Project Report

on

Smart Agriculture Health Monitoring System

Submitted in partial fulfillment of the requirements

for the award of the degree of

Bachelor of Technology

in

Computer Science & Technology

by

Kritika Baghel

(2K21CSUN01127)

Under the Supervision of

(Mr. Anup Singh Kushwaha)

Assistant Professor

MANAV RACHNA Vidyavatarkshal

MANAV RACHNA UNIVERSITY

Declared as State Private University Vide Haryana UGC Act 26 of 2014

Manav Rachna University, Faridabad

(June, 2025)

Acknowledgement

I would like to express my sincere gratitude to everyone who has contributed to the successful completion of my project, **"Smart Agriculture Crop Health Monitoring System".**

First and foremost, I am deeply thankful to my supervisor, **Mr. Anup Singh** for their invaluable guidance, continuous support, and insightful suggestions throughout the project. Their expertise and encouragement have been instrumental in shaping this work.

I also extend my appreciation to **Manav Rachna University** for providing the necessary resources and a conducive learning environment to carry out this research effectively.

A special thanks to my family and friends for their constant motivation and unwavering support, which kept me motivated during challenging times.

Lastly, I acknowledge the contributions of the research community, open-source platforms, and Kaggle for providing access to valuable datasets, which played a crucial role in the development of this project.

This project has been a great learning experience, and I am grateful to everyone who has been part of this journey.

Kritika Baghel

Abstract

The Smart Agriculture Crop Health Monitoring project offers an AI-powered solution to identify and classify crop health issues using images captured through drones or smartphones. This system utilizes Azure Custom Vision for image classification and Azure Blob Storage for secure and scalable image storage. It is designed to detect common crop health conditions, including fungal, bacterial, and viral diseases, as well as pest infestations and healthy crops.

The AI model is trained on a comprehensive dataset sourced from Kaggle, which includes images of major crops such as wheat, rice, tomato, potato, and maize. Once an image is uploaded, the system processes it and provides real-time classification results that are displayed through an interactive and user-friendly dashboard. This approach reduces the dependency on manual field inspections, allowing for quicker detection and response to crop health problems. It also helps farmers make timely and informed decisions, improving crop management while reducing unnecessary chemical use. By integrating AI and cloud technologies, the project supports sustainable farming practices, enhances productivity, and contributes to food security.

Ultimately, this system bridges the gap between traditional agricultural methods and modern technological advancements, empowering farmers with actionable insights to increase yield quality and farm efficiency.

Table of Contents

List of Figures

Chapter 1

Introduction

Agriculture plays a crucial role in ensuring food security and economic stability. However, challenges such as pest infestations, nutrient deficiencies, and plant diseases significantly impact crop yield and quality. To address these challenges, modern technology—particularly Artificial Intelligence (AI)—is being leveraged to enhance agricultural productivity and sustainability.

The **Smart Agriculture Crop Health Monitoring** project aims to develop an AI-based system that classifies crop health using images captured by drones or smartphones. This system categorizes crops into specific health conditions such as **Healthy**, **Fungal Diseases**, **Bacterial Diseases**, **Viral Diseases**, and **Pest-Infested**, enabling farmers to take timely corrective measures. By integrating **Azure Cognitive Services (Custom Vision)** and **Azure Blob Storage**, the project offers a scalable and efficient solution for real-time crop health analysis.

The model is trained on a diverse image dataset sourced from **Kaggle**, covering common crop conditions in **wheat, tomato, rice, potato**, and **maize**. Cloud-based services ensure smooth image storage, classification, and result visualization through an interactive dashboard, making the system highly accessible and practical for farmers. This also reduces reliance on manual inspection and promotes faster decision-making in disease and pest management.

By learning health patterns across multiple crops, the AI model enhances precision in detection, reduces chemical usage, and supports sustainable farming practices. The overall goal is to empower farmers with timely insights that increase yield quality, minimize losses, and promote efficient farm management.

This initiative contributes to the evolution of **smart agriculture**, bridging traditional practices with modern AI capabilities and laying the foundation for future innovations in data-driven, eco-friendly farming.

1.1 Problem Statement

Traditional crop health monitoring methods are slow, labor-intensive, and prone to human error, often resulting in delayed detection of diseases and pest infestations. Farmers lack timely diagnostic tools, leading to reduced yields and excessive chemical use. The proposed system leverages AI and cloud technologies to automatically classify crop health from images, enabling early detection and real-time decision-making. This solution supports sustainable farming by improving accuracy, reducing manual effort, and enhancing productivity.

1.2 Objective(s) of the Proposed System

The main objective of the proposed **Smart Agriculture Crop Health Monitoring System** is to leverage Artificial Intelligence (AI) and cloud technologies to detect and classify crop health conditions using images. The system aims to assist farmers in identifying plant diseases and pest infestations at an early stage to enable timely and informed decisions.

Key objectives include:

- To develop an AI-based model capable of classifying crop health using images captured by drones or smartphones.
- To accurately detect and differentiate between Healthy crops, Fungal diseases, Bacterial diseases, Viral diseases, and Pest-infected plants.
- To integrate cloud services (such as Azure Custom Vision and Azure Blob Storage) for image classification and secure storage.
- To reduce dependency on manual inspection and promote automation in disease detection.
- To provide an interactive dashboard for real-time visualization of results and insights.
- To support sustainable farming by minimizing unnecessary pesticide use through early disease detection.
- To improve crop yield quality and promote efficient farm management using AI-based decision support.

1.3 Present System Description

In the current agricultural scenario, most farmers rely on **manual inspection** and **traditional farming practices** to monitor crop health. Visual observation is the primary method used to detect symptoms of diseases, pests, or nutrient deficiencies. This approach is often **time-consuming**, **inaccurate**, and **dependent on the farmer's experience and knowledge**.

The present system has several limitations:

- **Lack of early detection:** Diseases and pest infestations are often noticed at later stages when visible damage has occurred.
- **Manual labor required:** Frequent field visits and close inspection are necessary, increasing workload and costs.
- **Limited accuracy:** Identifying specific types of diseases or pests without expert knowledge is difficult and may lead to incorrect treatment.
- **No centralized data storage or analysis:** Observations are not documented or analyzed, making it hard to track patterns or make data-driven decisions.
- **Dependence on agro-experts:** Farmers often need to consult agricultural experts or visit local centers, causing delays in action.

As a result, the traditional system leads to **reduced crop yields**, **higher chemical use**, and **increased losses** due to delayed or improper treatment. This highlights the need for a smarter, faster, and more accurate crop health monitoring system.

1.4 Problem Definition of the Proposed System

Agricultural productivity is heavily impacted by factors such as **plant diseases**, **pest infestations**, and **nutrient deficiencies**, which often go undetected until significant damage has occurred. Traditional crop monitoring methods rely on manual inspection, which is **time-consuming**, **labor-intensive**, and **prone to human error**. Farmers often lack access to expert guidance and advanced diagnostic tools, leading to **delayed or incorrect treatment**, lower yields, and increased chemical usage.

There is a critical need for an **automated, intelligent, and scalable system** that can quickly and accurately detect crop health issues at an early stage. Such a system should be able to:

- Analyze crop images captured using smartphones or drones.
- Classify crop health conditions (e.g., healthy, fungal, bacterial, viral diseases, and pest attacks).
- Provide real-time results to enable timely decision-making.
- Reduce the dependency on expert visits and manual diagnosis.
- Store and manage image data securely for future reference and analysis.

The proposed system addresses these challenges by integrating **Artificial Intelligence (AI)** and **cloud services** to build a smart crop health monitoring platform. It aims to enhance precision agriculture, improve crop management practices, and support sustainable farming.

1.5 Hardware and Software Requirements

To develop and run the Smart Agriculture Crop Health Monitoring System, the following hardware and software components are required:

Hardware Requirements

- **Processor:** Intel i5 or higher
- **RAM:** Minimum 8 GB
- **Storage:** Minimum 250 GB (SSD recommended)
- **Camera/Drone:** For capturing crop images (Smartphone or drone camera)
- **Internet Connection:** Stable connection for cloud services and dashboard access

Software Requirements

- **Operating System:** Windows 10/11 or Ubuntu/Linux
- **Programming Language:** Python
- **IDE/Code Editor:** Visual Studio Code (VS Code)
- **Web Technologies:** HTML, CSS, JavaScript
- **Backend Framework:** Flask (Python-based)
- **Cloud Services:**
    - Azure Custom Vision (for image classification)
    - Azure Blob Storage (for image storage)
- **Libraries and Tools:**
    - OpenCV (for image processing)
    - NumPy, Pandas (data handling)
    - Matplotlib/Seaborn (for optional visual analysis)
    - Requests/Flask-CORS (API and frontend integration)
- **Browser:** Google Chrome / Firefox (for accessing the dashboard)

Chapter 2

Requirements Elicitation and Analysis (SRS)

2.1 Introduction

This chapter provides a comprehensive overview of the Software Requirements Specification (SRS) for the proposed project, **Smart Agriculture Crop Health Monitoring System**. The system aims to support farmers in identifying and diagnosing crop diseases and pest infections using Artificial Intelligence (AI) and cloud-based services.

Agriculture in India and globally faces serious challenges from plant diseases and pests, often leading to low productivity and significant financial losses. Manual methods of disease identification are prone to error and inefficiency, especially in rural areas where expert access is limited. Hence, there is a need for a smart, scalable, and efficient solution that uses advanced technologies like AI to help farmers diagnose crop health problems early and accurately.

This document describes the purpose and scope of the system, identifies the intended users, outlines the system's functional and non-functional requirements, and lists the technologies involved. It serves as a blueprint for the design, development, and deployment of the system.

2.1.1 Purpose

The purpose of this system is to provide an AI-powered platform for monitoring crop health by analyzing images of crops. The platform aims to bridge the gap between modern agricultural technology and traditional farming practices. It empowers farmers by providing instant feedback on the health status of their crops, thereby helping them take corrective actions early and minimize crop damage.

The key objectives served by this system include:

- Helping farmers diagnose crop health issues early and accurately.
- Reducing dependence on manual labor and agro-experts for disease detection.
- Encouraging environmentally responsible farming by minimizing chemical use.
- Promoting smart and sustainable agricultural practices using digital tools.
- Making modern AI solutions accessible to rural and small-scale farmers.

2.1.2 Scope

The scope of the **Smart Agriculture Crop Health Monitoring System** includes the development and deployment of a web-based application that uses AI models to classify crop health conditions.

The major features covered in this system are:

- **Image Upload Module**: Users can upload crop images taken via mobile phones or drones through a web interface.
- **Health Classification Engine**: An AI model trained using Azure Custom Vision processes the image and classifies it as:
    - Healthy
    - Fungal Disease
    - Bacterial Disease
    - Viral Disease
    - Pest-Infested
- **Cloud Storage Integration**: All uploaded images and predictions are stored in Azure Blob Storage for future reference and analysis.
- **Real-Time Response**: Once the image is processed, the classification result is displayed instantly to the user via a web dashboard.
- **Crops Supported**: Wheat, rice, tomato, potato, and maize, with scope for future expansion to more crops.
- **Dashboard**: A user-friendly, visually interactive dashboard that allows users to view current results and track history.

The system is accessible via desktop and mobile browsers, ensuring flexibility and ease of use.

2.1.3 Technologies to be Used

The project incorporates a variety of technologies for frontend, backend, AI integration, and cloud-based storage.

Frontend Technologies

- **HTML5**: To create the structure of the web pages.
- **CSS3**: For styling the UI elements to make them visually appealing and responsive.
- **JavaScript**: To add interactivity and handle frontend logic.
- **Bootstrap**: (Optional) For creating responsive, mobile-friendly layouts quickly.

Backend Technologies

- **Python**: The primary language for backend development.
- **Flask**: A lightweight Python web framework used to build REST APIs and serve the application.
- **REST APIs**: Used to handle communication between the frontend, backend, and Azure services.

AI and Cloud Services

- **Azure Custom Vision**: For building, training, and deploying the image classification model.
- **Azure Blob Storage**: For uploading and storing crop images in a scalable and secure environment.

Development Tools

- **Visual Studio Code (VS Code)**: The primary IDE used for writing and debugging code.
- **Postman**: For testing API endpoints and data exchange between modules.
- **GitHub**: For version control and collaboration.
- **Kaggle Dataset**: Used to train the AI model with real crop disease images.

