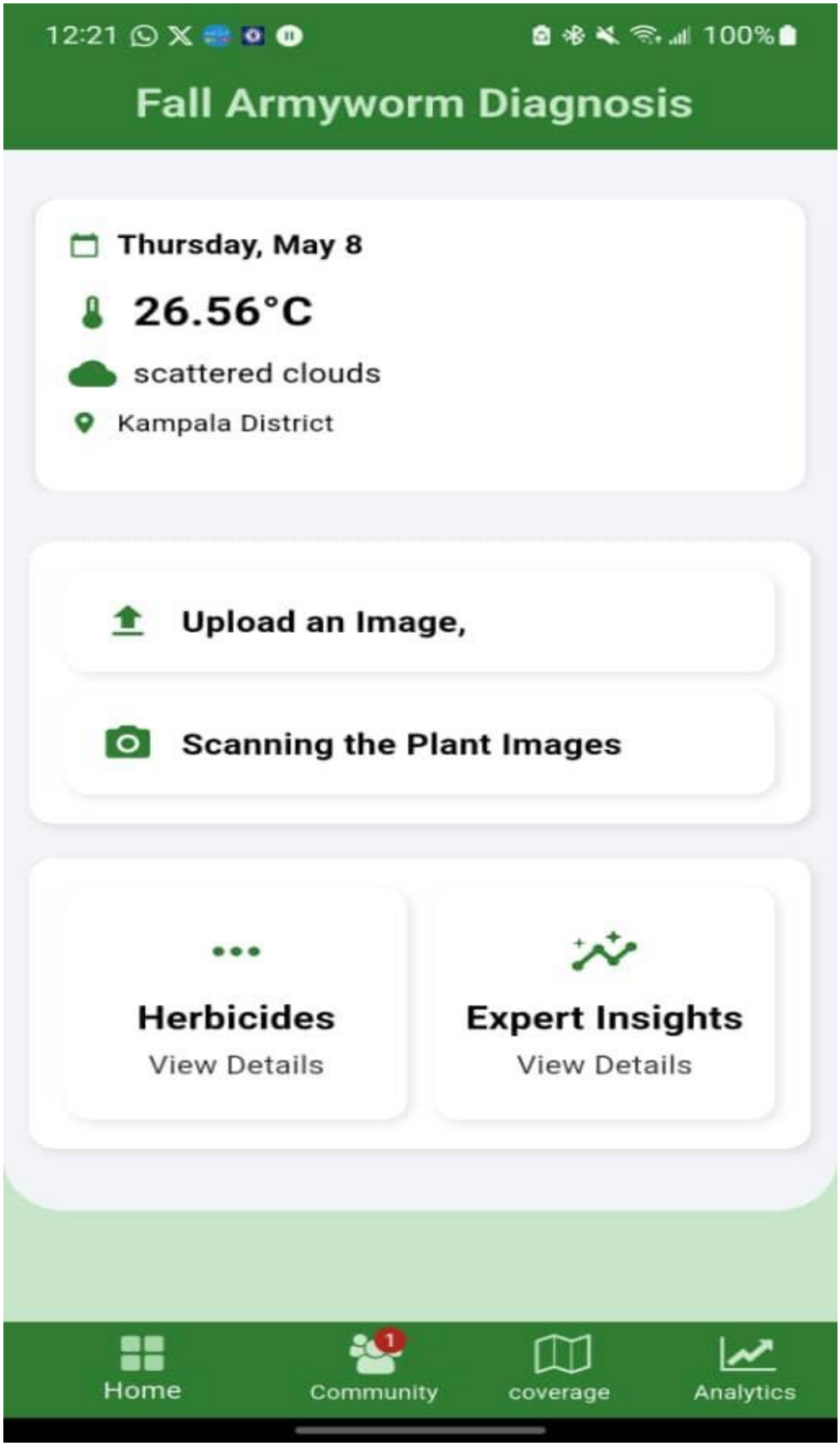

12:21
100%
Fall Armyworm Diagnosis
Thursday, May 8
26.56°C
scattered clouds
Kampala District
Upload an Image,
Scanning the Plant Images
Herbicides
View Details
Expert Insights
View Details
Home
Community
coverage
Analytics

Figure 1: Home screen with weather, upload button, scan button, and bottom navigation bar.

Note: Users are prompted to enable location access before weather and map-based features work.

8.2 Uploading an Image

To diagnose a maize leaf using a saved or captured image.

1. Tap **“Upload an Image”** on the home screen.

2. The **camera preview** interface opens.

3. Below the preview are three icons:

○ **Gallery**: Select an image from device storage.

○ **Camera**: Capture a new photo using the device’s camera.

○ **Tips**: Displays guidance for capturing clear and valid maize leaf images.

○ If it is the user's **first time**, the tips are automatically shown. Users can choose to **hide tips permanently** after viewing.

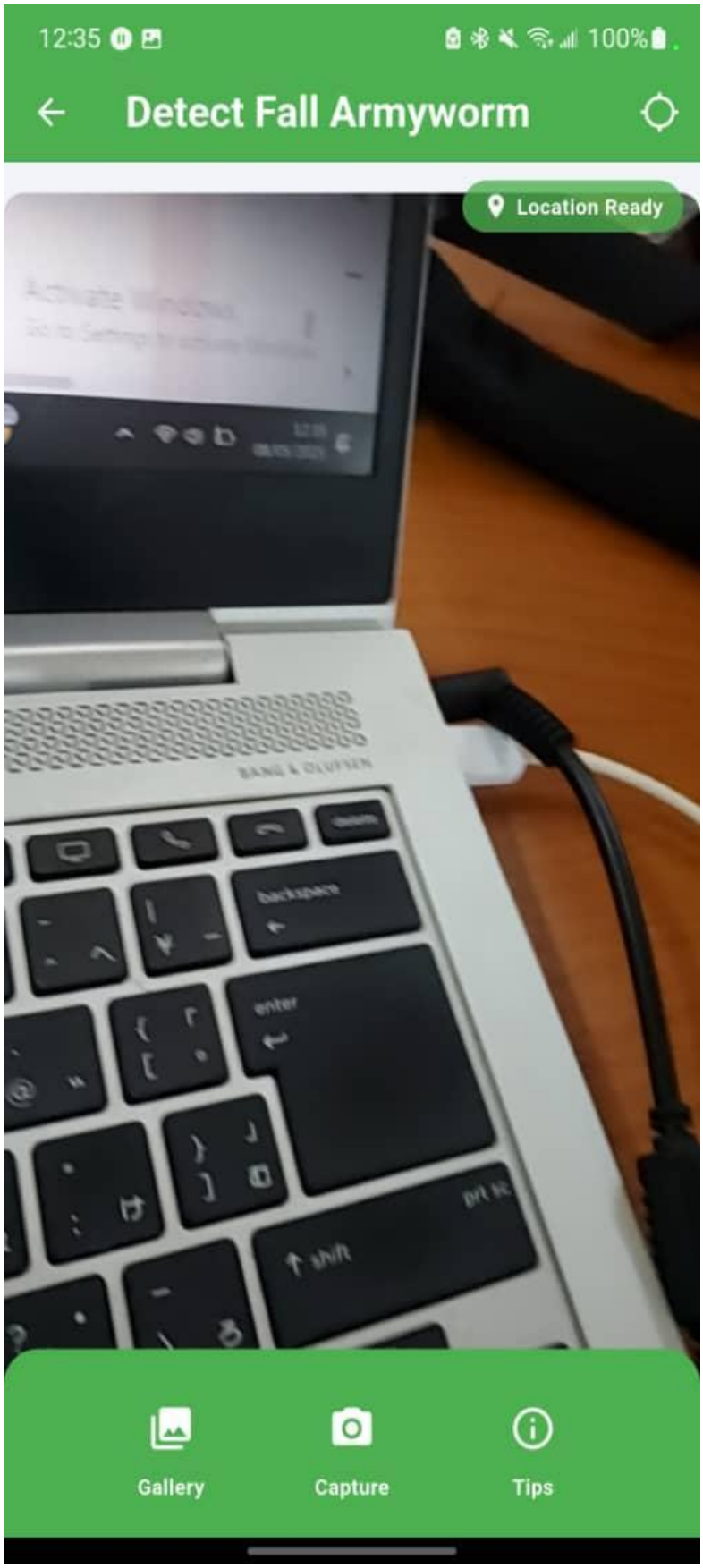


Figure 2: Camera preview with gallery, camera, and tips icons and Real time Camera preview

4. Once an image is selected or captured, the screen displays **"Analyzing..."** as the system:

- Validates the image using the **pre-screening model** to check if it is a maize leaf.
- If not valid, a message like **"Please upload a clear maize leaf image"** is shown.
- If valid, it proceeds to **stage detection** using the MobileNet-powered classifier.

8.3 Viewing Detection Results

If the image is valid, the system detects the stage of the infestation and displays:

- The **predicted stage**: e.g., "Frass", "Eggs", or "Larval Damage"
- A **confidence score**
- A description of the **symptoms** observed in the detected stage
- Detailed **treatment recommendations**

Below the results, users have two main actions:

- **Download Detection Report**: A PDF report is generated with all detection data (image, location, timestamp, result, and treatment).
- **View Herbicides and Application Guide**: This opens a page listing:
- Recommended chemicals
- Mixing instructions
- Safe application tips

- When and how to spray (based on the weather)

Figure 3: Detection results screen with stage, confidence, recommendations, and download button.

8.4 Real-Time Scanning

To use the real-time scanning feature:

1. Tap **“Scan a Plant”** on the home screen.
2. The camera opens in **live detection mode**.
3. Before scanning starts, the app **checks if location access is enabled**.
    - If not, it prompts the user to allow GPS access and then the start detection button is enabled.
4. When a maize leaf enters the frame, detection begins automatically.