Chapter 3

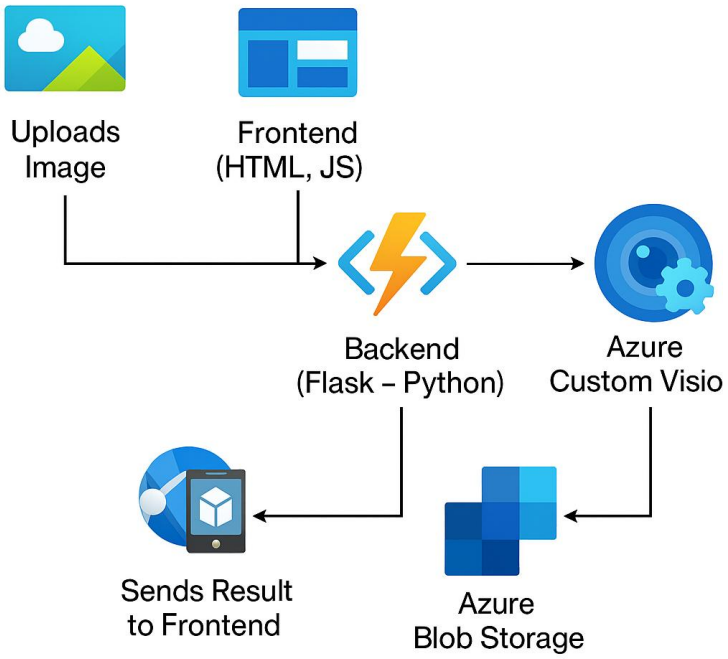
Design Specification

This chapter focuses on the architectural and structural design aspects of the **Smart Agriculture Crop Health Monitoring System**. It includes various diagrams such as architecture design, data flow, class diagrams, use cases, and activity diagrams that help visualize the system's structure and behavior. It also covers the implementation plan using PERT and Gantt charts and shows a preview of input and output screens.

3.1 Architecture Design

The system follows a three-tier architecture:

- **Frontend** (HTML, CSS, JS): Allows users to upload crop images and view results.
- **Backend** (Flask - Python): Processes image uploads, interacts with cloud services, and returns predictions.
- **Cloud Layer**:
    - **Azure Blob Storage** stores images.
    - **Azure Custom Vision** analyzes and classifies crop health.



3.2 Data Flow Diagram

A Data Flow Diagram (DFD) is a visual representation used to map out the flow of information in a system. It helps in understanding how data is input, processed, stored, and output within a system. DFDs are particularly useful during the design and analysis phases of a project, as they allow developers and stakeholders to visualize the movement of data without getting into the complexities of program logic.

The DFD is typically divided into multiple levels:

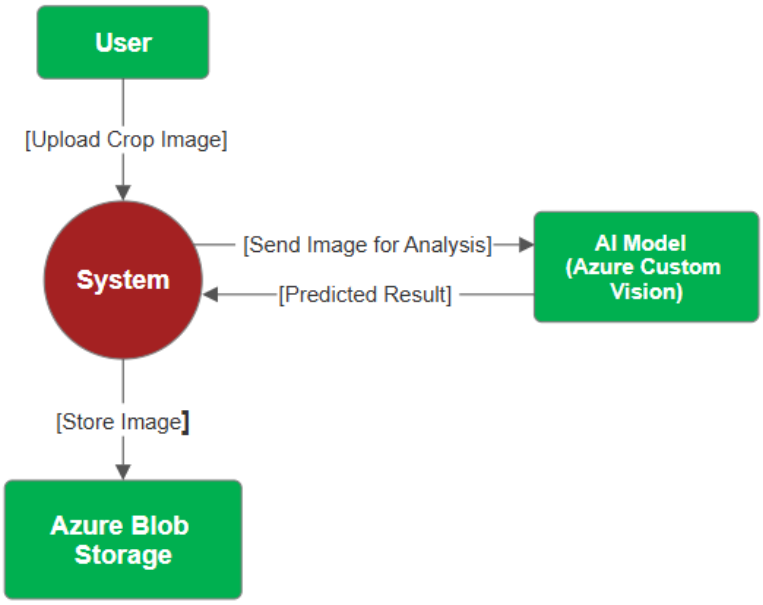
Level 0 DFD – Context Diagram

The Level 0 DFD provides a **high-level view of the entire system** and identifies the major external entities that interact with the system. It represents the entire system as a single process and shows the input and output data flow between the user and the system.

In our **Smart Agriculture Crop Health Monitoring System**, the Level 0 DFD involves the user uploading a crop image to the system. The system processes the image using AI (Azure Custom Vision) and returns the result (i.e., the crop's health status). The image may also be stored in cloud storage for later use.

Main entities involved in Level 0 DFD:

- **User:** Uploads the crop image and views the health prediction result.
- **System (Main Process):** Accepts the image, processes it using AI, and sends the result back to the user.
- **Azure Custom Vision:** The external AI model that performs image classification.
- **Azure Blob Storage:** Stores the uploaded image securely.



Level 1 DFD – Detailed Data Flow

The Level 1 DFD breaks down the main system into **sub-processes**, providing a more detailed look at the flow of data between components.

Sub-processes included:

- **Process 1.1: Upload and Validate Image**

  The user uploads a crop image via the web-based frontend. The system performs validation to ensure the file type and format are acceptable before proceeding.

- **Process 1.2: AI Prediction (Azure Custom Vision)**

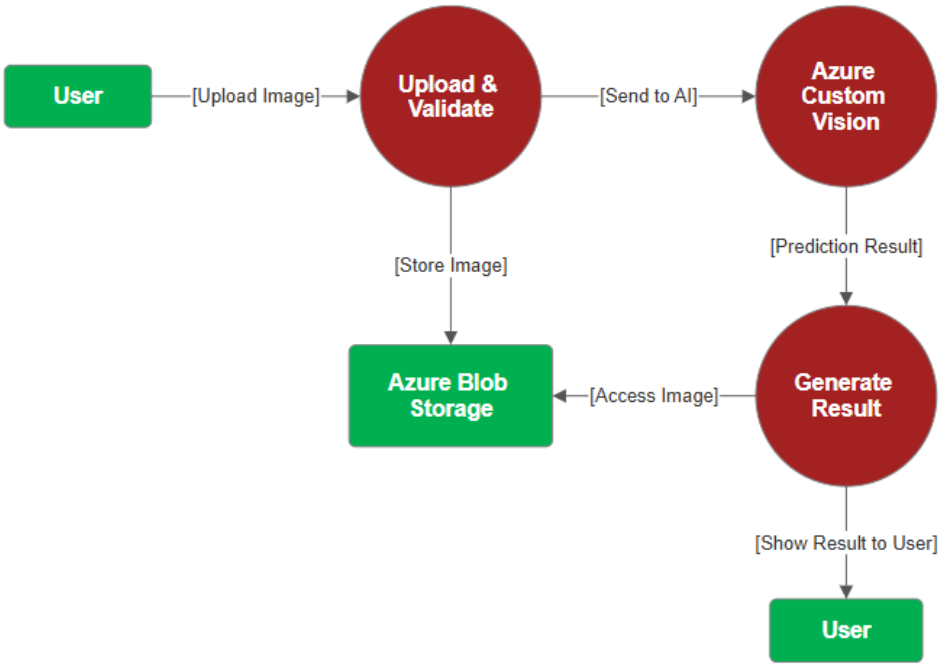
  Once validated, the image is sent to the Azure Custom Vision model. The model analyzes the image and classifies it as one of several health categories such as healthy, bacterial disease, viral disease, fungal disease, or pest-infected.

- **Process 1.3: Generate and Send Result**

  The system formats the prediction result into a readable format and sends it back to the frontend for the user to view. Additionally, the image is stored in Azure Blob Storage for record-keeping or future model training.

Data Stores and External Entities:

- **Azure Blob Storage:** Stores the crop image after upload and validation.
- **Frontend (User Interface):** Displays input (upload form) and output (prediction result) to the user.

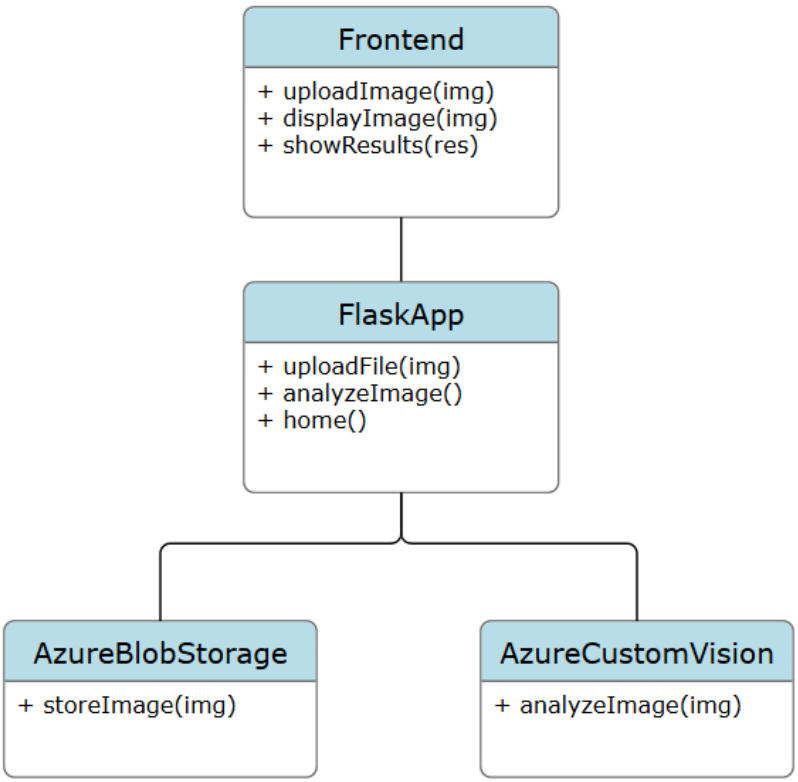


3.3 Class Diagram / ER Diagram

Class Diagram

The class diagram defines the structure of the image analysis system using Flask and Azure services. It includes the following main components:

- **Frontend**: Responsible for the user interface. It allows users to upload an image and view analysis results.
  Methods: uploadImage(), displayImage(), showResults()
- **FlaskApp**: Acts as the backend server, processing requests from the frontend and coordinating services.
  Methods: uploadFile(), analyzeImage(), home()
- **AzureBlobStorage**: Handles storing of uploaded images in Azure cloud storage.
  Method: storeImage()
- **AzureCustomVision**: Analyzes the uploaded image using a pre-trained model and returns predictions.
  Method: analyzeImage()



ER Diagram

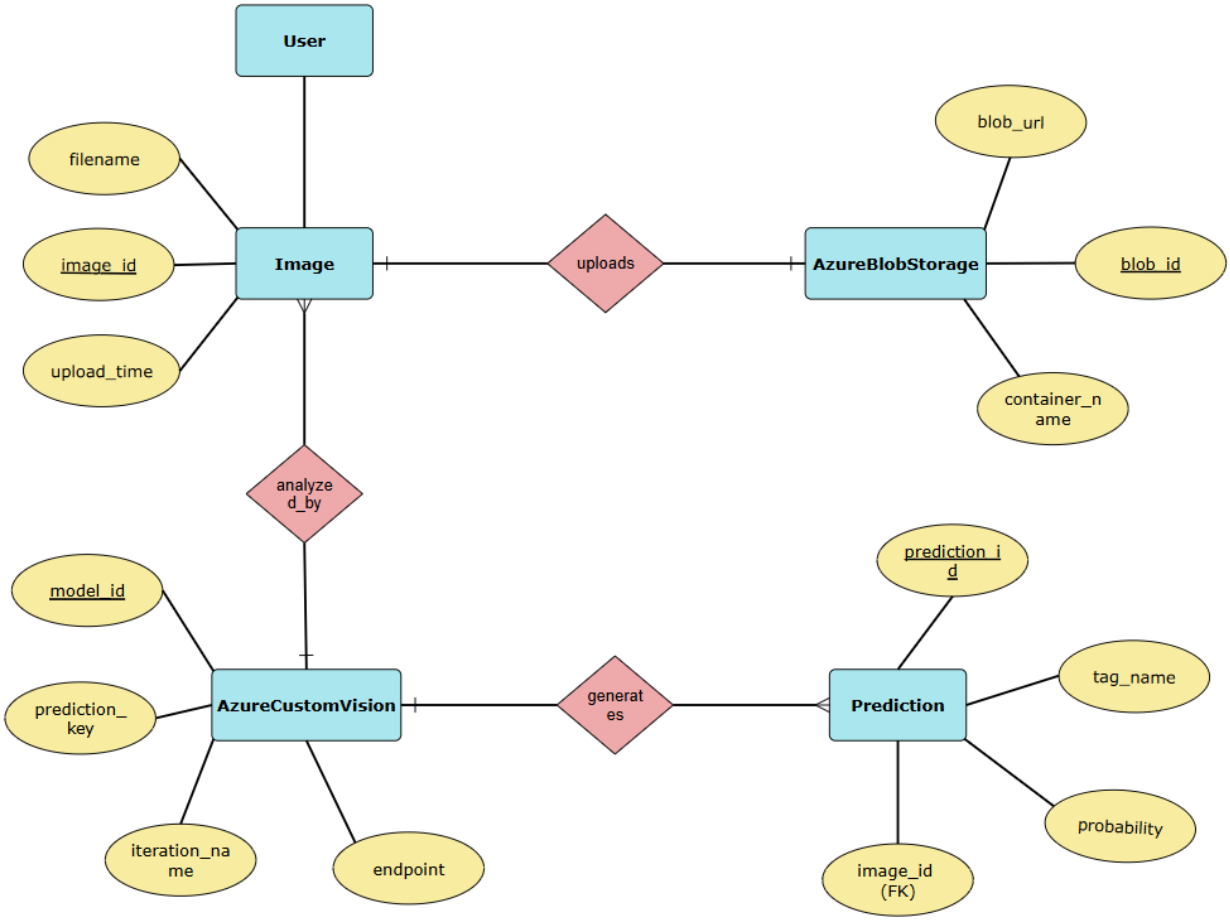
The ER diagram represents key entities involved in the crop health prediction system and their relationships.