5. A real-time result is shown once detection is complete.

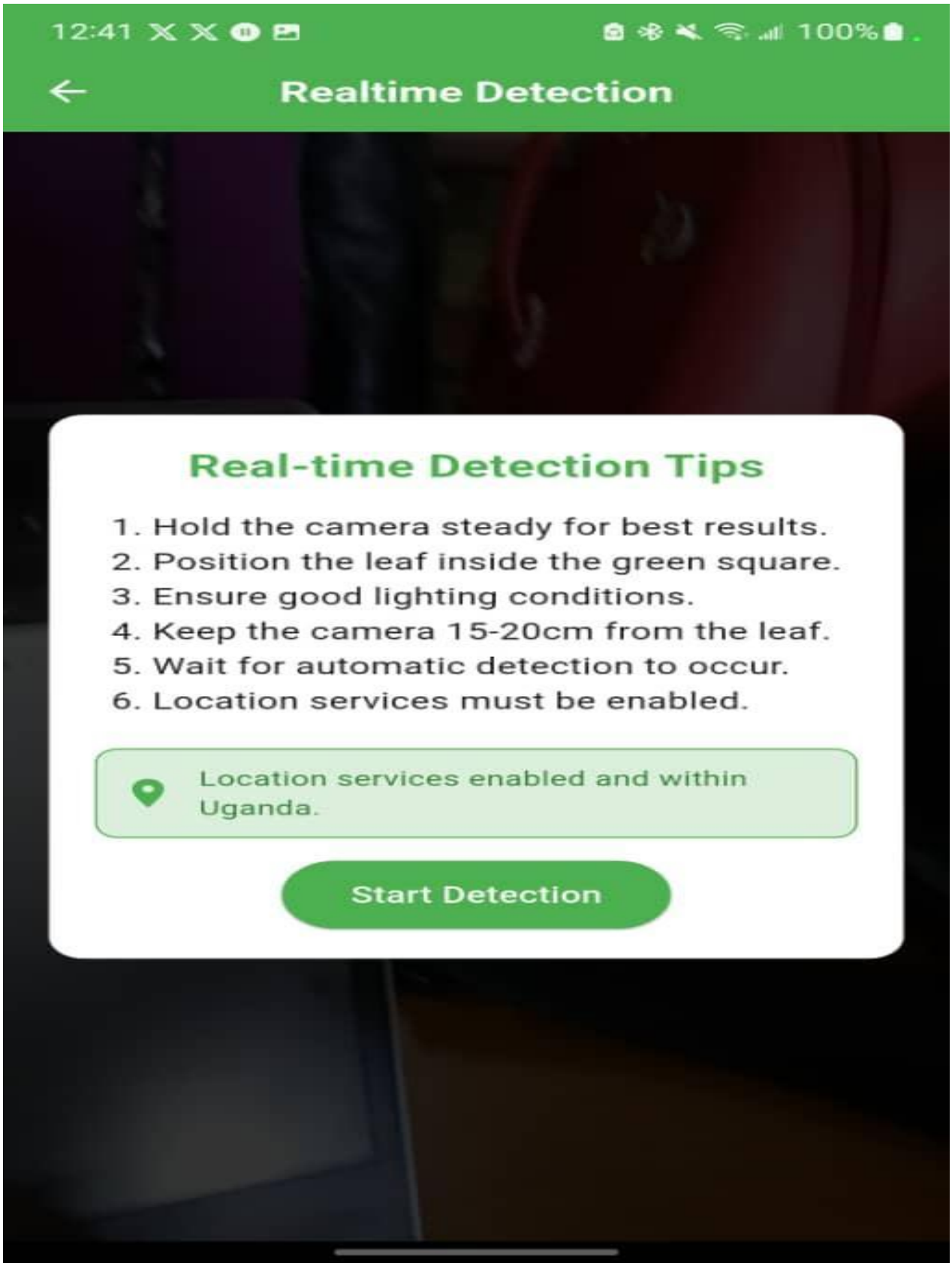


Figure 4: Scan mode with live camera feed and detection box overlay.

8.5 Community Support

To access expert advice or share experiences:

1. Navigate to the **Community tab** via the bottom navigation bar.
2. To post or comment, the user must **register or sign in**.
   - Authentication is handled through Firebase.
   - Registered users can create a profile and submit questions.
3. Farmers can post questions and upload supporting images.

4. **Recommended experts** can respond with detailed advice.

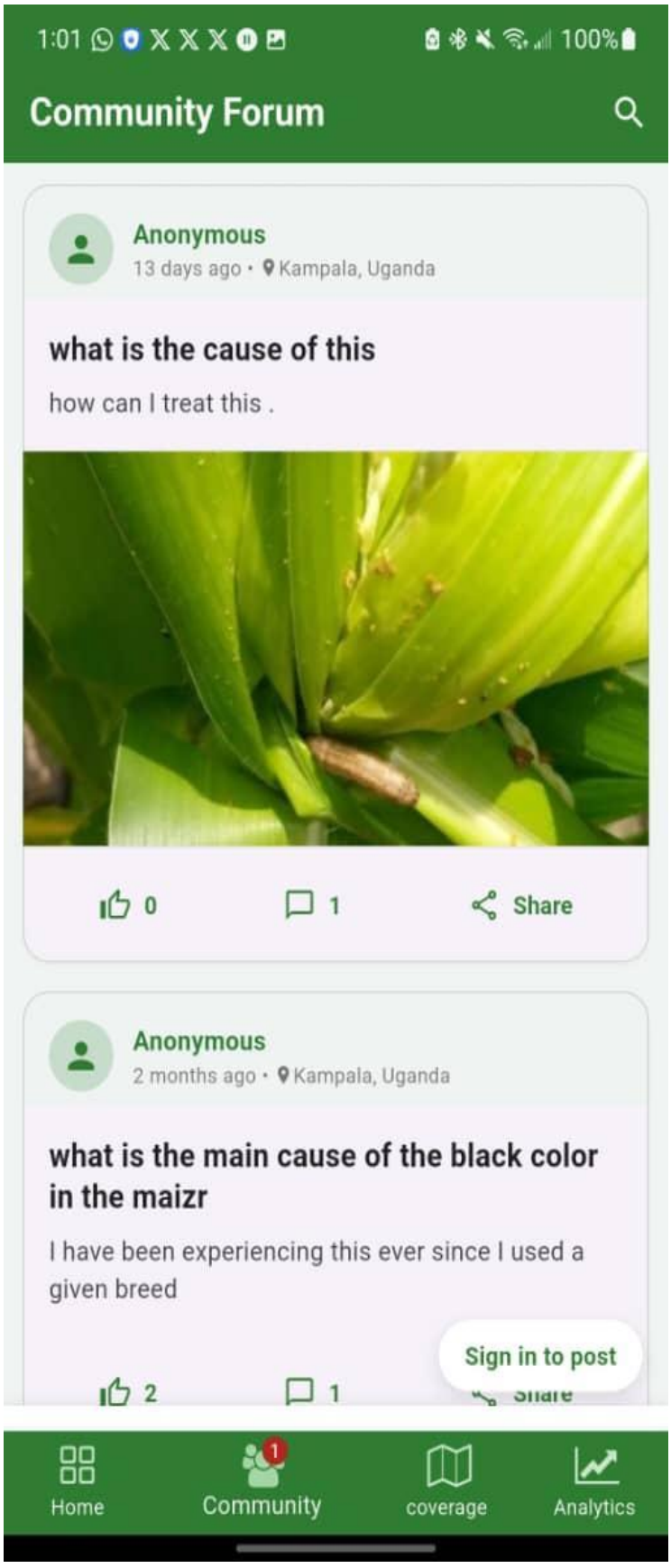


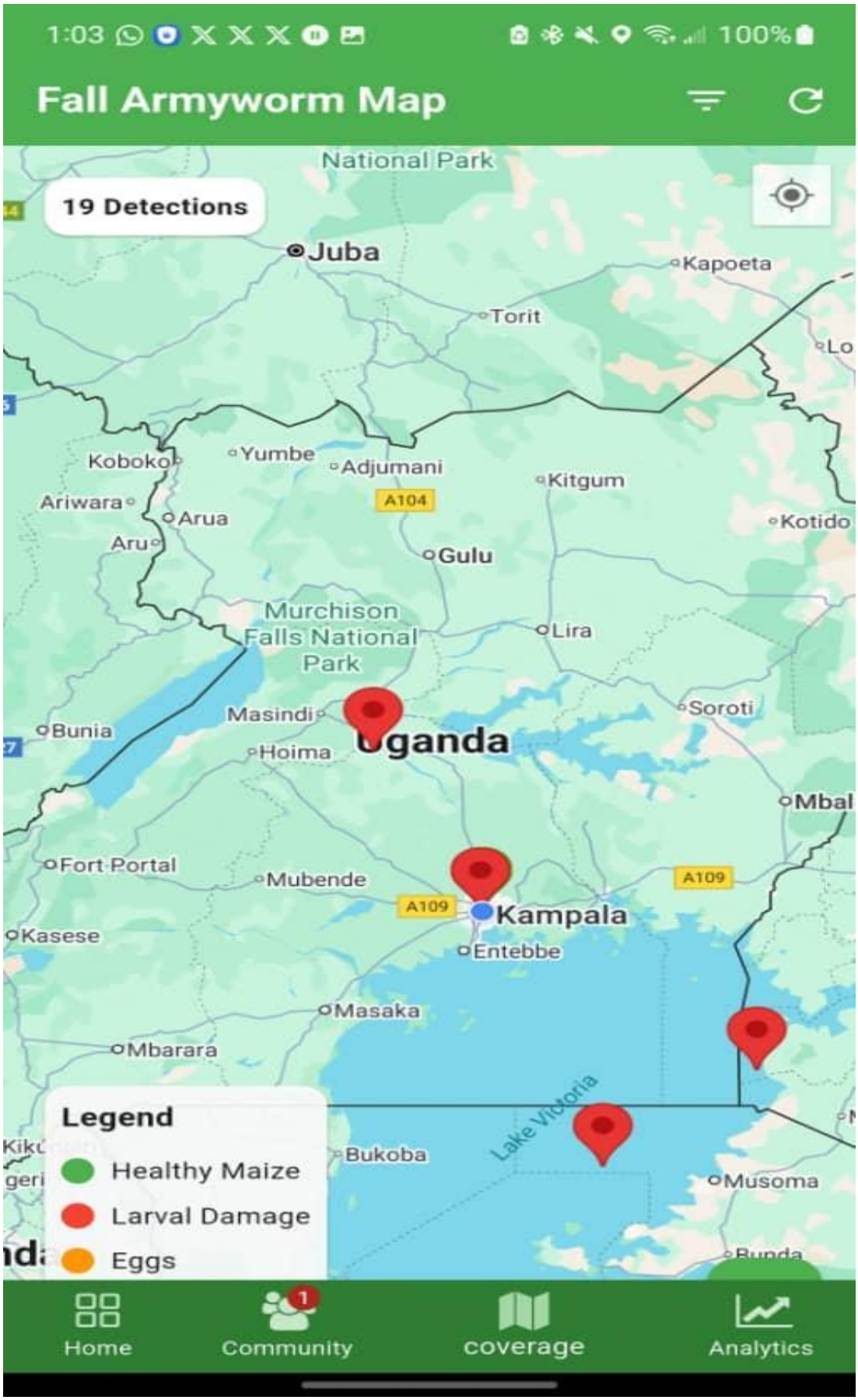
Figure 5: Community feed showing user questions and expert replies.