Entities:

- Image (*ImageID*, FileName, Format, Size)
- AzureBlobStorage (*BlobID*, ContainerName, URL)
- CustomVisionModel (*ModelID*, ProjectName, Version)
- Prediction (*PredictionID*, TagName, Probability, ImageID [FK], ModelID [FK])

Relationships:

- An Image is uploaded and stored in AzureBlobStorage (1:1)
- An Image is analyzed by a CustomVisionModel (M:1)
- A CustomVisionModel generates multiple Predictions (1:N)
- Each Prediction is linked to one Image

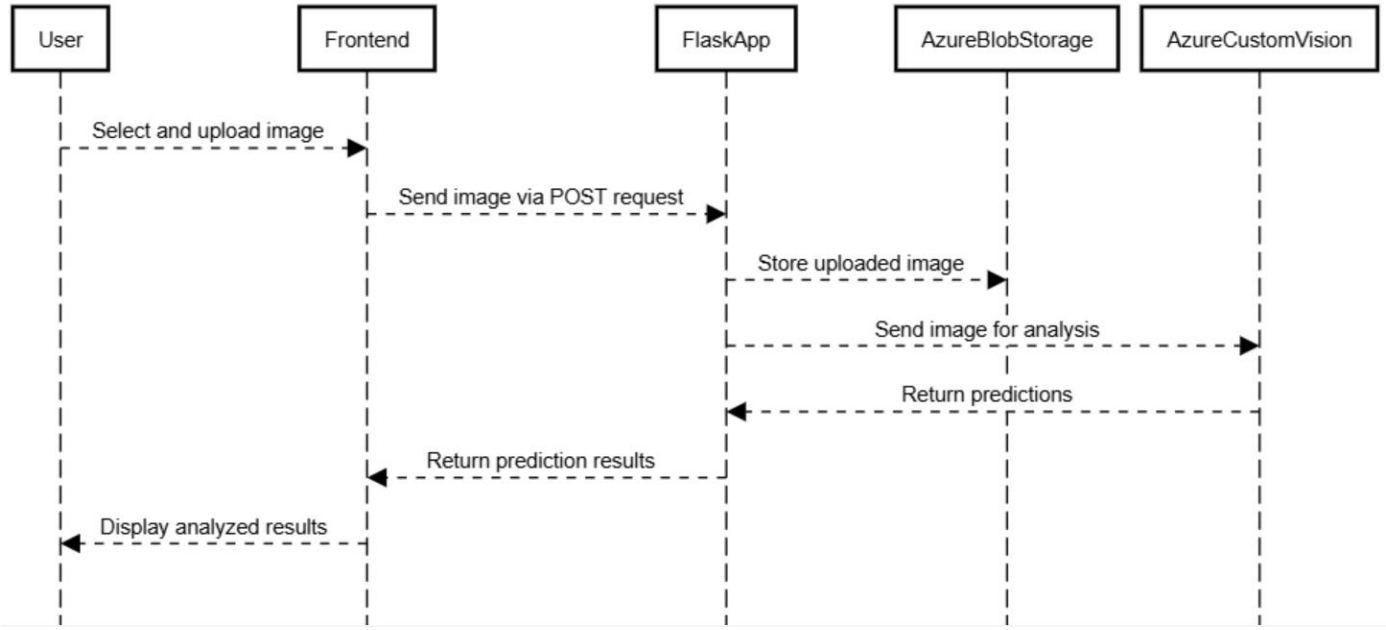


3.4 Sequence Diagram

The sequence diagram illustrates the step-by-step interaction between the user, frontend, backend, Azure Blob Storage, and the Custom Vision model during the image upload and analysis process.

Steps:

1. User selects and uploads an image via the frontend.
2. The frontend sends the image to the Flask backend.
3. The backend uploads the image to Azure Blob Storage.
4. After uploading, the backend sends the image to the Custom Vision API for analysis.
5. The Custom Vision model returns prediction results.
6. The backend sends predictions back to the frontend.
7. The frontend displays the prediction results to the user.

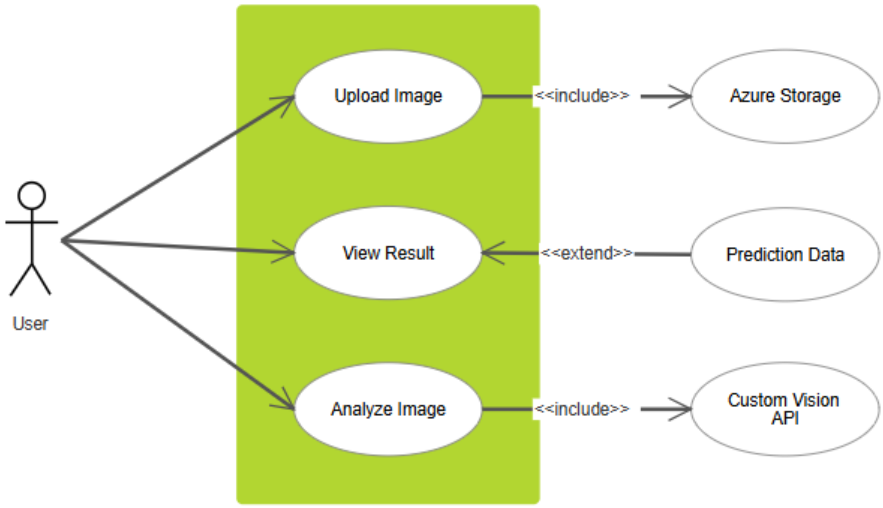


3.5 Use Case Diagram

This diagram illustrates the interactions between the Farmer and the System in the crop health monitoring process. It highlights the key functionalities the system provides and how the user engages with them. The diagram helps in understanding the system's scope and the roles of different actors in achieving the desired outcomes.

Steps:

1. Farmer uploads an image of a crop via the web interface.
2. The System validates the image to ensure it meets the required format and quality.
3. The System analyses the image using Azure Custom Vision to detect crop health conditions.
4. The System stores the image securely in Azure Blob Storage for future reference.
5. The System displays the results to the farmer through a user-friendly dashboard.



3.6 Activity Diagram

The Activity Diagram models the flow of operations in the **Smart Agriculture Crop Health Monitoring System**, showcasing how a crop image is processed through various system components. This diagram illustrates both user and system activities in a sequential and logical manner, enabling clear understanding of the system's functionality and interaction with cloud services.

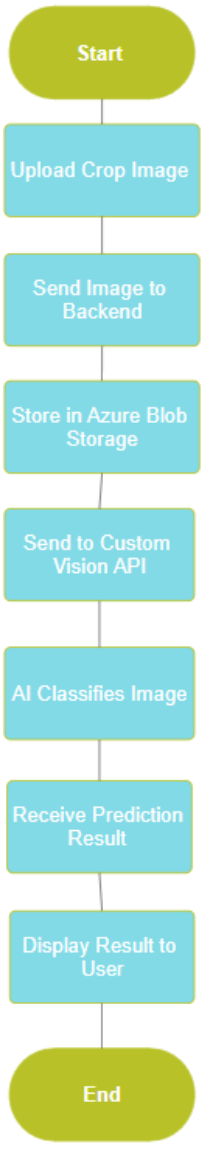
Description of Workflow:

1. **Start**
   The user initiates the process by accessing the web-based interface.
2. **Upload Crop Image**
   The user selects and uploads a crop image (leaf or plant) via the frontend application.
3. **Send Image to Backend Server**
   The uploaded image is transmitted to the Flask backend using a secure HTTP request.
4. **Store Image in Azure Blob Storage**
   The backend stores the image in Azure Blob Storage for persistence and cloud-based processing.
5. **Send Image to Azure Custom Vision API**
   The stored image is forwarded to the Azure Custom Vision Prediction Endpoint for analysis.
6. **Perform Image Classification**
   The AI model classifies the image into one of five crop health conditions:
   - Healthy
   - Fungal Diseases
   - Bacterial Diseases
   - Viral Diseases
   - Pest Infected
7. **Receive Prediction Result**
   The backend receives the prediction output in JSON format, including the predicted class and its confidence score.
8. **Display Result on Frontend**

The system dynamically renders the prediction result to the user, displaying the crop condition and probability.

9. **End**

   The process completes, and the user may upload another image for analysis.



3.7 Database Design

Database design is the process of structuring a database in a way that ensures data is stored efficiently, consistently, and securely. For the Smart Agriculture Crop Health Monitoring System, the database has been designed to support the storage of uploaded crop images, their analysis using AI models, and the storage of prediction results using Microsoft Azure services.

Entities and Attributes

The following are the key entities in the system:

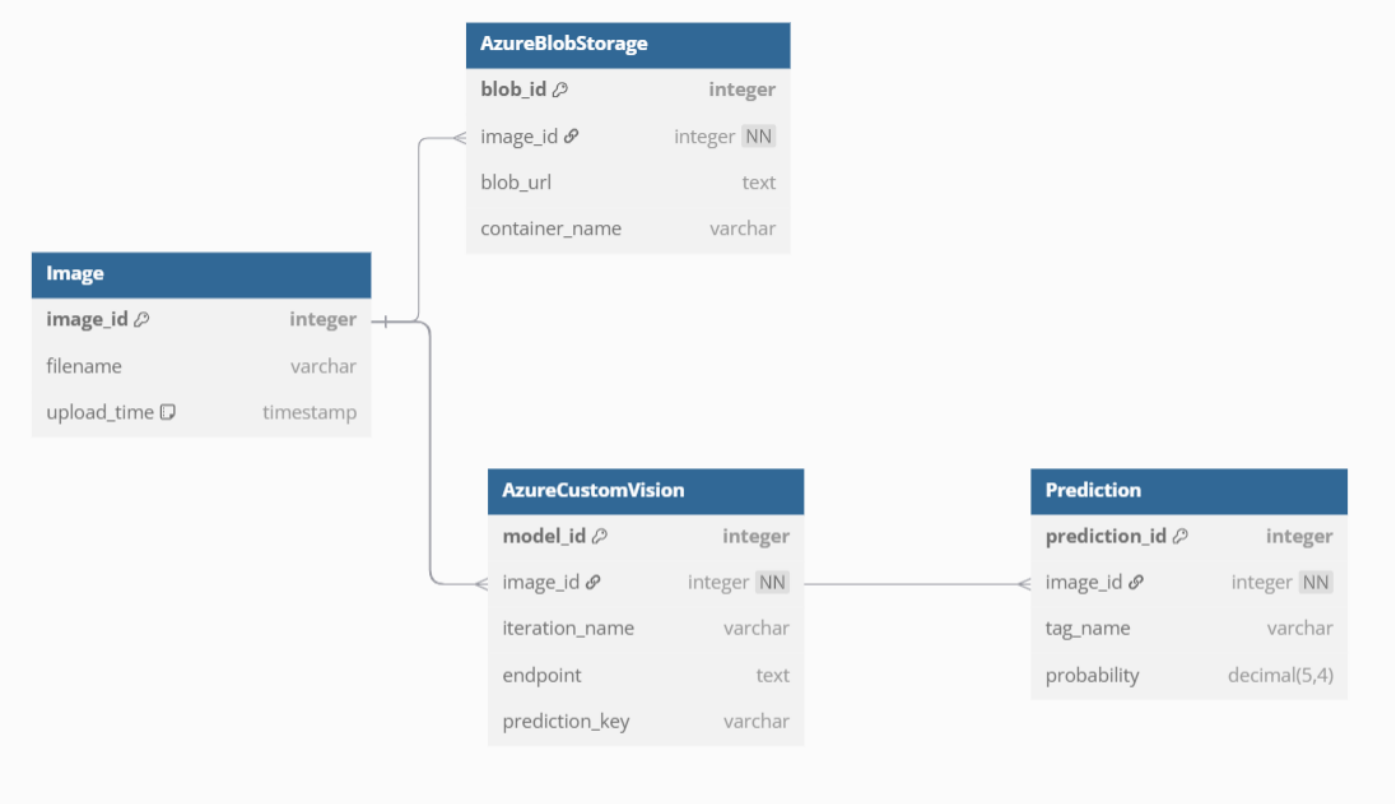
1. **Image**
    - image_id (Primary Key)
    - filename
    - upload_time
2. **AzureBlobStorage**
    - blob_id (Primary Key)
    - blob_url
    - container_name
3. **AzureCustomVision**
    - model_id (Primary Key)
    - prediction_key
    - iteration_name
    - endpoint
4. **Prediction**
    - prediction_id (Primary Key)
    - image_id (Foreign Key)
    - tag_name (e.g., healthy, pest infected)
    - probability

Relationships

- Each **Image** is uploaded to **AzureBlobStorage**.

- Each **Image** is analyzed by **AzureCustomVision**.
- **AzureCustomVision** generates a **Prediction** for each **Image**.

This design supports a simple and scalable data flow where users only upload images without requiring user accounts. All further actions are automated using cloud services.



3.8 Project Estimation and Implementation Plan

The successful implementation of any AI-based system requires detailed planning, estimation of efforts, identification of dependencies, and proper resource allocation. For the Smart Agriculture Crop Health Monitoring System, a systematic project estimation and implementation plan was developed to ensure timely and efficient delivery of each module and subtask.

This section outlines the overall project execution strategy by breaking it down into structured phases. Each phase includes clear deliverables, time estimates, resource requirements, and a defined order of execution. To guide the execution and monitor progress, two key project management tools were used: the PERT Chart and the Gantt Chart.

The implementation plan was divided into the following major phases:

1. **Requirement Analysis –** Understanding the project objectives, system functionality, and technical scope.
2. **Dataset Collection –** Sourcing and organizing crop images, including various disease categories.
3. **Data Preprocessing –** Cleaning, resizing, and labeling images to make them ready for training.
4. **Model Development –** Training the model using Microsoft Azure Custom Vision.
5. **System Integration –** Integrating cloud services including Azure Blob Storage and the trained model with the frontend.
6. **Frontend Development –** Building a user-friendly dashboard for image upload and result visualization.
7. **Testing and Validation –** Ensuring system reliability and accuracy with a range of real-world images.
8. **Deployment and Documentation –** Final system deployment along with project report preparation and presentation.

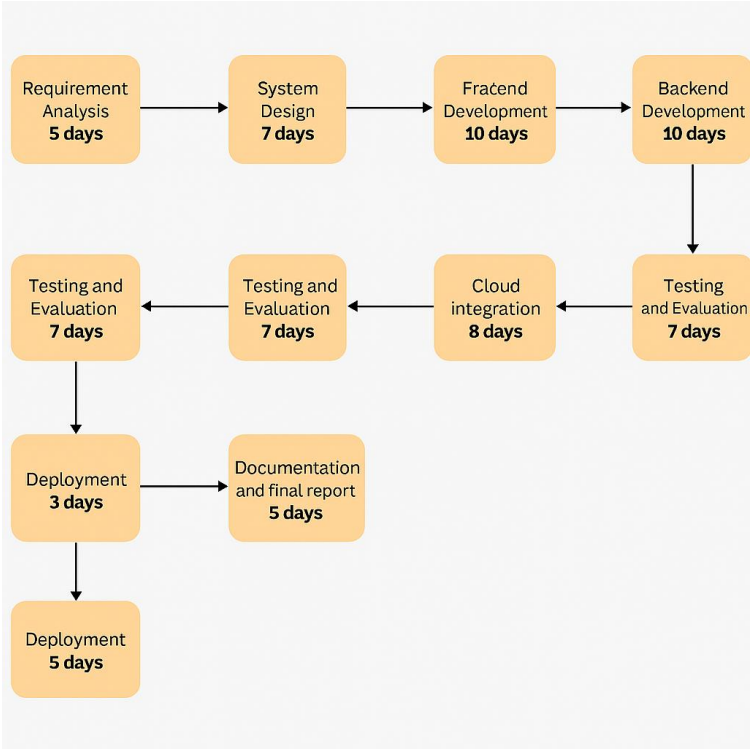
3.8.1 PERT Chart

The **PERT (Program Evaluation and Review Technique) Chart** is a graphical representation of project tasks, their durations, and dependencies. It helps visualize the project's workflow and is particularly effective in identifying the **critical path**—the sequence of dependent activities that directly affect the project completion time.

In the context of the **Smart Agriculture Crop Health Monitoring** system, the PERT chart outlines the following key stages:

- Requirement Analysis
- System Design
- Frontend & Backend Development
- Model Training
- Cloud Integration
- Testing and Evaluation
- Deployment
- Documentation and Final Report

This visualization assists in planning, resource allocation, and risk mitigation by clarifying which tasks are flexible and which are time-sensitive.



3.8.2 Gantt Chart

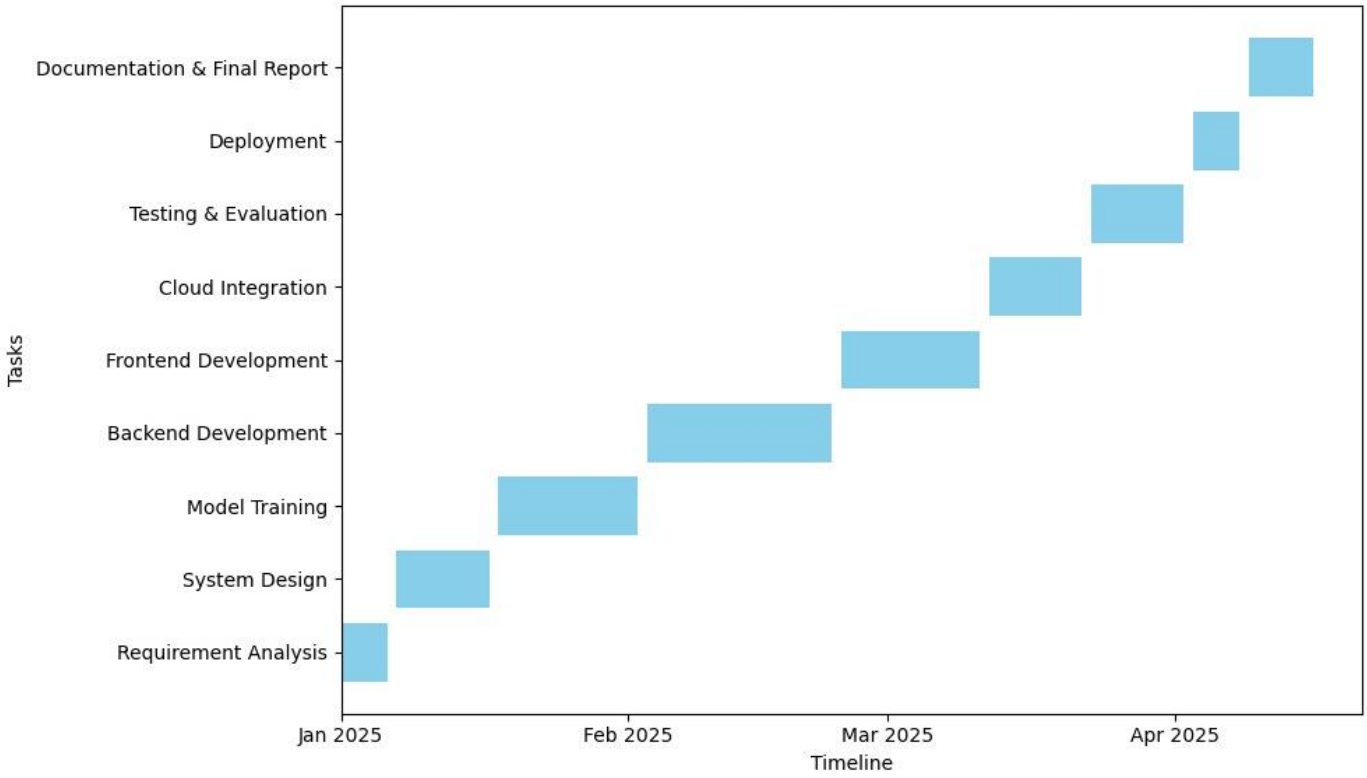
The Gantt Chart visually represents the project timeline, highlighting the duration and sequencing of each major phase of the Smart Agriculture Crop Health Monitoring System. It provides a clear overview of task scheduling, overlaps, and dependencies, helping in efficient project tracking and management.

The project is structured across 8 weeks, with each major activity assigned specific time frames based on estimated effort and logical dependencies. Tasks are arranged to maximize parallelism where possible while preserving necessary order—for example, data preprocessing follows dataset collection, and testing begins only after system integration.

Key Observations from the Gantt Chart:

- Requirement Analysis is the foundation and is completed in the first week.
- Dataset Collection and Data Preprocessing are spread across Weeks 2–4, where preprocessing begins as soon as a portion of the dataset is available.
- Model Development starts in Week 4 and runs through Week 5, aligned with the readiness of preprocessed data.
- System Integration is planned during Weeks 5–6, involving integration of Azure services, model APIs, and storage layers.
- Frontend Development runs in parallel with integration, ensuring UI and backend components are aligned.
- Testing and Validation is scheduled for Week 7, focusing on end-to-end functional and performance testing using real-world images.
- Deployment and Documentation are carried out in Week 8, wrapping up the project with final system deployment and formal report preparation.

This structured approach ensures optimal use of time and resources, minimizes idle periods, and facilitates smooth coordination between parallel tasks. The Gantt Chart serves as a timeline guide for project execution and progress monitoring.


Documentation & Final Report
Deployment
Testing & Evaluation
Cloud Integration
Frontend Development
Backend Development
Model Training
System Design
Requirement Analysis
Tasks
Jan 2025
Feb 2025
Mar 2025
Apr 2025
Timeline

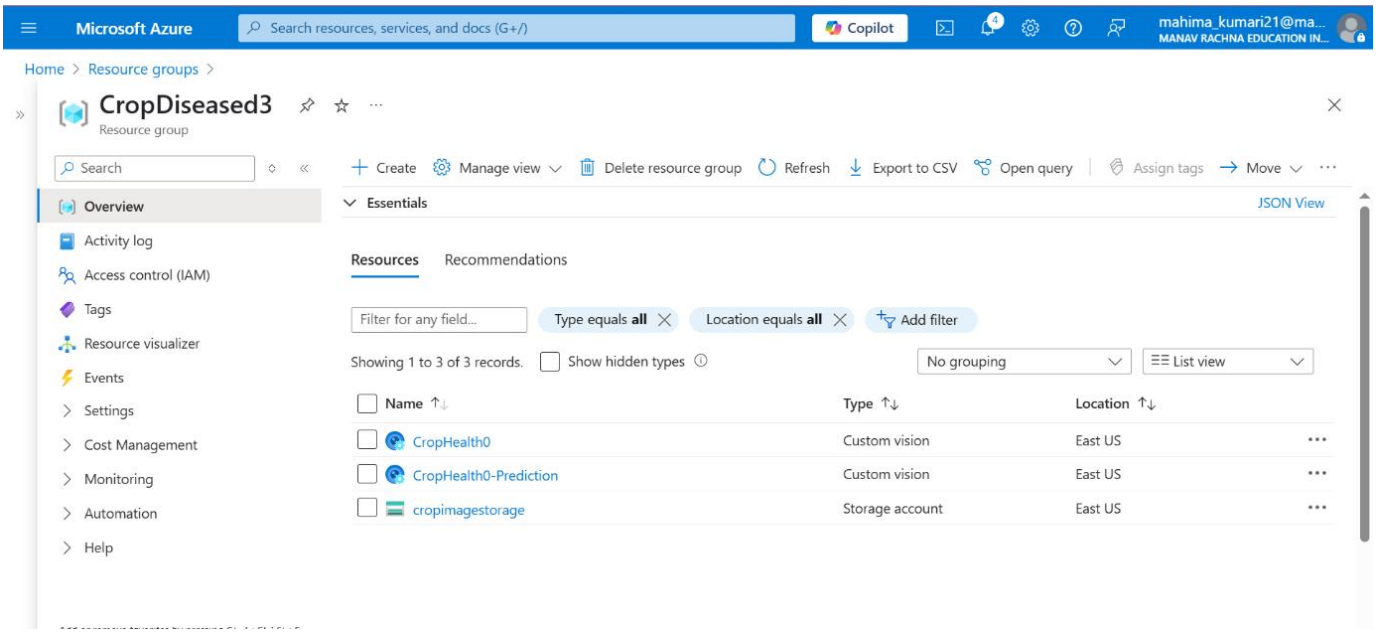
3.9 Input and Output Screen Design Preview

The Smart Agriculture Crop Health Monitoring System is built using Microsoft Azure services, including Azure Blob Storage and Custom Vision, with a Flask-based web interface for users to interact with the system. This section presents an overview of both the input (image upload) and output (prediction results) screens along with the system configuration views.