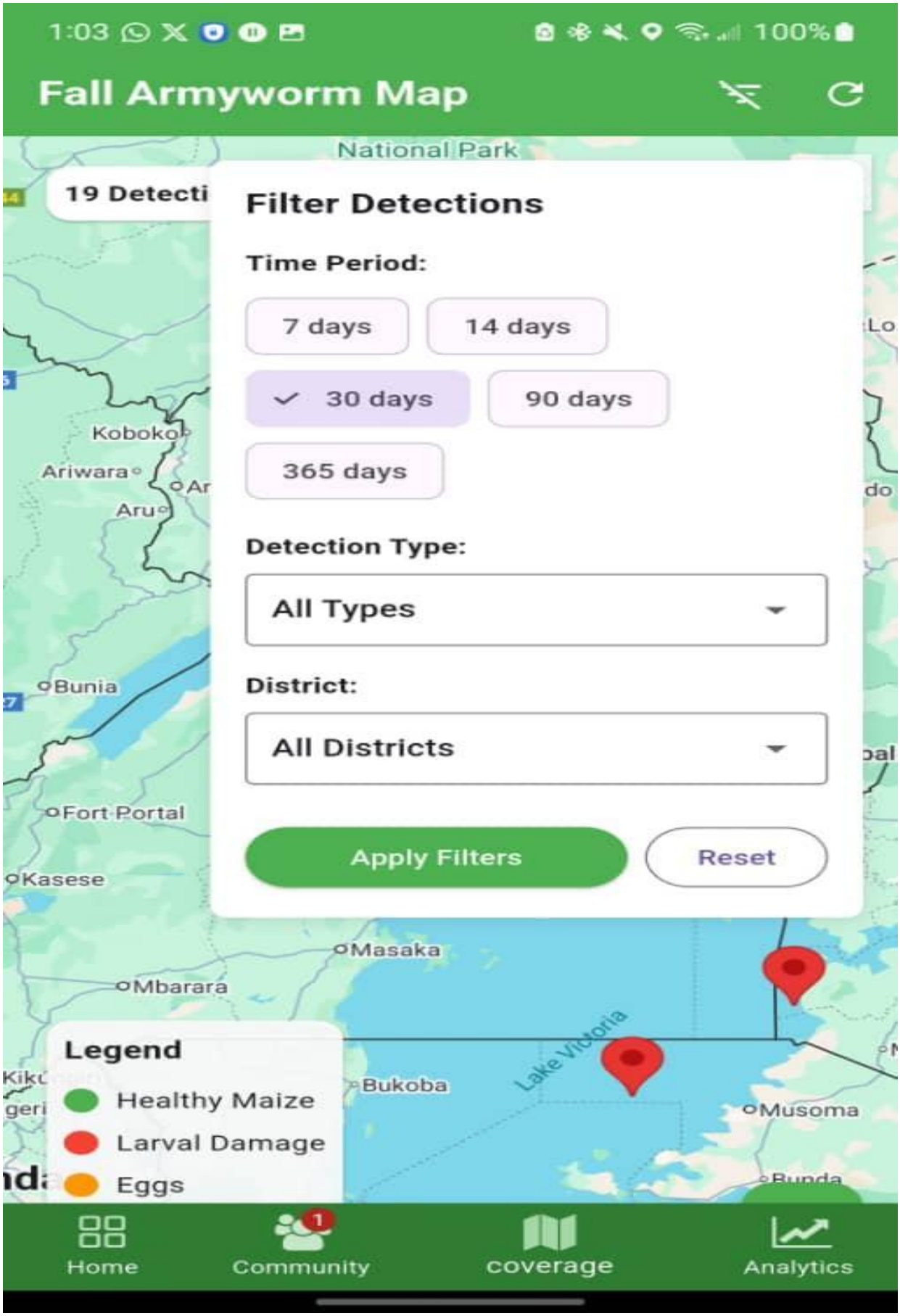
8.6 Coverage (Detection Map)

This feature visualizes how Fall Armyworm is spreading across Uganda:

- The **map displays detection points** using color-coded markers:
  - Green: Healthy
  - Yellow: Eggs
  - Orange: Frass
  - Red: Larval Damage
- Tapping a marker shows:
  - Image submitted
  - Detection stage
  - District and timestamp

Users can **filter by date, region, or stage** to focus on specific outbreaks.




1:03
100%
Fall Armyworm Map
National Park
19 Detecti
Filter Detections
Time Period:
7 days
14 days
30 days
90 days
365 days
Detection Type:
All Types
District:
All Districts
Apply Filters
Reset
Kobokol
Ariwara
Aru
Bunia
Fort Portal
Kasese
Masaka
Mbarara
Bukoba
Lake Victoria
Musoma
Legend
Healthy Maize
Larval Damage
Eggs
Home
Community
coverage
Analytics

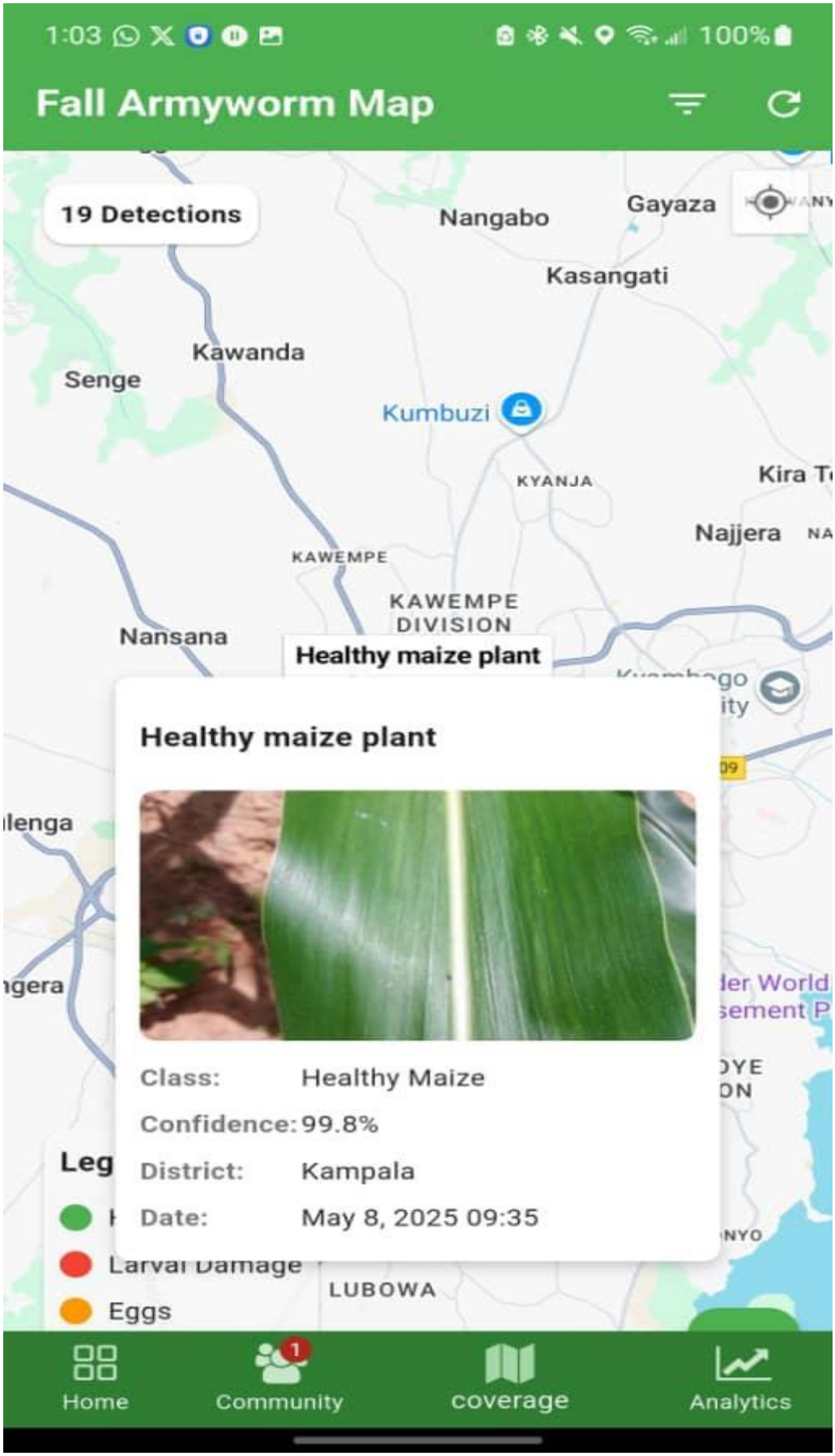


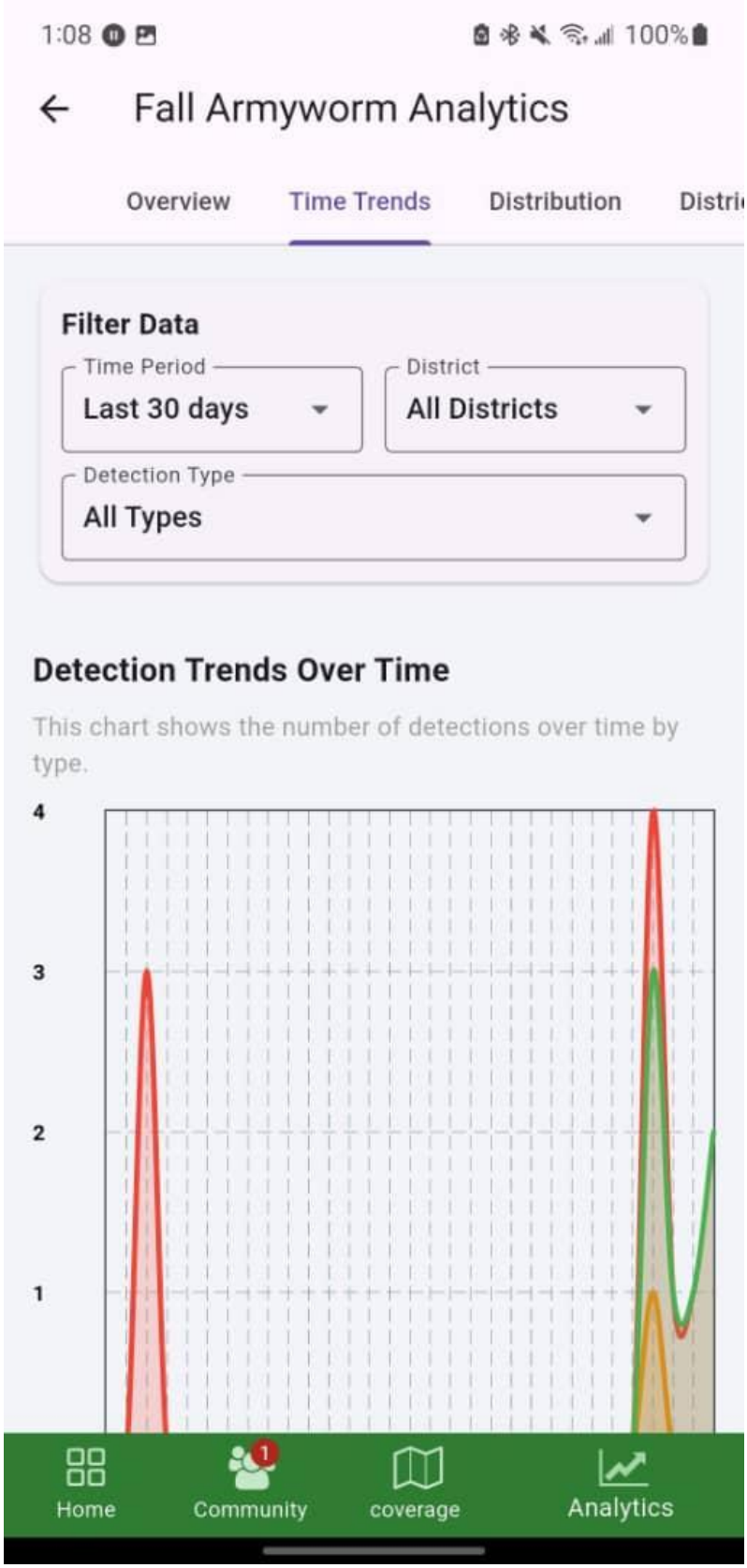
Figure 6: Map showing color-coded detection pins