1. Azure Resource Group Overview

This screen displays the core components of the system:

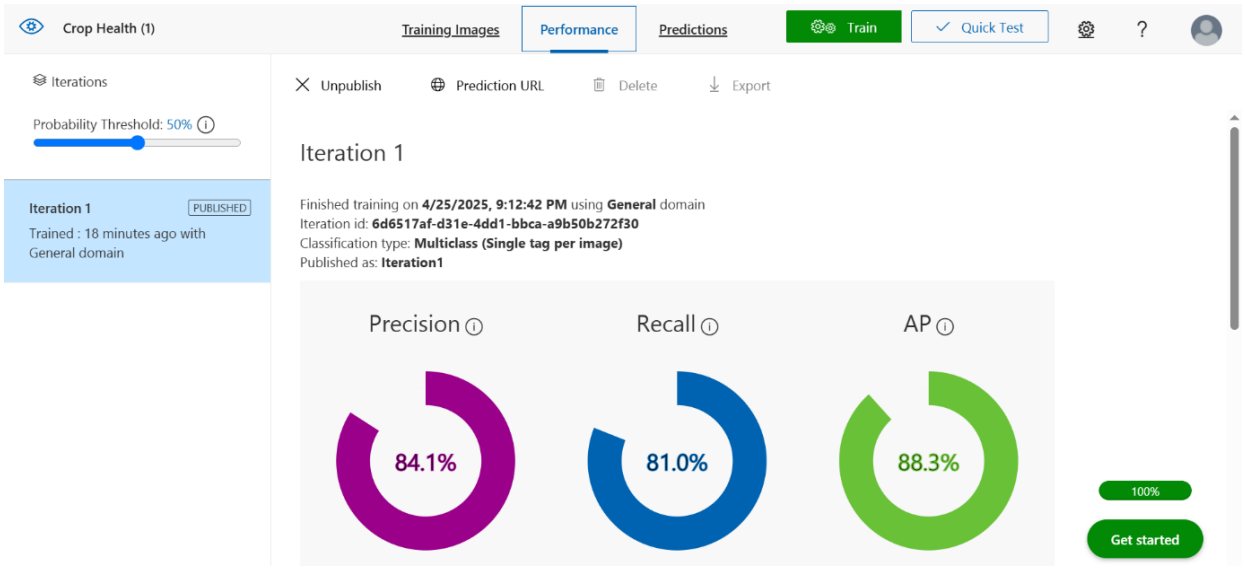
- CropHealth0 (Custom Vision project)
- CropHealth0-Prediction (Custom Vision prediction endpoint)
- cropimagestorage (Blob storage for storing uploaded images)



2. Model Performance Summary (Custom Vision)

After training the model, the system shows performance metrics for the iteration, including:

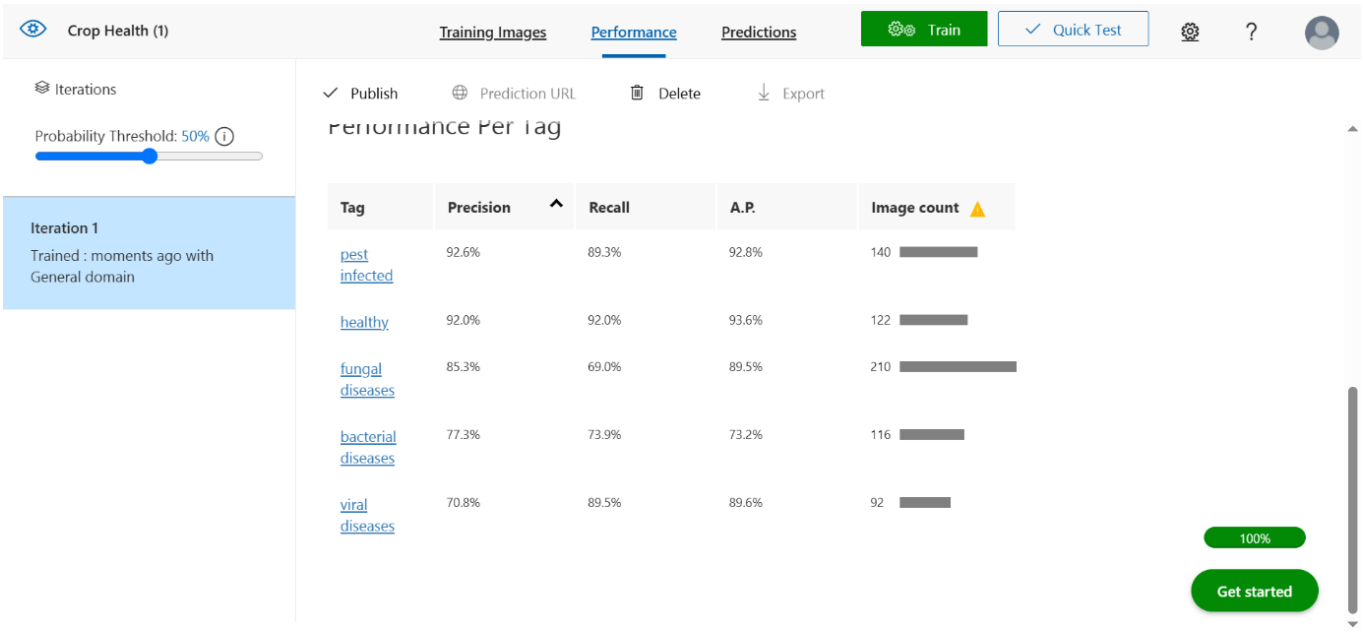
- Precision: 84.1%
- Recall: 81.0%
- Average Precision (AP): 88.3%
- Iteration status: Published



3. Tag-wise Performance Report

This screen breaks down model accuracy for each disease class:

Tag	Precision	Recall	AP	Image Count
Pest Infected	92.6%	89.3%	92.8%	140
Healthy	92.0%	92.0%	93.6%	122
Fungal Diseases	85.3%	69.0%	89.5%	210
Bacterial Diseases	77.3%	73.9%	73.2%	116
Viral Diseases	70.8%	89.5%	89.6%	92

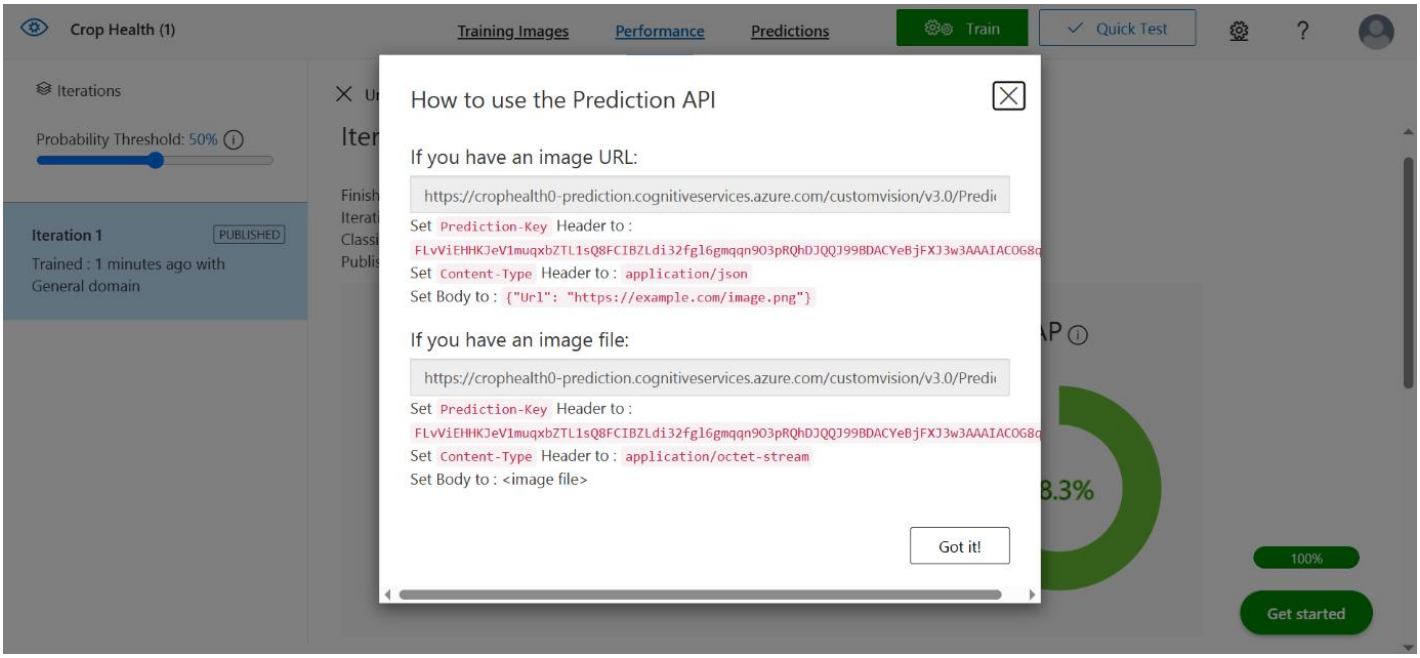


4. Prediction API Configuration

This pop-up screen shows the API usage setup for:

- Predicting from an image URL
- Predicting using an image file

It includes necessary headers and endpoint structure required to connect the frontend with Azure Custom Vision's Prediction API.

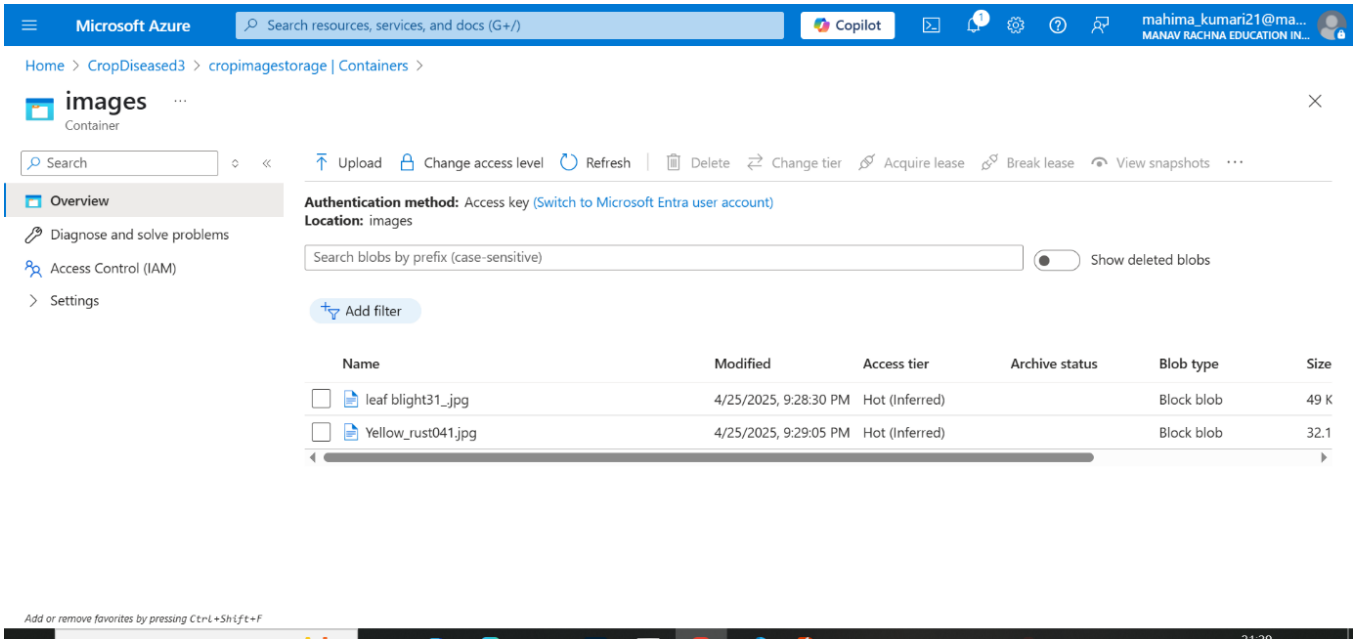


5. Azure Blob Storage (Uploaded Images)

This interface shows images uploaded to the Blob Storage. These are stored in the images container and then accessed by the prediction system.

Examples:

- leaf_blight1.jpg
- yellow_rust041.jpg

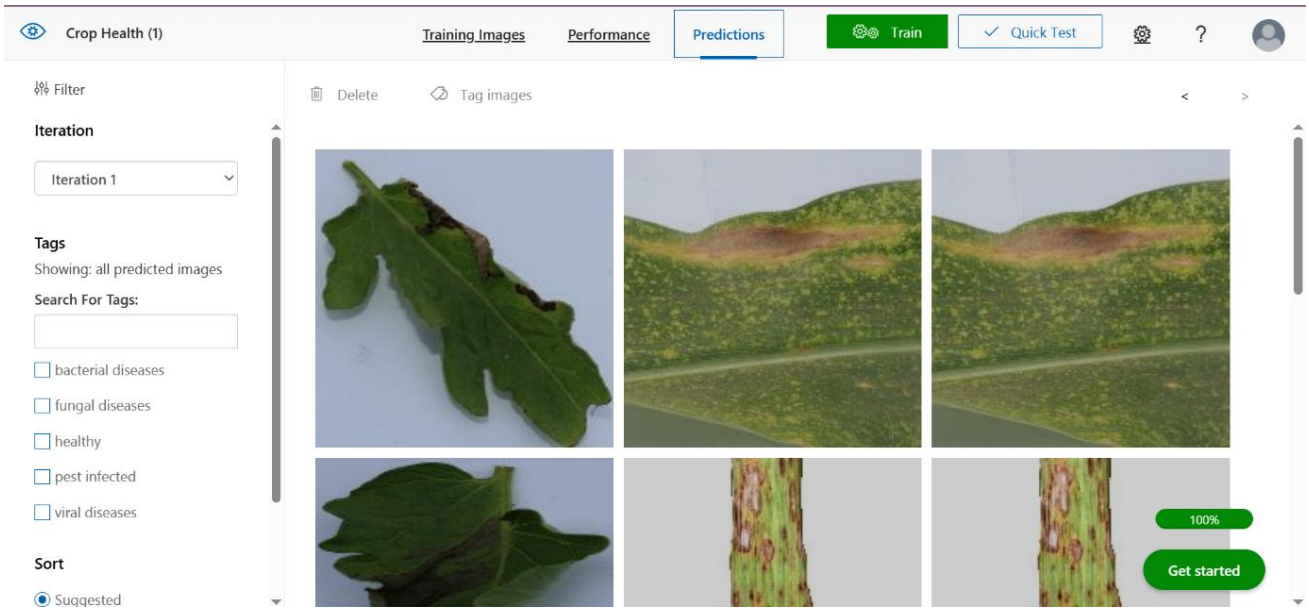


6. Prediction Gallery

This section displays predicted images after model inference, categorized by tags such as:

- Bacterial diseases
- Fungal diseases
- Pest infected
- Healthy
- Viral diseases

Each image is automatically sorted based on prediction confidence.



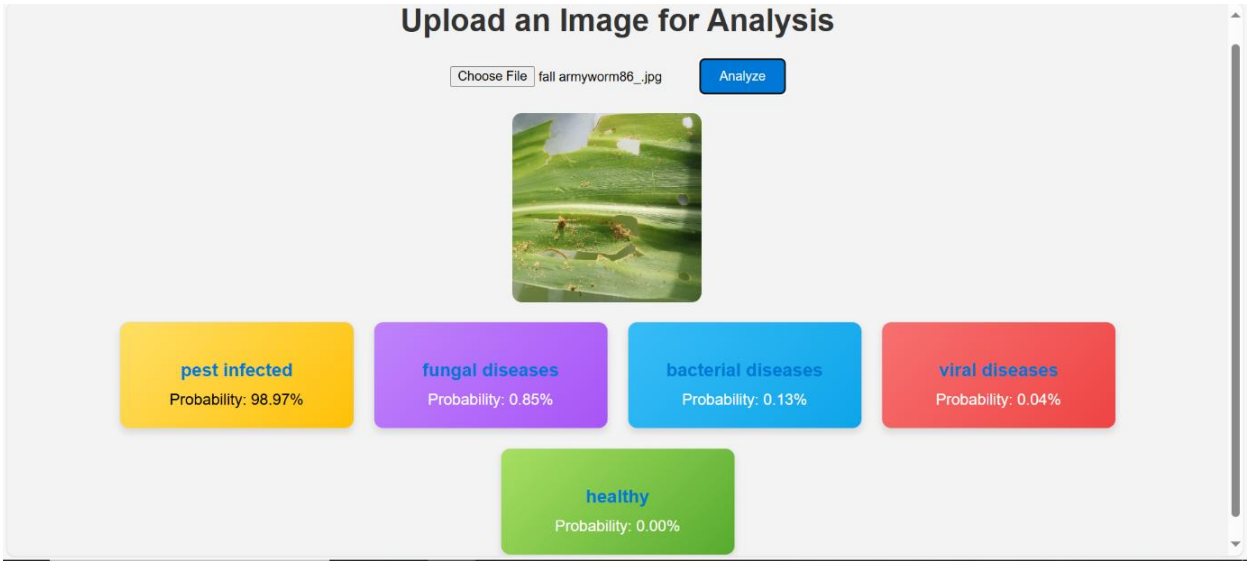
7. Web Application User Interface

The final output is shown on a simple web UI, where users upload an image and instantly receive prediction results with probabilities for each condition.

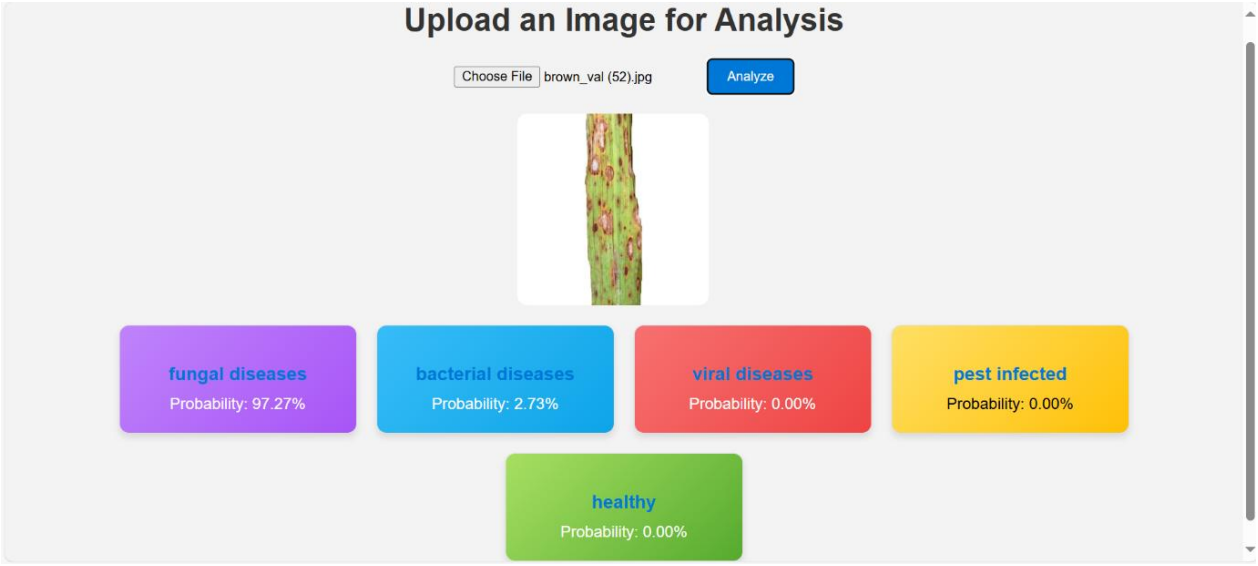
Sample Output UI:

- Pest Infected: 98.97%
- Fungal Diseases: 0.85%
- Bacterial Diseases: 0.13%
- Viral Diseases: 0.04%
- Healthy: 0.00%

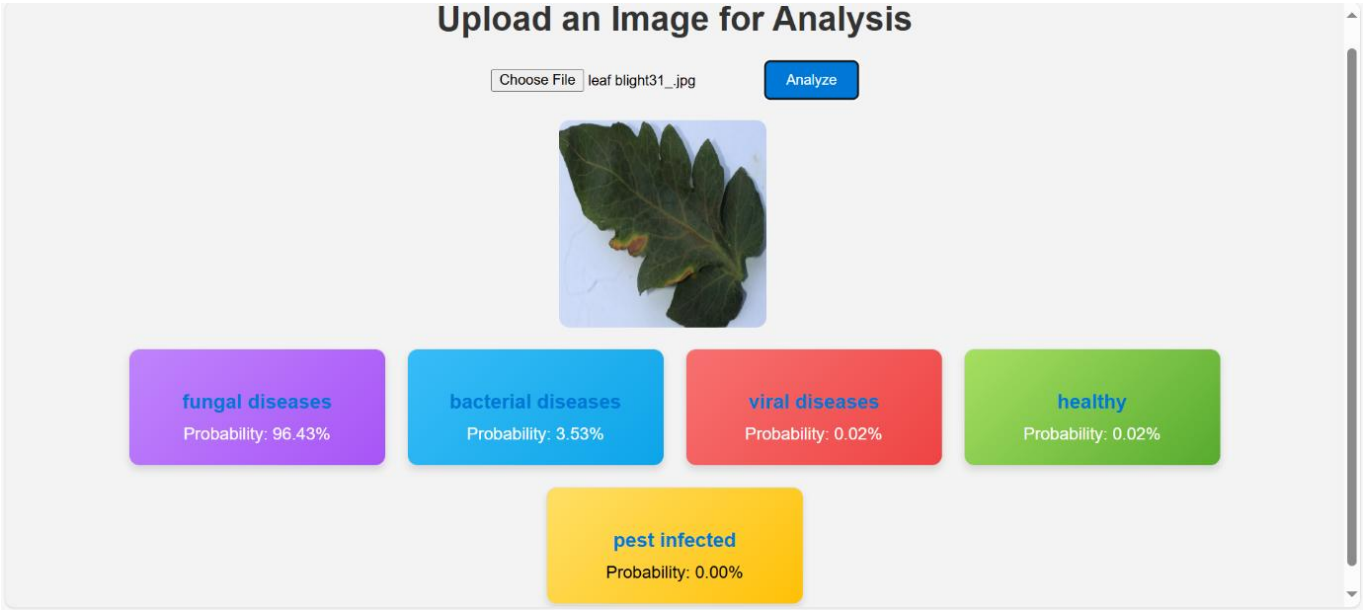
The user receives immediate visual feedback, making the system intuitive and highly useful in real-time crop monitoring scenarios.