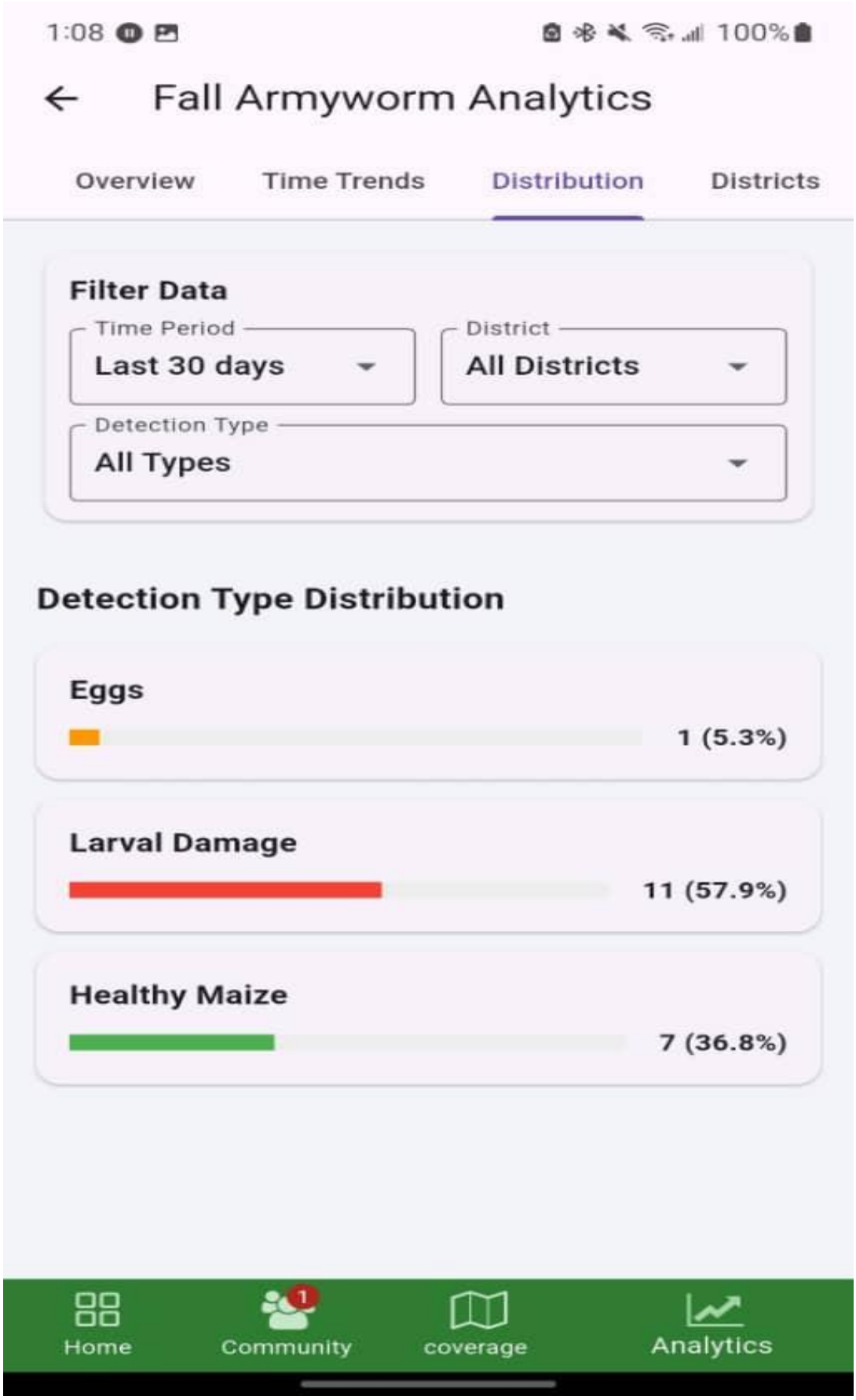
8.7 Analytics

This tab provides visual data insights to the user:

- **Detection type charts**: Pie or bar charts showing percentage of each stage detected.
- **Time trends**: Line graphs showing detections over time (daily, weekly).
- **District distribution**: Visualization of most affected areas.