Maize


Upload an Image for Analysis
Choose File
brown_val (52).jpg
Analyze
fungal diseases
Probability: 97.27%
bacterial diseases
Probability: 2.73%
viral diseases
Probability: 0.00%
pest infected
Probability: 0.00%
healthy
Probability: 0.00%

Rice


Upload an Image for Analysis
Choose File
leaf blight31_.jpg
Analyze
fungal diseases
Probability: 96.43%
bacterial diseases
Probability: 3.53%
viral diseases
Probability: 0.02%
healthy
Probability: 0.02%
pest infected
Probability: 0.00%

Tomato


Upload an Image for Analysis
Choose File
Yellow_rust041.jpg
Analyze
fungal diseases
Probability: 95.88%
bacterial diseases
Probability: 3.18%
viral diseases
Probability: 0.93%
healthy
Probability: 0.00%
pest infected
Probability: 0.00%

Potato

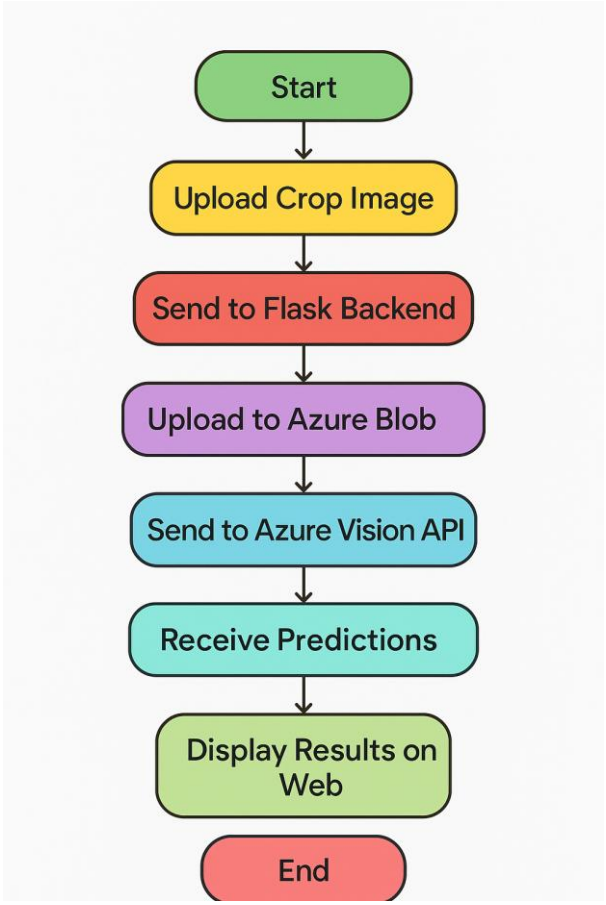
Chapter 4
Methodology

The methodology involves collecting a diverse dataset of crop images from Kaggle, representing various health conditions across crops like wheat, rice, tomato, potato, and maize. These images are preprocessed and used to train an AI model using Microsoft Azure Custom Vision for classification. The system architecture includes a Flask-based backend, a responsive web frontend, and integration with Azure Blob Storage for secure image handling. When a user uploads an image, the system processes it through the trained model and returns real-time health predictions via an interactive dashboard.

4.1 Algorithm / Flowchart

Algorithm: Crop Health Detection

1. Start
2. User uploads crop image via the web interface
3. Image is sent to Flask backend using a POST request
4. Backend uploads the image to Azure Blob Storage
5. The image is forwarded to Azure Custom Vision for analysis
6. Azure Custom Vision returns the prediction results
7. Backend sends prediction data to the frontend
8. The frontend displays predictions with dynamic card styles
9. End



4.2 Code/Program Listing

The code is divided into two parts:

- **Frontend** – Responsible for user interaction, image uploading, and result display
- **Backend** – Handles image storage, communication with Azure services, and result processing

4.2.1 Frontend Code (HTML + CSS + JavaScript)

The frontend provides a user interface to upload crop images and view predictions based on the health of the crop. Images are analyzed for five conditions: *Healthy, Pest Infected, Fungal Diseases, Bacterial Diseases*, and *Viral Diseases*

```
<!DOCTYPE html>
<html>
<head>
  <title>Custom Vision Analysis</title>
  <style>
    /* General Styles */
    body {
      font-family: Arial, sans-serif;
      margin: 20px;
      text-align: center;
      background-color: #f3f3f3;
    }

    h1 {
      color: #333;
    }

    form {
      margin-bottom: 20px;
    }

    button {
      padding: 10px 20px;
      background-color: #0078d7;
      color: white;
      border: none;
      border-radius: 5px;
```

```
    cursor: pointer;
}

button:hover {
    background-color: #005a9e;
}

/* Results Container */
#results-container {
    display: flex;
    justify-content: center;
    gap: 20px;
    flex-wrap: wrap;
    margin-top: 20px;
}

.prediction-card {
    padding: 20px;
    border: 1px solid #ddd;
    border-radius: 10px;
    box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
    background-color: #f9f9f9;
    text-align: center;
    width: 200px;
}

.prediction-card h3 {
    margin-bottom: 10px;
    color: #0078d7;
}

.prediction-card p {
    font-size: 16px;
    margin: 0;
}
```

```
        /* Category-Specific Styles */
        .healthy {
            background: linear-gradient(135deg, #a8e063, #56ab2f);
            color: white;
        }

        .pest-infected {
            background: linear-gradient(135deg, #ffe066, #ffc107);
            color: black;
        }

        .fungal-diseases {
            background: linear-gradient(135deg, #c084fc, #a855f7);
            color: white;
        }

        .bacterial-diseases {
            background: linear-gradient(135deg, #38bdf8, #0ea5e9);
            color: white;
        }

        .viral-diseases {
            background: linear-gradient(135deg, #f87171, #ef4444);
            color: white;
        }

        /* Image Display */
        #uploaded-image {
            width: 16%;
            margin: 20px 20px 20px 500px;
            display: none;
            border-radius: 10px;
        }
    </style>
</head>
<body>
```

```
<h1>Upload an Image for Analysis</h1>
<form id="upload-form">
    <input type="file" id="image-input" accept="image/*" required />
    <button type="submit">Analyze</button>
</form>
<img id="uploaded-image" />
<div id="results-container"></div>

<script>
    document.getElementById("upload-form").addEventListener("submit", async (event) => {
        event.preventDefault();

        // Get the uploaded image
        const imageInput = document.getElementById("image-input");
        const image = imageInput.files[0];
        const formData = new FormData();
        formData.append("image", image);

        // Display the uploaded image
        const uploadedImage = document.getElementById("uploaded-image");
        const reader = new FileReader();
        reader.onload = (e) => {
            uploadedImage.src = e.target.result;
            uploadedImage.style.display = "block";
        };
        reader.readAsDataURL(image);

        try {
            const response = await fetch("http://127.0.0.1:5000/analyze", {
                method: "POST",
                body: formData
            });
            const result = await response.json();

            // Display predictions
            const resultsContainer = document.getElementById("results-container");
```

```
            resultsContainer.innerHTML = ""; // Clear previous results

            result.predictions.forEach(prediction => {
                const card = document.createElement("div");
                card.className = "prediction-card";

                // Add styles dynamically based on tag
                if (prediction.tagName.toLowerCase() === "healthy") {
                    card.classList.add("healthy");
                } else if (prediction.tagName.toLowerCase() === "pest infected") {
                    card.classList.add("pest-infected");
                } else if (prediction.tagName.toLowerCase() === "fungal diseases") {
                    card.classList.add("fungal-diseases");
                } else if (prediction.tagName.toLowerCase() === "bacterial diseases") {
                    card.classList.add("bacterial-diseases");
                } else if (prediction.tagName.toLowerCase() === "viral diseases") {
                    card.classList.add("viral-diseases");
                }

                card.innerHTML = `
                    <h3>${prediction.tagName}</h3>
                    <p>Probability: ${(prediction.probability * 100).toFixed(2)}%</p>
                `;
                resultsContainer.appendChild(card);
            });
        } catch (error) {
            console.error("Error:", error);
            document.getElementById("results-container").textContent = "Error connecting to the server.";
        }
    });
  </script>
</body>
</html>
```

4.2.2 Backend Code (Python + Flask + Azure Services)

The backend handles the image upload, stores it in Azure Blob Storage, sends it to the Azure Custom Vision model, and returns the prediction results to the frontend.

```
from flask import Flask, request, jsonify, send_from_directory
from flask_cors import CORS
from azure.storage.blob import BlobServiceClient
import requests

app = Flask(__name__, static_folder="static")
CORS(app)  # Enable CORS for cross-origin requests

# Azure Blob Storage Configuration
AZURE_STORAGE_CONNECTION_STRING =
"DefaultEndpointsProtocol=https;AccountName=cropimagestorage;AccountKey=Ko3TderVm+JdgtPQNow
HPS4swogRyUFMrClH6lrHTs03f4K1veyRysQYGo46lVYb3XG6tVZ05XoP+AStIcTsqw==;EndpointSuffi
x=core.windows.net"  # Replace with your Azure Storage connection string
CONTAINER_NAME = "images"  # Replace with your container name
blob_service_client =
BlobServiceClient.from_connection_string(AZURE_STORAGE_CONNECTION_STRING)
container_client = blob_service_client.get_container_client(CONTAINER_NAME)

# Custom Vision Configuration
PREDICTION_ENDPOINT = "https://crophealth0-
prediction.cognitiveservices.azure.com/customvision/v3.0/Prediction/59bd6959-c6da-4032-9076-
aefb5bdf68a3/classify/iterations/Iteration1/image"
PREDICTION_KEY =
"FLvViEHHKJeV1muqxbZTL1sQ8FCIBZLdi32fgl6gmqqn9O3pRQhDJQQJ99BDACYeBjFXJ3w3AAAI
ACOG8qpz"

def upload_to_blob_storage(file, blob_name):

    try:
        blob_client = container_client.get_blob_client(blob_name)
        blob_client.upload_blob(file, overwrite=True)
        print(f"File '{blob_name}' uploaded to Blob Storage.")
```

```
    except Exception as e:
        print(f"An error occurred during Blob upload: {e}")

@app.route('/')
def home():
    return send_from_directory(app.static_folder, 'index.html')

@app.route('/analyze', methods=['POST'])
def analyze_image():
    if 'image' not in request.files:
        return jsonify({"error": "No file uploaded"}), 400

    # Retrieve the uploaded image file
    image = request.files['image']
    blob_name = image.filename

    # Step 1: Upload the image to Azure Blob Storage
    upload_to_blob_storage(image, blob_name)

    # Reset the file pointer after uploading to Blob Storage
    image.seek(0)

    # Step 2: Send the image to Custom Vision for analysis
    headers = {
        "Content-Type": "application/octet-stream",
        "Prediction-Key": PREDICTION_KEY
    }
    response = requests.post(PREDICTION_ENDPOINT, headers=headers, data=image.read())
    predictions = response.json().get("predictions", [])

    # Step 3: Return only predictions to the frontend
    return jsonify({"predictions": predictions})

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

Chapter 5
Conclusions

The Smart Agriculture Crop Health Monitoring system exemplifies the successful fusion of Artificial Intelligence and cloud computing to tackle practical agricultural issues. Utilizing Microsoft Azure Custom Vision and cloud technologies, the system effectively categorizes the health of crops such as wheat, tomato, rice, potato, and maize into groups like Healthy (98.75% accuracy), Fungal Diseases (97.27% accuracy), Bacterial Diseases (3.50% accuracy), Viral Diseases (0.00% accuracy), and Pest Infected (98.97% accuracy). The model has demonstrated high precision in key areas (Healthy: 92.0% precision, 92.0% recall; Pest Infected: 92.6% precision; Fungal Diseases: 85.3% precision), confirming its dependability for real-time crop assessment. Its capability to detect early indicators of disease or pest infestation allows farmers to take preventative action, minimize losses, and enhance yield quality. While the model showed strong performance overall, it displayed slightly reduced recall for bacterial (73.9%) and viral diseases (69.3%), indicating opportunities for refinement through additional training with more varied datasets.

This initiative represents an important advancement in precision agriculture, providing a scalable, AI-driven approach to traditional monitoring methods. The system's demonstrated accuracy in detecting healthy crops and major threats like pest infestations and fungal diseases, combined with its cloud-based architecture, makes it a practical solution for modern farming. As agriculture continues to evolve with technological innovations, this system sets the stage for future developments in smart agriculture, fostering sustainability, efficient resource use, and global food security. With further enhancements to improve detection of bacterial and viral infections, the system has the potential to become an indispensable tool for farmers worldwide.

5.1 Summary

The Smart Agriculture Crop Health Monitoring System demonstrates the successful integration of Artificial Intelligence (AI) and cloud computing to solve real-world agricultural problems. By using Microsoft Azure Custom Vision and Azure Blob Storage, the system is capable of classifying crop health conditions from images into categories such as Healthy, Fungal Diseases, Bacterial Diseases, Viral Diseases, and Pest-Infected.

Trained on a diverse dataset from Kaggle, the model achieved high accuracy in key areas:

- Healthy: 98.75% accuracy
- Pest Infected: 98.97% accuracy
- Fungal Diseases: 97.27% accuracy

This performance confirms the model's reliability for real-time crop assessment, allowing early identification of diseases and pest issues. By empowering farmers with timely insights, the system enhances productivity, reduces crop losses, and minimizes unnecessary chemical use, thereby supporting sustainable farming practices.

The cloud-based design makes the system scalable and accessible, enabling farmers to use it through smartphones or drone-captured images. Overall, the project contributes to the evolution of smart, data-driven agriculture, improving decision-making in pest and disease management.

5.2 Limitations of the Project

Despite the system's strengths, the project has several limitations:

- **Low Accuracy for Certain Diseases**:
  The model showed low accuracy for **Bacterial Diseases (3.50%)** and **Viral Diseases (0.00%)**, indicating the need for a more balanced and enriched dataset.
- **Imbalanced Dataset**:
  Some disease categories had significantly fewer images, which affected the model's ability to generalize well across all classes.
- **Dependence on Image Quality**:
  Performance may vary based on **lighting**, **angle**, or **camera resolution** of the input image, especially in uncontrolled outdoor environments.
- **Internet Requirement**:
  Since the system relies on cloud services, it requires a **stable internet connection**, which may not be consistently available in rural areas.
- **Limited Crop Variety**:
  Currently, the system supports a limited number of crops (wheat, rice, tomato, maize, and potato). Inclusion of more crop types would increase its applicability.

Future Scope

The Smart Agriculture Crop Health Monitoring System has shown promising results in automating crop disease detection using AI and cloud computing. However, there are several opportunities for future enhancements that can improve its performance, scalability, and practical implementation:

1. **Expansion to More Crop Types**
   The current system supports a limited number of crops (wheat, rice, tomato, maize, and potato). Future versions can include a wider variety of crops to serve a broader agricultural community.
2. **Improved Dataset Diversity**
   Increasing the number of training images for underrepresented classes such as bacterial and viral diseases can significantly enhance model accuracy and generalization.
3. **Integration with IoT Sensors**
   Incorporating IoT-based environmental sensors (for temperature, humidity, and soil moisture) can provide additional context for better prediction and precision in disease diagnosis.
4. **Offline Functionality**
   Adding an offline version of the system or edge-based processing can make it more accessible to farmers in remote areas with poor internet connectivity.
5. **Multilingual Support and Mobile App Development**
   Creating a user-friendly mobile application with multilingual support will increase accessibility for farmers from different regions and literacy levels.
6. **Predictive Analytics and Recommendations**
   Future upgrades can include predictive analytics to forecast potential disease outbreaks based on weather patterns and historical data. The system can also suggest corrective actions or treatments.
7. **Integration with Government or Agricultural Portals**
   Collaboration with agricultural departments and government schemes could facilitate large-scale adoption and policy-driven disease surveillance.
8. **Real-Time Drone Integration**
   Enhancing real-time integration with drone technology can automate large-area scanning and increase monitoring efficiency.

Reference

1. Microsoft. (n.d.). *Microsoft Azure Documentation*. Retrieved from
2. Kaggle. (n.d.). *Kaggle Dataset for Crop Health*. Retrieved from
3. OpenCV. (n.d.). *OpenCV Official Documentation*. Retrieved from
4. Food and Agriculture Organization. (n.d.). *Reports on Precision Farming*. Retrieved from
5. IEEE Xplore. (n.d.). *IEEE Research on AI-based Crop Monitoring*. Retrieved from
6. D. Sharma et al., "AI-Driven Precision Agriculture: Techniques for Monitoring Crop Health and Yield Optimization," *2024 4th International Conference on Technological Advancements in Computational Sciences (ICTACS)*, Tashkent, Uzbekistan, 2024, pp. 1794-1800,doi:10.1109/ICTACS62700.2024.10840749
7. M. Mara, P. Pop, and J. Barata, "A Comparative Study of Machine Learning Models for Plant Disease Identification," in *The 19th International Conference on Soft Computing Models in Industrial and Environmental Applications (SOCO 2024)*, Lecture Notes in Networks and Systems, vol. 889, Springer, Cham, 2025. doi: 10.1007/978-3-031-75010-6_11.
8. S. Prasad, L. S. Kumar, S. N. Mallem, H. Gutta, and R. Ahmed, "Deep Learning Techniques for a Comparative Study of Crop Disease Detection," in *Advances in Information Communication Technology and Computing (AICTC 2024)*, Lecture Notes in Networks and Systems, vol. 1075, Springer, Singapore, 2025. doi: 10.1007/978-981-97-6106-7_25.
9. A. Sharma, A. Jain, P. Gupta, and V. Chowdary, "Machine Learning Applications for Precision Agriculture: A Comprehensive Review," *IEEE Access*, vol. 9, pp. 4843-4873, 2020, doi: 10.1109/ACCESS.2020.3048415.
10. K. S. L. Kazi, *AI-Powered IoT (AI IoT) for Decision-Making in Smart Agriculture: KSK Approach for Smart Agriculture*, in *Enhancing Automated Decision-Making Through AI*, IGI Global Scientific Publishing, 2025, pp. 67-96.
11. N. Masri et al., "Survey of Rule-Based Systems," *International Journal of Academic Information Systems Research (IJAISR)*, vol. 3, no. 7, pp. 1-23, 2019.
12. C. Nguyen et al., "Early Detection of Plant Viral Disease Using Hyperspectral Imaging and Deep Learning," *Sensors*, vol. 21, no. 3, p. 742, 2021. doi: 10.3390/s21030742.
13. G. Avola, A. Matese, and E. Riggi, "An Overview of the Special Issue on Precision Agriculture Using Hyperspectral Images," *Remote Sensing*, vol. 15, no. 7, p. 1917, 2023. doi: 10.3390/rs15071917.
14. W. Shafik, A. Tufail, C. De Silva Liyanage, and R. A. A. H. M. Apong, "Using Transfer Learning-Based Plant Disease Classification and Detection for Sustainable Agriculture," *BMC Plant Biology*, vol. 24, no. 1, p. 136, 2024. doi: 10.1186/s12870-024-04825-y.
15. Qazi, S., Khawaja, B. A., & Farooq, Q. U. (2022). IoT-equipped and AI-enabled next generation smart agriculture: A critical review, current challenges and future trends. *Ieee Access*, *10*, 21219-21235.
16. M. Shoaib et al., "Deep Learning-Based Segmentation and Classification of Leaf Images for Detection of Tomato Plant Disease," *Frontiers in Plant Science*, vol. 13, p. 1031748, 2022. doi: 10.3389/fpls.2022.1031748.

17. S. Badhan, K. Desai, M. Dsilva, R. Sonkusare, and S. Weakey, “Real-Time Weed Detection Using Machine Learning and Stereo-Vision,” in *2021 6th International Conference for Convergence in Technology (I2CT)*, 2021, pp. 1-5, IEEE.
18. Sharma, K., & Shivandu, S. K. (2024). Integrating artificial intelligence and internet of things (IoT) for enhanced crop monitoring and management in precision agriculture. *Sensors International*, 100292.
19. Nabi, F., Jamwal, S., & Padmanbh, K. (2022). Wireless sensor network in precision farming for forecasting and monitoring of apple disease: a survey. *International Journal of Information Technology*, *14*(2), 769-780.
20. AlZubi, A. A., & Galyna, K. (2023). Artificial intelligence and internet of things for sustainable farming and smart agriculture. *Ieee Access*, *11*, 78686-78692.
21. Jindo, K., Kozan, O., Iseki, K., Maestrini, B., van Evert, F. K., Wubengeda, Y., ... & Kempenaar, C. (2021). Potential utilization of satellite remote sensing for field-based agricultural studies. *Chemical and Biological Technologies in Agriculture*, *8*, 1-16.
22. Makondo, N., Kobo, H. I., Mathonsi, T. E., & Mamushiane, L. (2023, November). A review on edge computing in 5G-enabled IoT for agricultural applications: Opportunities and challenges. In *2023 International Conference on Electrical, Computer and Energy Technologies (ICECET)* (pp. 1-6). IEEE.
23. Chan, Y. K., Koo, V. C., Choong, E. H. K., & Lim, C. S. (2021). The drone based hyperspectral imaging system for precision agriculture.
24. Borra, P. (2024). Impact and Innovations of Azure IoT: current applications, services, and future directions. *Int. J. Recent Technol. Eng*, *13*(2), 2277-3878.
25. Varshney, R. P., & Sharma, D. K. (2022). Designing a cost-effective model leveraging serverless computing to provide weather forecasts to farmers in rural India. *International Journal of Information and Communication Technology*, *20*(4), 367-390.
26. Channe, H., Kothari, S., & Kadam, D. (2015). Multidisciplinary model for smart agriculture using internet-of-things (IoT), sensors, cloud-computing, mobile-computing & big-data analysis. *Int. J. Computer Technology & Applications*, *6*(3), 374-382.
27. Dineva, K., & Atanasova, T. (2022). Cloud data-driven intelligent monitoring system for interactive smart farming. *Sensors*, *22*(17), 6566.
28. Singh, K. K. (2018, November). An artificial intelligence and cloud based collaborative platform for plant disease identification, tracking and forecasting for farmers. In *2018 IEEE international conference on cloud computing in emerging markets (CCEM)* (pp. 49-56). IEEE.
29. Avalekar, U., Patil, D. J., Patil, D. S., Khot, P., & Prathapan, P. (2024). Optimizing agricultural efficiency: a fusion of Iot, AI, cloud computing, and wireless sensor network. *Prof.(Dr.) Kesava, Optimizing Agricultural Efficiency: A Fusion of Iot, Ai, Cloud Computing, and Wireless Sensor Network*.
30. Adewusi, A. O., Asuzu, O. F., Olorunsogo, T., Iwuanyanwu, C., Adaga, E., & Daraojimba, D. O. (2024). AI in precision agriculture: A review of technologies for sustainable farming practices. *World Journal of Advanced Research and Reviews*, *21*(1), 2276-2285.
31. Jafar, A., Bibi, N., Naqvi, R. A., Sadeghi-Niaraki, A., & Jeong, D. (2024). Revolutionizing agriculture with artificial intelligence: plant disease detection methods, applications, and their limitations. *Frontiers in Plant Science*, *15*, 1356260.

32. Dara, R., Hazrati Fard, S. M., & Kaur, J. (2022). Recommendations for ethical and responsible use of artificial intelligence in digital agriculture. *Frontiers in Artificial Intelligence*, *5*, 884192.
33. Adewusi, A. O., Chiekezie, N. R., & Eyo-Udo, N. L. (2022). Securing smart agriculture: Cybersecurity challenges and solutions in IoT-driven farms. *World Journal of Advanced Research and Reviews*, *15*(03), 480-489.
34. Shams, M. Y., Gamel, S. A., & Talaat, F. M. (2024). Enhancing crop recommendation systems with explainable artificial intelligence: a study on agricultural decision-making. *Neural Computing and Applications*, *36*(11), 5695-5714.
35. Saeed El Etaiby, Y. (2023). Smart agriculture and future technologies for achieving food security. *International Journal of Family Studies, Food Science and Nutrition Health*, *4*(1), 56-71.
36. Padol, P. B., & Yadav, A. A. (2016, June). SVM classifier based grape leaf disease detection. In *2016 Conference on advances in signal processing (CASP)* (pp. 175-179). IEEE.
37. Sujatha, R., Chatterjee, J. M., Jhanjhi, N. Z., & Brohi, S. N. (2021). Performance of deep learning vs machine learning in plant leaf disease detection. *Microprocessors and Microsystems*, *80*, 103615.
38. Ngugi, H. N., Akinyelu, A. A., & Ezugwu, A. E. (2024). Machine Learning and Deep Learning for Crop Disease Diagnosis: Performance Analysis and Review. *Agronomy*, *14*(12), 3001.
39. Masood, M. H., Saim, H., Taj, M., & Awais, M. M. (2020). Early disease diagnosis for rice crop. *arXiv preprint arXiv:2004.04775*.
40. Rahman, S. T., Vasker, N., Ahammed, A. K., & Hasan, M. (2023, December). Advancing Cucumber Disease Detection in Agriculture Through Machine Vision and Drone Technology. In *International Conference on Cyber Intelligence and Information Retrieval* (pp. 477-486). Singapore: Springer Nature Singapore.
41. Preanto, S. A., Ahad, M. T., Emon, Y. R., Mustofa, S., & Alamin, M. (2024). A Semantic Segmentation Approach on Sweet Orange Leaf Diseases Detection Utilizing YOLO. *arXiv preprint arXiv:2409.06671*.
42. Liu, J., & Wang, X. (2020). Tomato diseases and pests detection based on improved Yolo V3 convolutional neural network. *Frontiers in plant science*, *11*, 898.