- **Model accuracy**: For internal evaluation, users can view performance metrics.




1:08
100%
Fall Armyworm Analytics
Overview
Time Trends
Distribution
Districts
Filter Data
Time Period
Last 30 days
District
All Districts
Detection Type
All Types
Detection Type Distribution
Eggs
1 (5.3%)
Larval Damage
11 (57.9%)
Healthy Maize
7 (36.8%)
Home
Community
coverage
Analytics

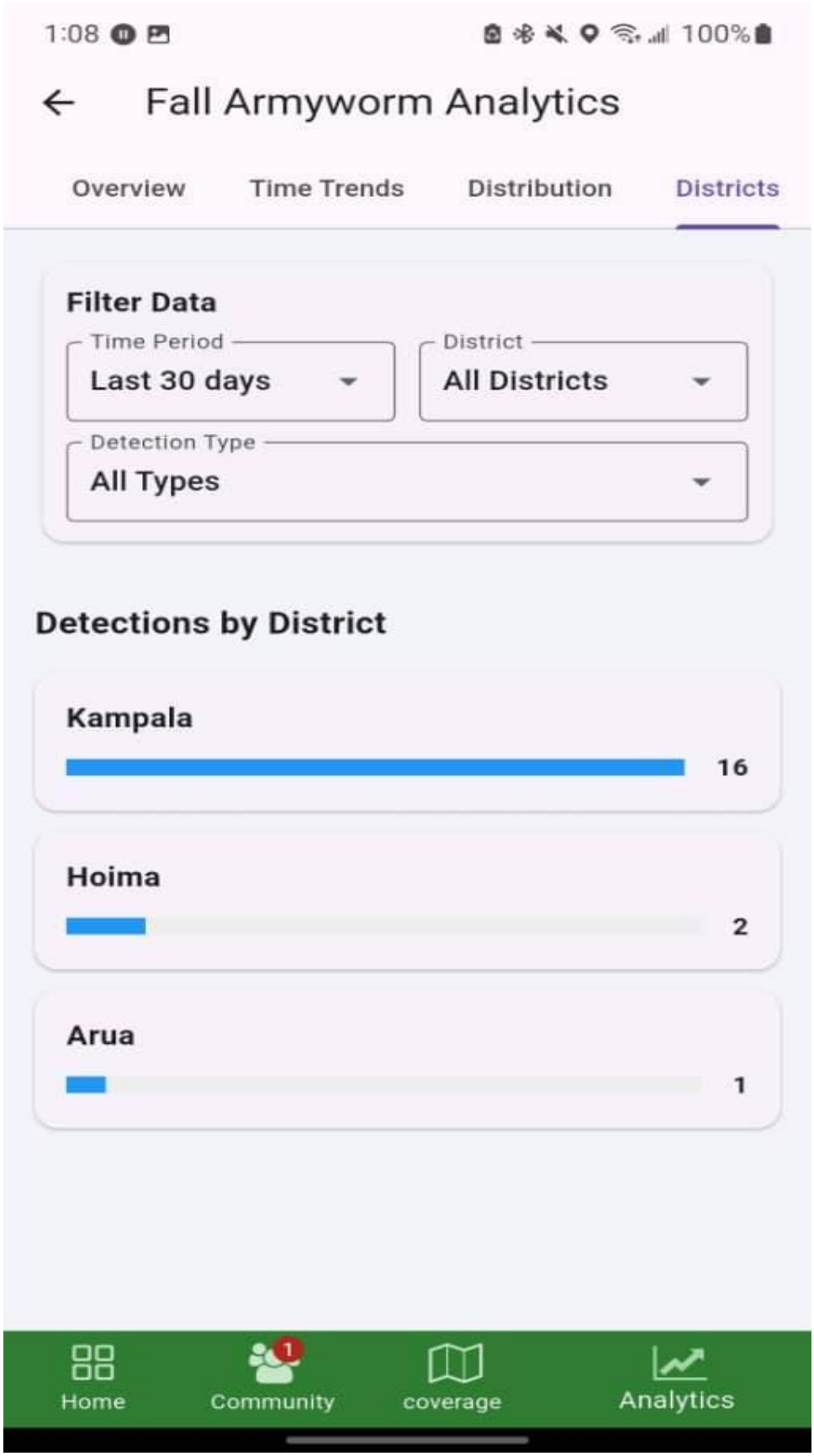


Figure 7: Analytics dashboard showing time trends and stage distribution.

8.8 Help and Support

If users encounter any issues or need help:

- Visit the **Profile/Settings page** and tap **"Help"**
- FAQs are listed for:
  - How to capture a good image
  - Why location is required
  - What to do if the system cannot detect the leaf
- A **contact form** or **support email** is provided for reporting bugs or feedback.

8.9 Summary

The Fall Armyworm Detection is designed to be easy to use, even in rural settings. From capturing images and receiving real-time analysis to getting pesticide advice and connecting with experts, each feature is built to empower farmers with timely, accurate, and practical information.

The screenshots provided illustrate the logical flow of the app, helping new users to understand and navigate the system effectively.

Final approval for use	
Identification:	
Responsible for validation:	
Remarks:	
Date:	Signature: