AGRI_AI : ENHANCING FARM PRODUCTIVITY AND SUSTAINABILITY

A PROJECT REPORT

Submitted by

SOWMYA SURAPANENI [211420243053]

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

In

ARTIFICIAL INTELLIGENCE & DATA SCIENCE

PANIMALAR ENGINEERING COLLEGE

(An Autonomous Institution, Affiliated to Anna University,Chennai)
Accredited By National Board Of Accreditation (NBA)

MARCH 2024

PANIMALAR ENGINEERING COLLEGE

(An Autonomous Institution, Affiliated To Anna University)
Accredited By National Board Of Accreditation (NBA)
Bangalore Trunk Road, Varadharajapuram, Nasarathpet, Poonamallee, Chennai – 600 123

BONAFIDE CERTIFICATE

Certified that this project report **"AGRI_AI : ENHANCING FARM PRODUCTIVITY AND SUSTAINABILITY"** is the bonafide work of **"SOWMYA SURAPANENI"** who carried out the project work under my supervision.

SIGNATURE
DR. S. MALATHI ,M.E.,PH.D.,
HEAD OF THE DEPARTMENT,

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE,
PANIMALAR ENGINEERING COLLEGE,
CHENNAI-123

SIGNATURE
DR. S. MALATHI ,M.E.,PH.D.,
PROFESSOR AND HEAD,

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE,
PANIMALAR ENGINEERING COLLEGE,
CHENNAI-123

Certified that the above mentioned student was examined in Project Work(AD8811) held on _____

INTERNAL EXAMINER **EXTERNAL EXAMINER**

DECLARATION BY THE STUDENT

I **SOWMYA SURAPANENI [211420243053],** hereby declare that this project report titled **"AGRI_AI : ENHANCING FARM PRODUCTIVITY AND SUSTAINABILITY"**, under the guidance of **DR. S. MALATHI ,M.E.,PH.D.,** is the original work done by me and I have not plagiarized or submitted to any other degree in any university by me.

ACKNOWLEDGEMENT

We would like to express our deep gratitude to our respected Secretary and Correspondent **Dr. P. CHINNADURAI, M.A., Ph.D.** for his kind words and enthusiastic motivation, which inspired us a lot in completing this project.

We express our sincere thanks to our directors **Tmt. C. VIJAYARAJESWARI**, **Dr. C. SAKTHI KUMAR, M.E., Ph.D. and Dr. SARANYASREE SAKTHI KUMAR B.E., M.B.A., Ph.D.,** for providing us with the necessary facilities to undertake this project.

We also express our gratitude to our Principal **Dr. K. MANI, M.E., Ph.D**. who facilitated us in completing the project.

We thank the Head of the AI&DS Department, **Dr. S. MALATHI, M.E., Ph.D.,** for the support extended throughout the project.

We would like to thank my Project Guide , **Dr. S. MALATHI, M.E., Ph.D.,** and all the faculty members of the Department of AI&DS for their advice and encouragement for the successful completion of the project.

SOWMYA SURAPANENI

ABSTRACT

Agri_AI is an innovative solution that leverages advanced Geographic Information System (GIS) technology to revolutionize the field of agriculture. By integrating precision crop planning, input tracking, AI-powered pest management, real-time weather updates, smart irrigation, and powerful data analytics, Agri_AI provides farmers with a comprehensive toolkit to optimize their farming techniques. One of the core features of Agri_AI is its GIS field mapping system, enabling farmers to accurately map and monitor their fields in real-time, facilitating better decision-making and resource allocation. With precision crop planning, farmers can maximize yields by strategically planning the planting of crops based on soil conditions, weather patterns, and other relevant parameters. Input tracking functionality allows farmers to effectively monitor and manage the usage of fertilizers, pesticides, and other resources, enabling cost savings and minimized environmental impact. The AI-powered pest management module utilizes advanced algorithms to detect pest infestations early, providing timely recommendations for effective pest control measures. Real-time weather updates keep farmers informed about current and upcoming weather conditions, allowing for proactive decision-making and minimizing weather-related risks. Smart irrigation technology enables farmers to optimize water usage based on crop requirements and soil moisture measurements, promoting water conservation while maximizing crop productivity. Finally, the powerful data analytics component of Agri_AI provides comprehensive insights and trends, helping farmers identify patterns, optimize farming practices, and make data-driven decisions.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

Abbreviation	Meaning
GIS	Geographic Information System
AI	Artificial Intelligence
IOT	IoT - Internet of Things
GPS	Global Positioning System
PMP	Precision Crop Mapping
ITT	Input Tracking Technology
WU	Weather Updates
SI	Smart Irrigation
DA	Data Analytics
PM	Pest Management
CV	Computer Vision
Agri_AI	Name of Application

CHAPTER 1

INTRODUCTION

CHAPTER 1

INTRODUCTION

1.1 GENERAL

Agri_AI is an advanced agricultural platform that revolutionizes the way farmers manage their operations. With its comprehensive suite of tools and features, Agri_AI empowers farmers to leverage the power of geographic information system (GIS) field mapping, precision crop planning, input tracking, AI-powered pest management, real-time weather updates, smart irrigation, and powerful data analytics for optimized farming. At the core of Agri_AI is its GIS field mapping capability, which enables farmers to accurately map and visualize their fields, ensuring that they have a clear understanding of their land boundaries, terrain variations, and other relevant information. By having this level of spatial awareness, farmers can make informed decisions about planting, fertilizing, and harvesting, leading to improved productivity and sustainability. Precision crop planning is another key feature offered by Agri_AI, as it allows farmers to optimize yield potential through precise seed placement and appropriate crop rotations. By analyzing soil composition, moisture levels, and other environmental factors, farmers can strategically determine the best crop varieties to plant in specific areas of their fields, maximizing their overall productivity while minimizing resource waste.

Tracking inputs is critical for efficient farm management, and Agri_AI offers a comprehensive input tracking system that enables farmers to monitor the usage of fertilizers, pesticides, and other agricultural inputs. By keeping track of these inputs, farmers can ensure that they adhere to environmental regulations and optimize their resource allocation, leading to cost savings and reduced environmental impact. The AI-powered pest management feature of Agri_AI is a game-changer in the battle against crop-damaging pests. By leveraging advanced machine learning algorithms, the platform can detect and identify different types of pests and diseases, allowing farmers to take

immediate action to mitigate the damage. Through proactive pest management, farmers can protect their crops and minimize yield losses, ultimately boosting profitability.

Real-time weather updates and smart irrigation are essential components of Agri_AI, providing farmers with crucial insights into current and forecasted weather conditions. By integrating with local weather stations and remote sensors, the platform enables farmers to make data-driven decisions regarding irrigation, ensuring that crops receive the optimal amount of water while conserving resources. Last but not least, Agri_AI's powerful data analytics capabilities allow farmers to transform their data into actionable insights, providing them with a competitive edge. By analyzing historical and real-time data on yields, inputs, weather patterns, and other variables, farmers can make evidence-based decisions to optimize their farming practices, improve crop quality, and increase profitability.

In conclusion, Agri_AI is a comprehensive agricultural solution that leverages GIS field mapping, precision crop planning, input tracking, AI-powered pest management, real-time weather updates, smart irrigation, and powerful data analytics to unlock the full potential of modern farming. By empowering farmers with these advanced tools and capabilities, Agri_AI revolutionizes the way farms are managed, leading to increased productivity, sustainability, and profitability in the agricultural industry.

1.2 PROBLEM DESCRIPTION

Agri_AI faces the challenge of modernizing and optimizing farming practices to meet the growing global demand for food while minimizing resource consumption and environmental impact. Traditional farming methods often lack precision and efficiency, leading to overuse of resources and increased pest-related losses.

Agri_AI aims to address these issues by employing advanced technologies such as GIS field mapping, precision crop planning, and AI-powered pest management. However, integrating these technologies into existing agricultural systems can be complex and costly. Ensuring seamless real-time data integration and analysis while maintaining user-friendly interfaces poses a significant challenge.

Additionally, adapting to changing weather patterns and optimizing irrigation to conserve water resources is crucial. Agri_AI must continually update its weather monitoring and smart irrigation systems to stay effective in the face of climate variability.

Ultimately, the success of Agri_AI hinges on its ability to provide farmers with actionable insights and tools that improve crop yield, resource efficiency, and profitability in a sustainable manner.

1.3 OBJECTIVE

The primary objective of Agri_AI is to revolutionize modern agriculture by harnessing cutting-edge technology to optimize farming practices. Our mission is to empower farmers with advanced tools and solutions to address the following key objectives:

1. Precision Farming: Agri_AI aims to enable precise and data-driven farming practices through GIS field mapping and precision crop planning. This ensures optimal land usage and resource allocation.
2. Pest Management: By utilizing AI-powered pest management systems, Agri_AI seeks to minimize crop losses caused by pests while reducing the reliance on chemical pesticides, promoting sustainable agriculture.
3. Resource Efficiency: We strive to enhance resource efficiency by providing real-time weather updates and smart irrigation solutions, enabling farmers to conserve water and energy resources while improving crop yields.
4. Data-Driven Insights: Agri_AI's powerful data analytics tools are designed to offer farmers actionable insights into their operations, helping them make informed decisions and increase overall farm productivity.
5. Sustainability: Our overarching goal is to promote environmentally sustainable farming practices, reducing the ecological footprint of agriculture and contributing to food security on a global scale.

Agri_AI is committed to revolutionizing agriculture for a more sustainable, efficient, and productive future.

1.4 EXISTING SYSTEM

The existing system which incorporates advanced technology for optimized farming, does have a few disadvantages. Firstly, the implementation of such a comprehensive system requires a significant initial investment in terms of hardware, software, and skilled personnel, which may pose a financial challenge for small-scale farmers or those with limited resources. Additionally, the complexity of the system may require farmers to undergo extensive training to fully understand and utilize all its features effectively. This can be time-consuming and may hinder the adoption of the system by farmers who are already burdened with numerous tasks on the farm. Furthermore, the system relies heavily on technology and connectivity, which may be unreliable in rural areas or regions with poor network coverage. This can lead to disruptions in data acquisition, communication between devices, and access to real-time updates, potentially compromising the accuracy and timeliness of decision-making processes. Another drawback is the reliance on AI-powered pest management, which may not be suitable for all types of pests or farming practices. Traditional methods may still be necessary in certain cases, and the exclusion of non-AI approaches could limit the effectiveness of pest control strategies. Lastly, while smart irrigation can help conserve water by providing plants with the right amount at the right time, it may also require farmers to be more dependent on technology and automated systems. This raises concerns regarding the sustainability and resilience of farming practices in the event of system failures or power outages. Despite these disadvantages, with proper planning, training, and support, the implementation of the Agri_AI system has the potential to revolutionize farming practices and enhance productivity, sustainability, and profitability in the agricultural sector.

1.5 PROPOSED SYSTEM

Agri_AI offers a comprehensive solution for advanced GIS field mapping, precision crop planning, input tracking, AI-powered pest management, real-time weather updates, smart irrigation, and powerful data analytics to optimize farming. With advanced GIS field

mapping, farmers can accurately map their fields, identify boundaries, and efficiently plan their planting and harvesting activities. Precision crop planning ensures that farmers maximize their yield potential by deploying the most suitable crops and varieties for their fields based on soil conditions, weather patterns and market demand.

Moreover, Agri_AI provides input tracking capabilities, enabling farmers to closely monitor and manage their use of fertilizers, pesticides and other key inputs. This helps to reduce waste, ensure sustainability, and improve cost efficiency. AI-powered pest management takes precision farming to the next level by using machine learning algorithms to detect and identify pest infestations early on. This proactive approach allows farmers to take swift and targeted action, minimizing the impact of pests on their crops.

Real-time weather updates play a crucial role in making informed decisions on crop management. Agri_AI integrates with meteorological data sources to provide farmers with up-to-date information on temperature, humidity, rainfall and other key weather parameters. This information enables farmers to adjust irrigation schedules, optimize pesticide applications, and take appropriate measures to protect their crops from extreme weather events. In addition, Agri_AI incorporates smart irrigation technology, which uses sensors and data analytics to determine precise irrigation needs based on soil moisture levels, crop water requirements, and weather conditions. This ensures efficient water usage, minimizes water waste, and promotes sustainable farming practices.

Agri_AI’s powerful data analytics capabilities bring all the collected data together to generate actionable insights and recommendations. By analyzing data on crop performance, input usage, weather patterns, and other relevant factors, farmers can make data-driven decisions to optimize their farming operations, increase productivity, and enhance profitability. The integration of artificial intelligence further enhances the system's ability to continuously learn and improve recommendations over time. Overall, Agri_AI offers a complete and integrated solution for farmers to embrace precision farming and harness the power of advanced technology for optimized farming practices.

1.6 KEY FUNCTIONALITIES

Based on the proposed method and the analysis provided, the key functionalities of your precision agriculture project could include:

1. GIS-based Field Mapping and Boundary Delineation:
    - Integration of GIS software or tools for accurate mapping of agricultural fields
    - Ability to delineate field boundaries and capture spatial data
2. AI-driven Crop Planning and Input Allocation:
    - Implementation of AI algorithms and machine learning models
    - Analysis of historical and real-time data (weather, soil, crop performance, etc.)
    - Decision support for selecting suitable crop varieties, determining planting dates, and optimizing input allocation (fertilizers, pesticides, etc.)
3. AI-powered Pest Management System:
    - Utilization of AI algorithms and computer vision techniques
    - Real-time monitoring and identification of pest infestations
    - Recommendation of appropriate pest control measures and interventions
4. Weather Monitoring and Integration:
    - Deployment of weather sensors for real-time data collection
    - Integration of weather data (temperature, humidity, precipitation, wind patterns) into crop planning and irrigation decisions
5. Computer Vision-based Smart Irrigation Control:
    - Implementation of computer vision algorithms for monitoring crop health and soil moisture levels
    - Automated control of irrigation systems based on real-time data
    - Efficient water usage and prevention of over-or-under-watering
6. Data Analytics and Decision Support Platform:
    - Central platform for aggregating and analyzing data from various sources (GIS, AI models, weather sensors, computer vision)

- Data visualization and reporting tools
- Generation of actionable insights and recommendations for farmers
- Decision support for optimizing farm operations, resource allocation, and crop management

7. User Interface and Farmer Interactions:
    - User-friendly interface for farmers to interact with the system
    - Access to real-time data, insights, and recommendations
    - Ability to input farm-specific information and preferences
    - Integration with mobile devices or web applications for remote access and monitoring
8. Data Integration and Interoperability:
    - Seamless integration and data exchange between different components (GIS, AI models, weather sensors, irrigation systems)
    - Adherence to industry standards and protocols for data formats and communication
9. Scalability and Extensibility:
    - Ability to scale the system to accommodate larger farm sizes or multiple farm operations
    - Modular design to allow for easy integration of new technologies or components as they become available
10. Security and Privacy Measures:
    - Implementation of appropriate security measures to protect sensitive farm data
    - Adherence to data privacy regulations and best practices

CHAPTER 2

LITERATURE SURVEY

CHAPTER 2

LITERATURE SURVEY

2.1 LITERATURE REVIEW

This literature survey covers a wide range of research papers exploring the applications of advanced technologies, such as geospatial and big data, Artificial Intelligence (AI), Internet of Things (IoT), machine learning, and remote sensing, in the field of smart farming and precision agriculture. The papers highlight the potential of these technologies in optimizing crop management, resource utilization, sustainability, and decision-making processes in agricultural operations.

Papers [1], [2], and [3] discuss the integration of geospatial and big data technologies, AI, and IoT in smart farming and precision agriculture. These papers emphasize the transformative potential of these technologies in addressing challenges faced by developing nations in agriculture, enabling data-driven decision-making, automation, and real-time monitoring.

The research focuses on the effective applications of remote sensing, UAVs/drones, and machine learning in precision agriculture, as discussed in [4]. Additionally, [5] explores the role of cloud-based IoT solutions in enhancing agricultural sustainability and efficiency, enabling real-time data collection, analysis, and remote monitoring. [6] and [7], Provides a comprehensive analysis of the advances in digital agricultural architecture in India and the concept of "Smart Agricultural Technology" within the context of precision agriculture.

[8] and [9] delve into advanced technologies and AI-enabled IoT applications in high-tech agriculture, as well as machine learning-enabled IoT solutions for smart agriculture operations. These papers explore the use of data analytics, predictive modeling, and automation to optimize farming practices and increase yields.

The advancements, technologies, and applications in precision agriculture for improved crop productivity and resource management, highlighting the integration of data-driven decision-making, sensor technology, automation, and remote monitoring[10].

The implementation of automation in agriculture, including irrigation and the application of pesticides and herbicides through robots and drones, and [11],[12], which proposes a long-range, real-time, scalable solution for field irrigation monitoring based on soil and weather conditions.A literature review [13] that examines smart irrigation monitoring and control strategies for improving water use efficiency in precision agriculture, arguing for the combination of soil-based, plant, and weather-based monitoring methods with model predictive control.[14] Reviews the diverse applications of Geographic Information Systems (GIS) in agriculture, from land-use planning to crop-soil-yield monitoring and post-harvest operations, enabling the realization of precision farming and sustainable food production goals. [15] Introduces the crop yield management process and its components, identifying the main categories of data mining techniques for crop yield monitoring and discussing the impact of big data on agriculture.[16] Utilizes literature- and NDVI-measurement-based successor crop suitability matrices and crop-specific attributes as input for training a reinforcement learning agent to generate crop rotation sequences, validated by practitioners and experts.[17]Explores precision agriculture as a modern technology for crop management, utilizing remote sensing, GIS, GPS, and IoT devices to optimize agricultural processes and enhance crop productivity.

[18] Aims to discover the best model for crop prediction using machine learning algorithms such as K-Nearest Neighbor (KNN), Decision Tree, and Random Forest Classifier, to help farmers decide on the type of crop to grow based on climatic conditions and soil nutrients.[19]Reviews time series algorithms and suggests appropriate crop recommendations using machine and deep learning by estimating crop yield through time series analysis to reduce food insufficiency in the future.

[21]There exists a system that proposes a pest identification model named MADN, which is based on a large-scale pest dataset named HQIP102. The model uses a densely connected convolutional network to improve classification accuracy.[22] This paper proposes an efficient pest detection method that accurately localizes pests and classifies them according to their desired category. The method uses a deep learning-based approach and a medium-scale benchmark dataset.[23] This paper focuses on tea disease leaves for image classification research, using a combination of convolution, iterative module, and embedded iterative region of interest encoding transformer.[24] This study aims to classify and detect insects in corn, soybean, and wheat using machine learning and insect pest detection algorithms at an early stage.[25] This paper proposes a residual convolutional neural network for pest identification based on transfer learning. The IP102 agricultural pest dataset is used for training and testing.[26] This paper provides a comprehensive review of the state of the art of pest monitoring using digital images and machine learning. It discusses the main weaknesses and research gaps that still remain and proposes possible directions for future research.[27] This paper proposes a vision-based pest detection method based on the support vector machine (SVM) classification method. The method is tested on a dataset of 1,000 images of pests and non-pests.[28] This paper proposes an automatic plant pest detection and recognition method using the K-means clustering algorithm and correspondence filters. The method is tested on a dataset of 100 images of pests and non-pests.[29] This paper proposes a tea insect pest classification method based on artificial neural networks. The method is tested on a dataset of 1,000 images of tea insect pests.[30] This paper proposes a fast and accurate detection and classification method for plant diseases. The method is based on a combination of image processing techniques and machine learning algorithms and is tested on a dataset of 1,000 images of plant diseases.

Finally, [20], which introduces the technical architecture and user-in-the-loop (UIL) principle of an expert system for AI-based crop rotation for sustainable agriculture worldwide, arguing for the system's user interface to broaden thinking habits and opting for new sustainable farming perspectives.This reference explores how geospatial and big

data technologies are being harnessed in smart farming to address the unique challenges faced by developing nations in agriculture. It provides insights into the practical applications of these technologies, emphasizing their potential in improving crop management, resource allocation, and decision-making processes. The chapter highlights the significance of leveraging data-driven approaches to enhance agricultural practices, particularly in regions where agriculture plays a vital role in food security and economic development.

2.2 EXISTING SYSTEM FOR COMPARISION

Traditional farming methods that rely on manual field mapping, intuitive crop planning, limited pest management techniques, and basic weather and irrigation monitoring.

Crop Yield Improvement (%):

- The proposed method's use of AI algorithms for crop planning, optimal input allocation, and real-time pest management is likely to result in higher crop yields compared to traditional farming methods.
- AI-driven decision-making can help identify the most suitable crop varieties, planting times, and input quantities, maximizing yield potential.
- Effective pest management enabled by AI can minimize crop losses due to pest infestations.

Water Usage Efficiency (%):

- The integration of weather sensors and computer vision for smart irrigation control in the proposed method can significantly improve water usage efficiency.
- Real-time monitoring of soil moisture levels and crop water requirements can prevent over-irrigation and water wastage.
- Traditional irrigation methods often rely on subjective judgments or fixed schedules, leading to inefficient water usage.

Reduction in Pest-Related Losses (%):

- The proposed method's AI-powered pest management system can significantly reduce pest-related losses compared to traditional methods.
- Early detection and timely intervention enabled by AI can prevent widespread infestations and minimize crop damage.
- Traditional pest management techniques may be less effective in identifying and addressing pest issues in a timely manner.

Resource Utilization Efficiency (%):

- The proposed method's use of GIS for precise field mapping, AI algorithms for input allocation, and data analytics for decision support can optimize resource utilization.
- By accurately identifying field variability and crop requirements, resources like fertilizers, pesticides, and labor can be optimized.
- Traditional farming methods may struggle with efficient resource allocation due to a lack of precise data and decision support tools.

Increase in Profit Margin (%):

- The proposed method's potential to improve crop yields, reduce pest-related losses, optimize resource utilization, and increase water usage efficiency can collectively contribute to an increase in profit margins.
- Higher yields, lower input costs, and minimized losses can translate into higher profitability for farmers.
- Traditional farming methods may have lower profit margins due to inefficiencies in resource utilization, pest management, and suboptimal yields.

CHAPTER 3

HARDWARE AND SOFTWARE SPECIFICATION

3.1 HARDWARE ENVIRONMENT

- Processor : min i5 or more
- Hard disk : min 16 GB
- Ram : 4 GB or above
- Monitor resolution : No Specification (any resolution)

3.2 SOFTWARE ENVIRONMENT

Operating System : Any(Windows 10 / 8, Linux, Mac)

Languages used :

- PYTHON
- HTML,CSS,JS

Tools used :

- VS CODE
- STREAMLIT UI
- GOOGLE COLAB

CHAPTER 4

PROPOSED SYSTEM DESIGN

CHAPTER 4

PROPOSED SYSTEM DESIGN

4.1 DATA FLOW DIAGRAM

4.1.1 ZERO LEVEL DFD

The diagram (4.1.1) illustrates the core components and functionalities of the AGRI_AI system, which is designed to provide comprehensive agricultural solutions.

The central circle represents the AGRI_AI system itself, which acts as a hub for integrating and processing various types of data inputs. The arrows pointing towards the circle indicate the different data sources that feed into the system.

Field Coordinates: AGRI_AI accepts field coordinate data, which likely includes geospatial information and mapping details of the agricultural fields or areas under cultivation.

Weather: The system incorporates weather data, which could include meteorological factors such as temperature, precipitation, humidity, wind speed, and forecasts. This information is crucial for making informed decisions about crop planning, irrigation, and other agricultural activities.

Crop Details: AGRI_AI takes into account specific details about the crops being cultivated, such as crop types, varieties, growth stages, and any relevant crop-specific data.

Irrigation Details: The system integrates irrigation-related data, which may include information about water sources, irrigation systems, soil moisture levels, and water usage patterns.

The arrows pointing outwards from the AGRI_AI circle represent the outputs or services provided by the system:

Crop Planning & Analysis: Based on the integrated data inputs, AGRI_AI provides crop planning and analysis capabilities, allowing users to make informed decisions about crop selection, planting schedules, and optimizing agricultural practices.

Reports & Alerts: The system generates reports and alerts to inform users about various aspects of their agricultural operations, such as crop health, irrigation requirements, weather impact, and potential issues or opportunities.

Overall, the diagram depicts AGRI_AI as a centralized platform that aggregates and analyzes diverse data sources related to agriculture, enabling users to make data-driven decisions and optimize their farming practices through crop planning, analysis, and reporting functionalities.

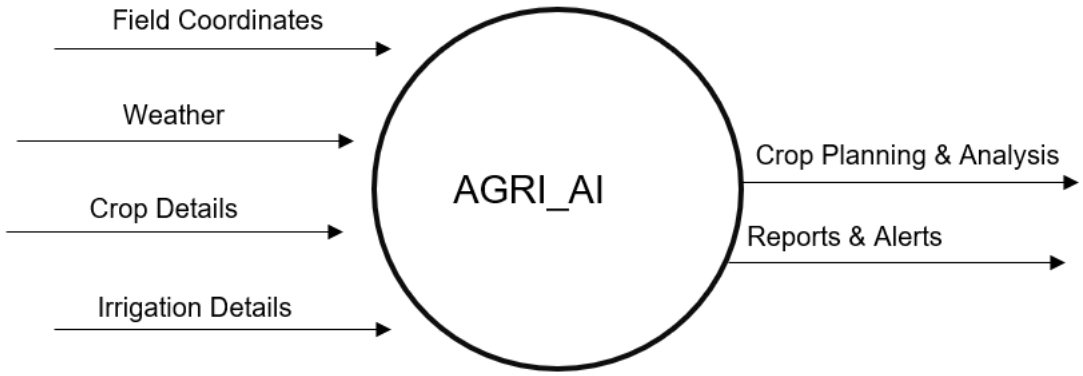

Fig 4.1.1 0-Level Data Flow Diagram of Agri_AI

4.1.2 1-LEVEL DFD

This first-level DFD (Fig 4.1.2) provides an overview of the main processes involved in the agricultural management system and how data flows between them, highlighting the system's integration of various data sources, analytical tools, and decision support functionalities.

External Entities:

Field Inputs: This entity represents the various data sources from agricultural fields, such as soil sensors, weather stations, and crop monitoring devices.

Processes:

Crop History Using Data Analytics: This process analyzes historical crop data using data analytics techniques to derive insights about past performance, yield trends, and other relevant factors.

Irrigation Management: This process handles the management and optimization of irrigation activities based on field inputs and data analysis.

Risk Analysis using Weather Tracking: This process continuously monitors and analyzes weather data to perform risk analysis and identify potential threats or opportunities related to weather conditions.

Geo Information System Mapping: This process involves geospatial mapping and the use of geographic information systems to provide spatial data and visualizations of agricultural fields.

Pest Management: This process focuses on detecting, monitoring, and mitigating pest infestations or diseases using field inputs and data analysis.

Crop Planning: This process utilizes inputs from data analytics, irrigation management, risk analysis, and other factors to develop informed crop planning strategies.

Data Stores:

Crop History: This data store contains historical crop data used for analysis by the "Crop History Using Data Analytics" process.

Data Flows:

The diagram shows the flow of data between the external entity (Field Inputs), the processes, and the data store (Crop History). The arrows indicate the direction of data movement.

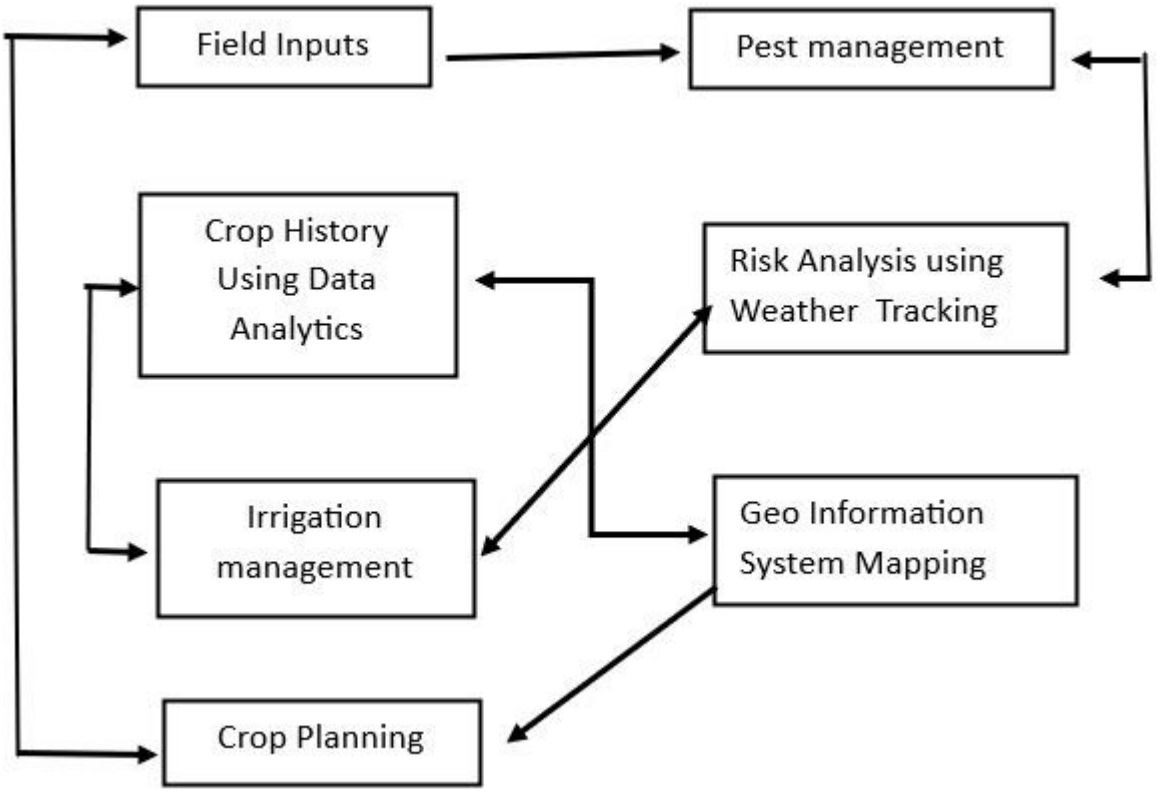

Fig 4.1.2 1-Level Data Flow Diagram of Agri_AI

4.2 METHODOLOGIES

Agri_AI is an advanced agricultural technology system designed to revolutionize farming practices by integrating various cutting-edge components. It encompasses the following key features:

1. Advanced GIS Field Mapping: Agri_AI utilizes Geographic Information System (GIS) technology to create detailed and accurate maps of farmland. These maps provide crucial insights into soil quality, topography, and land characteristics, aiding in precise decision-making.

2. Precision Crop Planning: The system offers precision crop planning tools that leverage data from GIS mapping. It assists farmers in optimizing crop selection and planting based on soil conditions, climate, and market demand.

3. Input Tracking: Agri_AI enables farmers to track and manage agricultural inputs such as seeds, fertilizers, and pesticides efficiently. This ensures optimal resource allocation and minimizes wastage.

4. AI-Powered Pest Management: The system employs Artificial Intelligence (AI) for pest detection and management. It uses data from various sources to predict and respond to pest outbreaks, reducing crop losses.

5. Real-Time Weather Updates: Agri_AI provides farmers with real-time weather information, allowing them to make informed decisions regarding irrigation, planting, and harvesting, mitigating weather-related risks.

6. Smart Irrigation: It includes smart irrigation solutions that adjust water usage based on weather conditions and crop needs, conserving water resources and promoting sustainable farming.

7. Powerful Data Analytics: The system employs robust data analytics tools to process information from multiple sources, providing farmers with actionable insights for optimizing farming practices.

4.3 SYSTEM FLOW

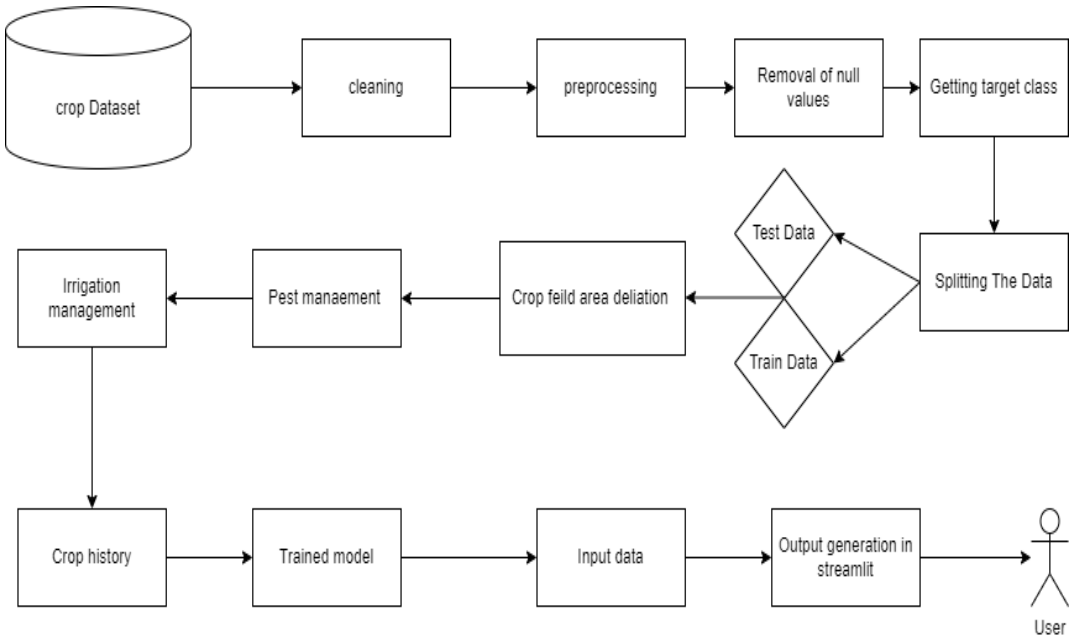

Fig 4.3 Block diagram of the proposed system.

The system diagram 4.3 illustrates the integration of data cleaning, preprocessing, pest management, field mapping, irrigation optimization, and machine learning techniques to provide accurate and informed decisions for precision agriculture practices, ultimately aiming to improve crop yields and resource utilization efficiency.

4.4 USECASE DIAGRAM

A use case diagram 4.4 is used to represent the dynamic behavior of a system. It encapsulates the system's functionality by incorporating use cases, actors, and their relationships. It models the tasks, services, and functions required by a system/subsystem of an application. It depicts the high-level functionality of a system and also tells how the user handles a system. The main purpose of a use case diagram is to portray the dynamic aspect of a system. It accumulates the system's requirement, which includes both internal as well as external influences. It invokes persons, use cases, and several things that invoke the actors and elements accountable for the implementation of use case diagrams. It

represents how an entity from the external environment can interact with a part of the system.

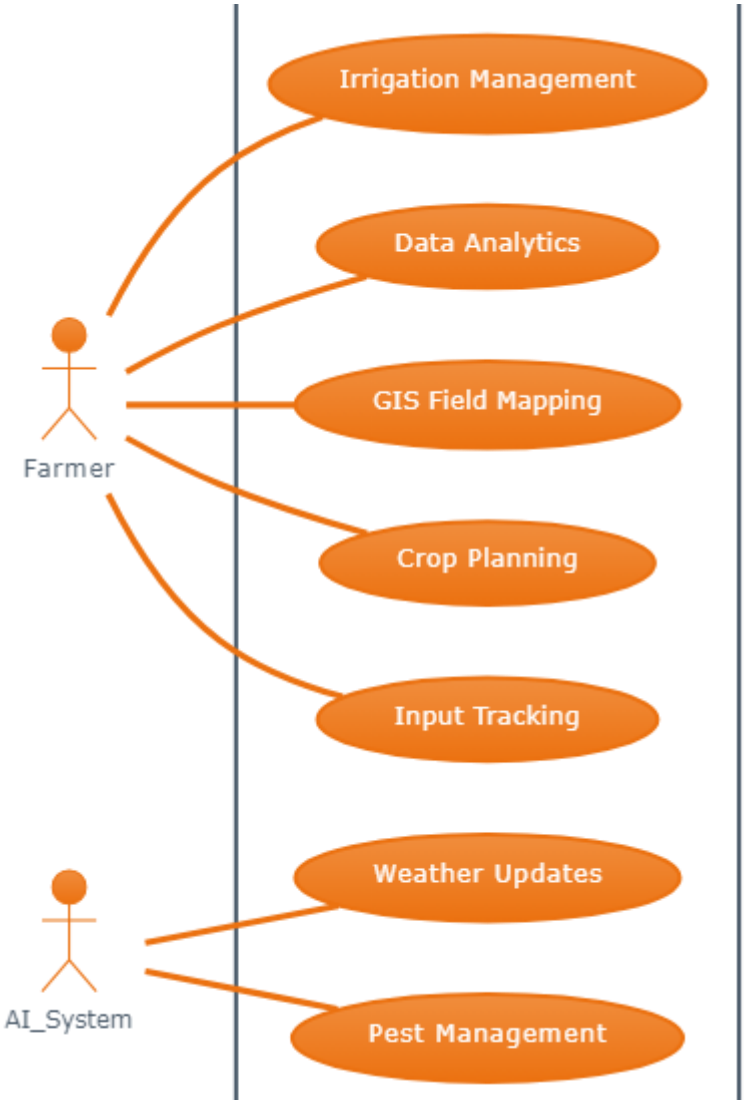

Fig 4.4 Use case diagram of the proposed system.

4.5 ACTIVITY DIAGRAM

An activity diagram 4.5 is a kind of graphical representation that may be used to depict events visually. It is made up of a group of nodes that are linked to one another by means of edges. They are able to be connected to any other modelling element, which enables the behaviour of activities to be replicated using that methodology. Simulations of use cases, classes, and interfaces, as well as component collaborations and component interactions, are all made feasible with the help of this tool.

Fig 4.5 Activity diagram of the proposed system.

4.6 MODULE ARCHITECTURE

The architecture diagram 4.6 presents a comprehensive system for intelligent irrigation control and monitoring, integrating various components such as weather data, AI analysis, irrigation control, data processing, and user interfaces.

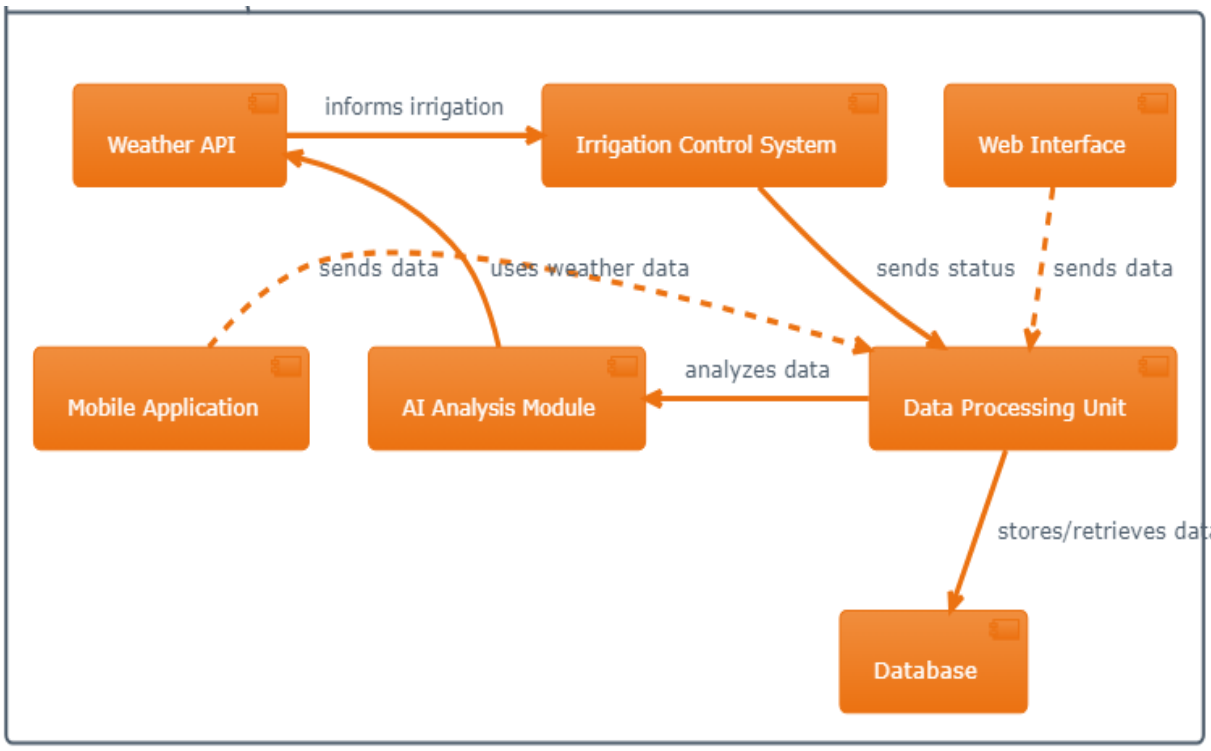

Fig 4.6 Architecture diagram of the proposed system.

4.7 ARCHITECTURE

This architecture diagram (Fig 4.7) presents a comprehensive system for precision agriculture and crop management. Here are a few insights:

- Field mapping and area mapping play a crucial role, utilizing geospatial data from sources like GeoJSON files, field coordinates, and drag-and-drop interfaces to accurately delineate field boundaries and generate field insights.
- Crop details, soil details, crop rotation, and other field characteristics are integrated into the area mapping process, enabling predictive analytics for optimizing planting cycles and harvest time frames.
- Various data sources are leveraged, including weather analysis, soil data, sensor data, crop data, field data, and pest data, providing a comprehensive view of the agricultural environment.
- Open CV and data analytics techniques are employed to process and analyze the collected data, generating reports and alerts for informed decision-making.

- The architecture incorporates data processing and formatting components, as well as AI algorithms (specifically mentioned as EfficientNetV2-L), to extract valuable insights and support visual analytics tools, likely through a user interface like STREAMLIT UI.

Overall, this architecture aims to integrate diverse data sources, leverage geospatial mapping, predictive analytics, and AI algorithms to optimize agricultural practices, support data-driven decision-making, and provide visual analytics for better crop management.

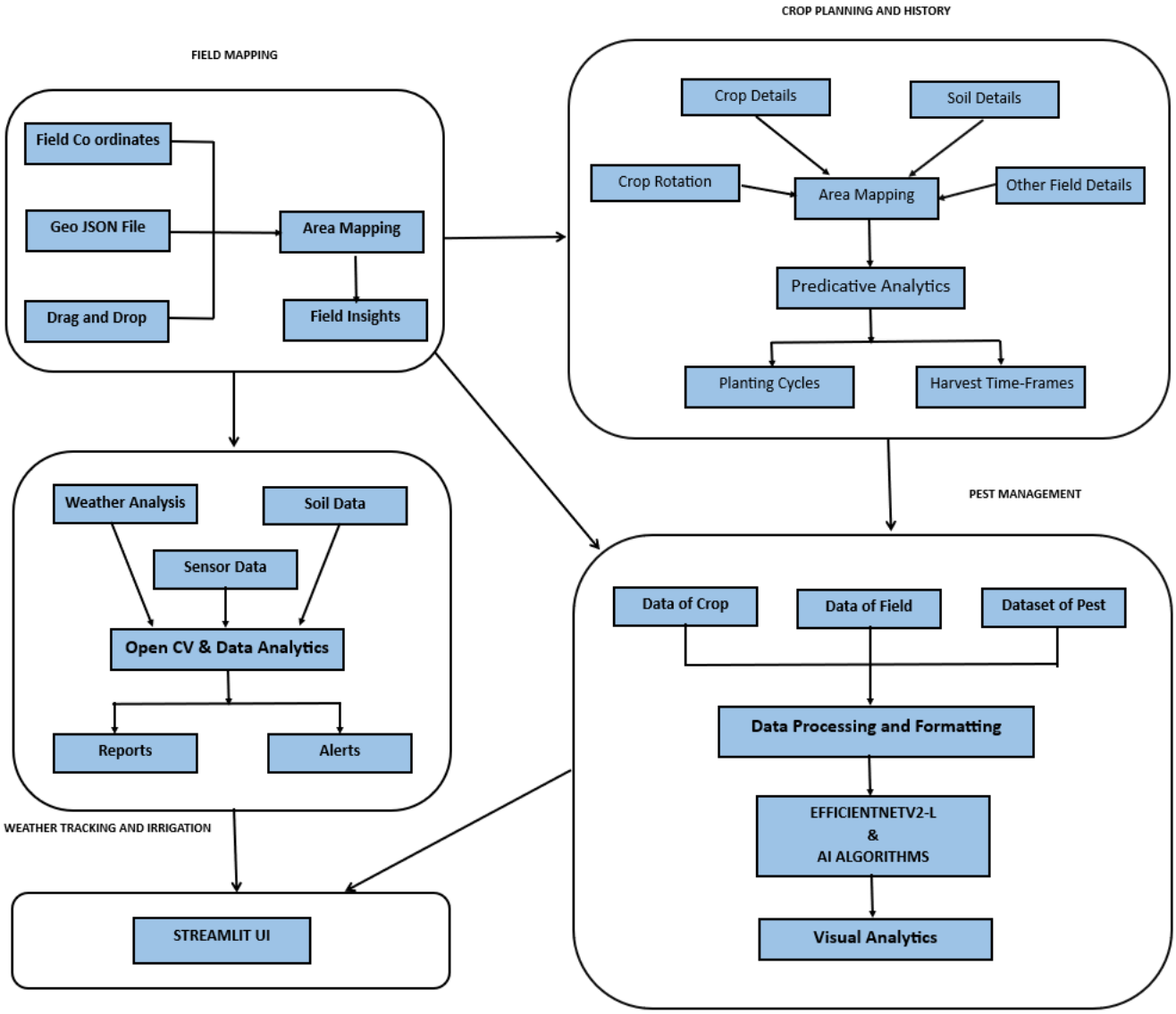

Fig 4.7 System Architecture

CHAPTER 5

PROJECT WORK IMPLEMENTATION

CHAPTER 5
PROJECT WORK IMPLEMENTATION

5.1 MODULE DESCRIPTION

This system consists of six important modules which includes:

- Field Mapping
- Crop Planning
- Crop History
- Pest Management
- Weather tracking
- Irrigation System

5.1.1 Advanced Precision Agriculture Field Analysis and Prescription Mapping Utilizing Geographic Information Systems

This module leverages geographic information systems (GIS) and precision agriculture technologies to create detailed maps of farmland. High-resolution satellite imagery and sensor data will map soil properties, crop health indices, drainage, and yield variability within fields. Advanced analytics of these geospatial data layers over time can provide valuable insights to guide variable rate input prescriptions and improve farm efficiency.

PSEUDOCODE 1	Field Mapping
Step 1	Define a function field_mapping()
Step 2	Display a header with the title "MAPPING"
Step 3	Create a sidebar section with the title "Enter GPS Coordinates"
Step 4	Take user input for latitude and longitude coordinates using number input fields
Step 5	Create a folium map centered at the input coordinates with a starting zoom level of 13
Step 6	Add a drawing tool (plugins.Draw) to the map, allowing the user to draw shapes
Step 7	Display the folium map in the Streamlit app using folium_static()
Step 8	Create a sidebar section with the title "Upload
Step 9	GeoJSON Implement a file uploader for the user to upload a GeoJSON file
Step 10	If a file is uploaded: a. Load the GeoJSON data from the uploaded file b. Initialize a counter i to 0 c. Iterate through each feature in the GeoJSON data i. If the feature geometry type is "Polygon": 1. Extract the coordinates of the polygon vertices 2. Flip the coordinate order (lon, lat) to (lat, lon) 3. If there are at least 3 vertices: a. Create a Polygon object using the vertices b. Calculate the area of the polygon c. Display a header with the area of the field 4. Iterate through the vertices (except the last one) a. Calculate the geodesic distance between the current vertex and the next vertex

	b. **: :** Display a header with the distance between the vertices
Step 11	End of function

The above pseudocode outlines the steps involved in creating a Streamlit app that allows the user to input GPS coordinates, draw shapes on a map, upload a GeoJSON file, and calculate the area and distances between vertices of polygons contained in the GeoJSON data.

5.1.2 Data-Driven Predictive Modeling and Forecasting for Optimized Crop Cycles, Rotations, and Harvest Timelines to Maximize Agricultural Productivity

This module applies predictive analytics to optimize crop planning. Historical data will allow fine-tuned modeling of planting schedules, crop rotations, and harvesting timelines. Advanced crop cycle forecasts and prescriptive insights aim to boost agricultural productivity across farm operations.

PSEUDOCODE 2	**Crop Planning**
Step 1	Define a function crop_planning()
Step 2	Display a header with the title "Crop Planning"
Step 3	Create a sidebar section with the title "Inputs"
Step 4	Get the number of crops from the user using a number input field
Step 5	Create empty lists to store crop data (crop names, areas, pod counts, grain counts, grain weights, and rainfall data)
Step 6	For each crop: a. Get the crop name from the user using a text input field b. Get the crop area from the user using a number input field c. Get the average pod count from the user using a number input field d. Get the average grain

	count from the user using a number input field e. Get the grain weight per grain from the user using a number input field f. Get the months for which the user wants to provide rainfall data using a multiselect field g. For each selected month: i. Get the rainfall data from the user using a number input field h. Append the crop data (name, area, pod count, grain count, grain weight, and rainfall data) to the respective lists
Step 7	Perform crop planning calculations (e.g., crop yields) using the collected data
Step 8	Display crop planning results a. Create a pandas DataFrame with crop names and yields b. Display the DataFrame as a table c. Create a pie chart to compare crop yields d. Create a bar chart for individual crop yields e. Create a line chart for monthly rainfall data
Step 9	Display images a. Create a file uploader for each crop to upload images b. Display the uploaded images in a grid layout

The above pseudocode 2 defines a function crop_planning() that creates a user interface for crop planning using the Streamlit library. It collects various input data from the user, such as the number of crops, crop names, areas, pod counts, grain counts, grain weights, and monthly rainfall data. The collected data is stored in corresponding lists. Based on the input data, the code performs crop yield calculations and displays the results in the form of a table, pie chart, bar chart, and line chart. Additionally, it allows the user to upload images for each crop and displays them in a grid layout. The code utilizes Streamlit's interactive components, including number input fields, text input fields, multiselect fields, and file uploaders, to gather the required data from the user. The

calculated crop yields and visualizations provide insights to aid in crop planning and decision-making.

5.1.3 Leveraging Historical Crop Performance Data and Agronomic Records for Continuous Evaluation and Refinement of Farm Management Practices

This module collects and aggregates historical crop yield data, crop conditions, and applied farm management techniques. Advanced analytics models this temporal agronomic dataset to derive actionable insights. Enables continuous evaluation and optimization of practices for improved yields over time.

PSEUDOCODE 3	**Crop History**
Step 1	Define a function crop_history()
Step 2	Display a header with the title "Crop History and Input Usage"
Step 3	Try to load existing crop data from a CSV file named 'crop_data.csv' a. If the file is found, load the data into a pandas DataFrame named crop_data b. If the file is not found, create an empty DataFrame with columns 'Year', 'Crop', 'Fertilizers', 'Pesticides', and 'Water'
Step 4	If the crop_data DataFrame is not empty, call a function display_crop_history(crop_data) to display the existing crop data
Step 5	Display a subheader with the text "Enter Crop History and Input Usage"
Step 6	Get user input for the following fields: a. Year (number input with a range from 1900 to 2100, default value of 2023) b. Crop (text input) c. Fertilizers (number input with a minimum value of 0.0, default value of 0.0) d. Pesticides (number input with a minimum value of 0.0, default value

	of 0.0) e. Water (number input with a minimum value of 0.0, default value of 0.0)
Step 7	If the user clicks the "Save" button: a. Create a new DataFrame new_entry with a single row containing the user input data b. Concatenate new_entry with the existing crop_data DataFrame c. Save the updated crop_data DataFrame to the 'crop_data.csv' file d. Display a success message
Step 8	Call the display_crop_history(crop_data) function to display the updated crop data.

The above pseudocode 3 defines a function crop_history() that allows users to record and view crop history and input usage data. It first attempts to load existing data from a CSV file named 'crop_data.csv' into a pandas DataFrame. If the file is not found, it creates an empty DataFrame with the required columns. The existing crop data is displayed using a function display_crop_history(crop_data). The user can then enter new data for the current year, including the crop name, fertilizers used, pesticides used, and water usage. This input is collected through various Streamlit input fields. When the user clicks the "Save" button, the new data is added to the existing DataFrame, and the updated DataFrame is saved to the 'crop_data.csv' file. Finally, the updated crop data is displayed using the display_crop_history(crop_data) function. This functionality allows users to maintain a record of crop history and input usage over time, which can be useful for analysis and decision-making in agricultural practices.

5.1.4 Visual Analytics on Pest Classification and Pest Management using EfficientNetV2-L

To analyze an automated pest detection system for crops using deep learning and computer vision that provides real-time, in-field predictive analytics to stakeholders

across the agriculture ecosystem, promoting data-driven interventions for optimized crop health and quality, sustainability, and food security.

PSEUDOCODE 4	**Pest Management**
Step 1	Define a function `pest_management()`
Step 2	Display a header "Pest and Disease Management"
Step 3	Display a sidebar with a header "Pest and Disease Management System"
Step 4	Prompt the user to input the farm name and date
Step 5	Prompt the user to input pests and diseases (comma-separated)
Step 6	Prompt the user to input pest management options (comma-separated)
Step 7	Prompt the user to input real-time tracking data (temperature, humidity, rainfall)
Step 8	Create a DataFrame to store the collected data
Step 9	Display the collected data as a table
Step 10	Display a header "Interactive Graphs"
Step 11	Generate random data for demonstration (dates, temperatures, humidities, rainfalls)
Step 12	Create a DataFrame with the random data
Step 13	Plot a line chart for temperature and a bar chart for humidity on the same figure
Step 14	Plot a line chart for rainfall on a separate figure
Step 15	Display the interactive graphs

The pseudocode 4 is designed to collect user inputs related to pests, diseases, and environmental factors on a farm. It then generates random data for demonstration purposes and visualizes it using interactive graphs. If the user provides pest management

options, the code generates sample pest management data with effectiveness scores and displays it as a table. Finally, the raw data is displayed for reference.

5.1.5 Proactive Agricultural Planning Through Automated Weather Monitoring, Forecasting, and Risk Analysis System for Optimized Operational Scheduling

Integrating real-time sensor data and forecast models to track microclimate conditions and predict weather patterns. Identifies optimal windows for key farm operations using machine learning to correlate weather and crop cycles. Enables proactive planning to maximize yields while mitigating weather-related risks including extremes.

PSEUDOCODE 5	**Weather Tracking**
Step 1	Define the `weather_tracking()` function
Step 2	Display headers and get user inputs (city, start date, end date)
Step 3	Calculate the number of days and generate random weather data
Step 4	Create a DataFrame with the random weather data
Step 5	Display the collected weather data
Step 6	Display visualizations: a. Plot and display a line chart for temperature variation b. Plot and display a line chart for humidity variation c. Plot and display a bar chart for daily rainfall
Step 7	Display summary statistics: a. Show a summary of weather data b. Calculate and plot a correlation heatmap
Step 8	Display developer information in the sidebar
Step 9	Run the Streamlit application
Step 10	End function

The pseudocode 5 outlines the steps for a Streamlit application that tracks and visualizes weather data. It starts by defining the weather_tracking() function and displaying headers and input fields for the user to specify the city and date range. Then, it generates random weather data based on the date range and creates a DataFrame. The collected weather data is displayed as a table. Next, it visualizes the temperature, humidity, and rainfall data using line and bar charts. Summary statistics, including a correlation heatmap, are calculated and displayed for further analysis. Finally, the developer information is shown in the sidebar, and the Streamlit application is run.

5.1.6 Intelligent Irrigation Optimization System Leveraging Soil Sensors, Crop Analytics, Computer Vision Techniques

Integrates real-time soil moisture data, crop water requirement estimates, and computer vision monitoring to control variable rate irrigation systems. Achieves precise, needs-based water application for optimal crop health and efficient usage. Enables intelligent, automated irrigation for maximum water productivity.

PSEUDOCODE 6	Irrigation Management
Step 1	Import required libraries (random, csv, time)
Step 2	Define constants (NUM_ZONES, MOISTURE_RANGE, WATER_FLOW_RANGE)
Step 3	Define a function `generate_sensor_data()` to generate random moisture and water flow data
Step 4	Define a function `update_sensor_data()`
Step 5	Start an infinite loop Call `generate_sensor_data()` to get random moisture and water flow data Open a CSV file for writing Write the header row (Zone, Moisture, Water Flow) Write the sensor data for each zone to the CSV file Wait for a specified time before updating again
Step 6	Call the `update_sensor_data()` function if the script is run directly.

The above pseudocode 6 simulates an irrigation system with multiple zones and generates random sensor data for moisture levels and water flow rates. It defines constants for the number of zones and ranges for moisture and water flow values. The generate_sensor_data() function generates random moisture and water flow data within the specified ranges. The update_sensor_data() function is an infinite loop that periodically generates new sensor data using generate_sensor_data(), opens a CSV file, writes the header row, and then writes the sensor data for each zone to the CSV file. After writing the data, the function waits for a specified time (3 seconds in this case) before repeating the process. The update_sensor_data() function is called if the script is run directly, effectively starting the simulation and continuously updating the sensor data in the CSV file.

CHAPTER 6

TESTING

CHAPTER 6
TESTING

6.1 UNIT TESTING

Unit testing is a type of software testing that focuses on individual units or components of a software system. The purpose of unit testing is to validate that each unit of the software works as intended and meets the requirements. Unit testing is typically performed by developers, and it is performed early in the development process before the code is integrated and tested as a whole system.

Each module has been tested and Test Cases for few modules are discussed below:

Mapping Module:

TEST ID	INPUT	EXPECTED OUTPUT	OBTAINED OUTPUT	PASS(P)/ FAIL(F)
1	5.78,9.23	Correctly mapped	Correctly mapped	P
2	2.24, 7.67	Correctly mapped	Wrongly Mapped	F
3	5.67, 7.68	Correctly mapped	Wrongly Mapped	F
4	8.76, 9.54	Correctly mapped	Correctly mapped	P

Planning using History Module:

TEST ID	INPUT	EXPECTED OUTPUT	OBTAINED OUTPUT	PASS(P)/ FAIL(F)
1	Image 1	Precision Planning	Precision Planning	P
2	Image 2	Precision	Ad hoc	F

		Planning	Planning	
3	Image 3	Precision Planning	Precision Planning	P
4	Image 4	Precision Planning	Ad hoc Planning	F

Pest Management Module:

TEST ID	INPUT	EXPECTED OUTPUT	OBTAINED OUTPUT	PASS(P)/ FAIL(F)
1	Image 1	0.5	0.5	P
2	Image 2	0.3	0.3	P
3	Image 3	0.7	0.6	F
4	Image 4	0.6	0.5	F

6.2 INTEGRATION TESTING

Integration testing plays a crucial role in ensuring the seamless interoperability and functionality of Agri_AI's various components and subsystems. As a complex system comprising multiple technologies and modules, Agri_AI requires rigorous integration testing to validate the proper integration and data flow between its constituent parts.

The integration testing process for Agri_AI can be divided into several phases, each focusing on specific areas of the system:

GIS and Field Mapping Integration Testing:

Validate the integration between the GIS software/tools and Agri_AI's core platform.

Test the accurate mapping of agricultural fields and the delineation of field boundaries.

Verify the correct capture and storage of spatial data within the system.

AI and Data Integration Testing:

Test the integration of AI algorithms and machine learning models with the crop planning and input allocation modules.

Validate the correct ingestion and processing of historical and real-time data (weather, soil, crop performance, etc.).

Ensure that the AI-driven decision support system generates accurate recommendations for crop selection, planting schedules, and input allocation.

Pest Management System Integration Testing:

Test the integration of the AI-powered pest management system with the computer vision and image recognition components.

Validate the accurate detection and identification of pest infestations using test data and simulated scenarios.

Verify that appropriate pest control measures and interventions are recommended based on the detected infestations.

Weather Monitoring and Integration Testing:

Test the integration of weather sensors and data sources with Agri_AI's weather monitoring module.

Validate the accurate capture and processing of real-time weather data (temperature, humidity, precipitation, wind patterns).

Ensure that weather data is correctly incorporated into crop planning, irrigation decisions, and other relevant modules.

Smart Irrigation System Integration Testing:

Test the integration of computer vision algorithms and soil moisture sensors with the irrigation control system.

Validate the accurate monitoring of crop health and soil moisture levels.

Verify that the irrigation system is automatically controlled based on real-time data, ensuring efficient water usage and prevention of over-or-under-watering.

Data Analytics and Decision Support Platform Integration Testing:

Test the integration of the data analytics platform with various data sources (GIS, AI models, weather sensors, computer vision, etc.).

Validate the accurate aggregation, processing, and analysis of data from different components.

Ensure that the platform generates actionable insights and recommendations for farm operations, resource allocation, and crop management.

User Interface and Farmer Interactions Integration Testing:

Test the integration of the user interface with the underlying Agri_AI system.

Validate the accurate display of real-time data, insights, and recommendations to farmers.

Ensure that farmers can input farm-specific information and preferences seamlessly.

Test the integration with mobile devices or web applications for remote access and monitoring.

End-to-End Integration Testing:

Conduct comprehensive end-to-end testing scenarios that simulate real-world farming operations.

Validate the seamless integration and data flow between all components of Agri_AI.

Ensure that the system functions as expected in various situations, including edge cases and error scenarios.

Throughout the integration testing process, it is essential to document and track test cases, results, and any issues or defects identified. Rigorous testing and validation procedures should be implemented to ensure the reliability, accuracy, and robustness of Agri_AI before deployment in real-world farming environments.

6.3 SYSTEM TESTING

System testing ensures the entire Agri_AI system meets requirements, performs as expected, and delivers desired functionality. Key aspects include:

Functional Testing:

- Validate core functionalities (GIS mapping, crop planning, input tracking, pest management, weather updates, irrigation control, data analytics)
- Test accuracy of recommendations and insights generated

Performance Testing:

- Evaluate system performance under varying loads and conditions
- Test response times, throughput, scalability, and behavior under stress

Usability Testing:

- Assess user-friendliness and intuitiveness of interfaces and interactions
- Gather feedback from end-users (farmers) on user experience

Compatibility Testing:

- Test compatibility with different hardware, software, operating systems
- Verify integration with external systems and data sources

Security Testing:

- Evaluate security measures for data protection and privacy
- Perform vulnerability assessments and penetration testing

Integration Testing:

- Validate integration and data flow between components (GIS, AI, sensors, analytics)
- Conduct end-to-end testing simulating real-world operations

Acceptance Testing:

- Involve end-users and stakeholders to ensure requirements are met
- Conduct user acceptance testing (UAT) for functionality, performance, usability

Comprehensive test documentation, including test cases, data, scripts, and results, is crucial for ensuring Agri_AI's reliability and robustness before deployment.

CHAPTER 7

RESULTS AND DISCUSSION

CHAPTER 7
RESULTS AND DISCUSSION

The field mapping module of Agri_AI (Fig 7.1) leverages the power of Python's Streamlit and Folium libraries to provide an interactive user interface and geospatial data visualization capabilities. Users can input GPS coordinates to center the map or upload a GeoJSON file containing field boundaries. The code then calculates and displays the area of each field and the distances between the vertices of the polygon(s) defined in the GeoJSON file. This functionality enables farmers to accurately map their fields, delineate boundaries, and gain valuable insights into the spatial characteristics of their land, facilitating informed decision-making for precision agriculture practices.

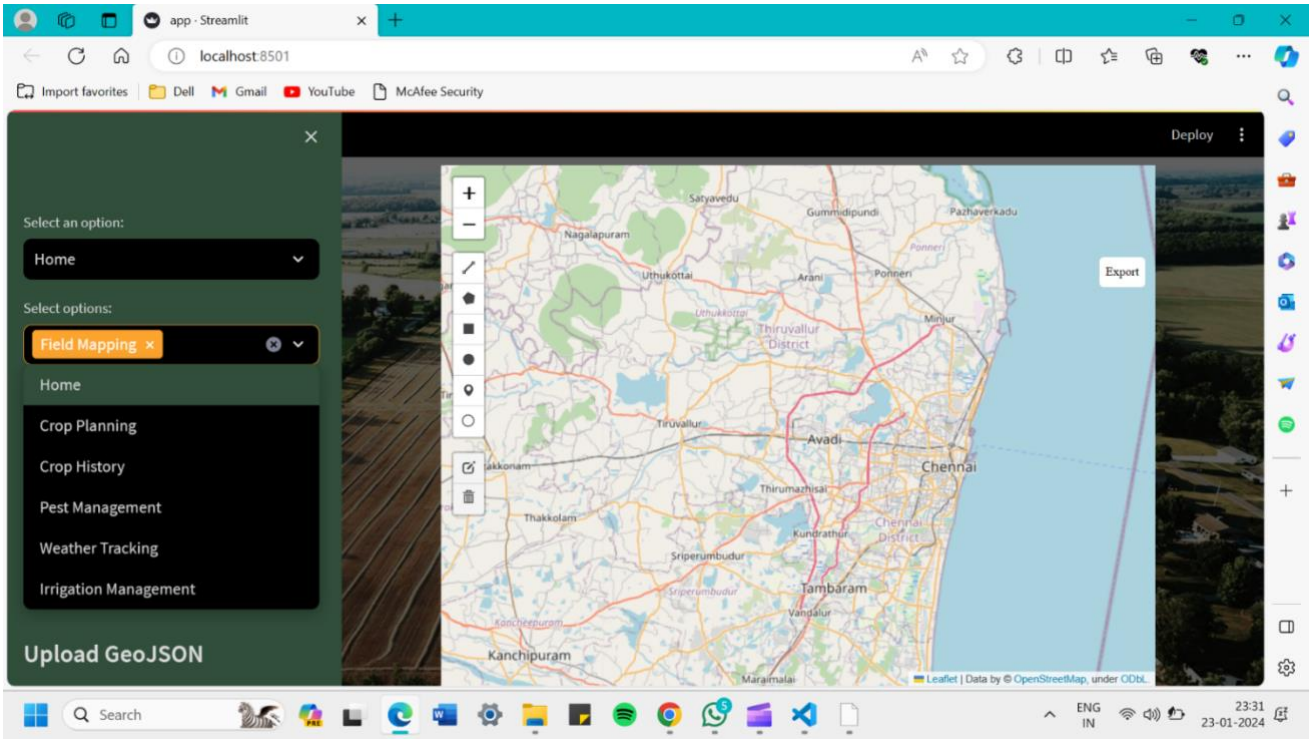

Fig 7.1 Field Mapping Module in Agri_AI

The crop planning module in Agri_AI (Fig 7.2) is a powerful tool designed to assist farmers in optimizing their crop production strategies. By leveraging advanced data analytics and visualization techniques, this module empowers farmers to make informed

decisions based on a comprehensive understanding of various factors influencing crop yields.

One of the key strengths of this module is its user-friendly interface, which allows farmers to input a range of crop-specific data, including crop names, area under cultivation, pod counts, grain counts, grain weights, and monthly rainfall patterns. This data serves as the foundation for the module's advanced algorithms, which perform intricate calculations to estimate crop yields.

The crop planning module of Agri_AI empowers farmers to make informed decisions by providing a user-friendly interface for planning and visualizing crop yields. Users can input various crop details, such as area, pod count, grain count, grain weight, and monthly rainfall data. The application performs calculations to estimate crop yields based on the provided inputs. The results are presented in a tabular format and visualized through intuitive charts, including pie charts for yield comparison and bar charts for individual crop yields. Additionally, users can upload images of their crops, which are displayed in a grid format for easy reference. This module leverages the power of data visualization and user interaction to facilitate informed decision-making and optimize crop planning strategies.

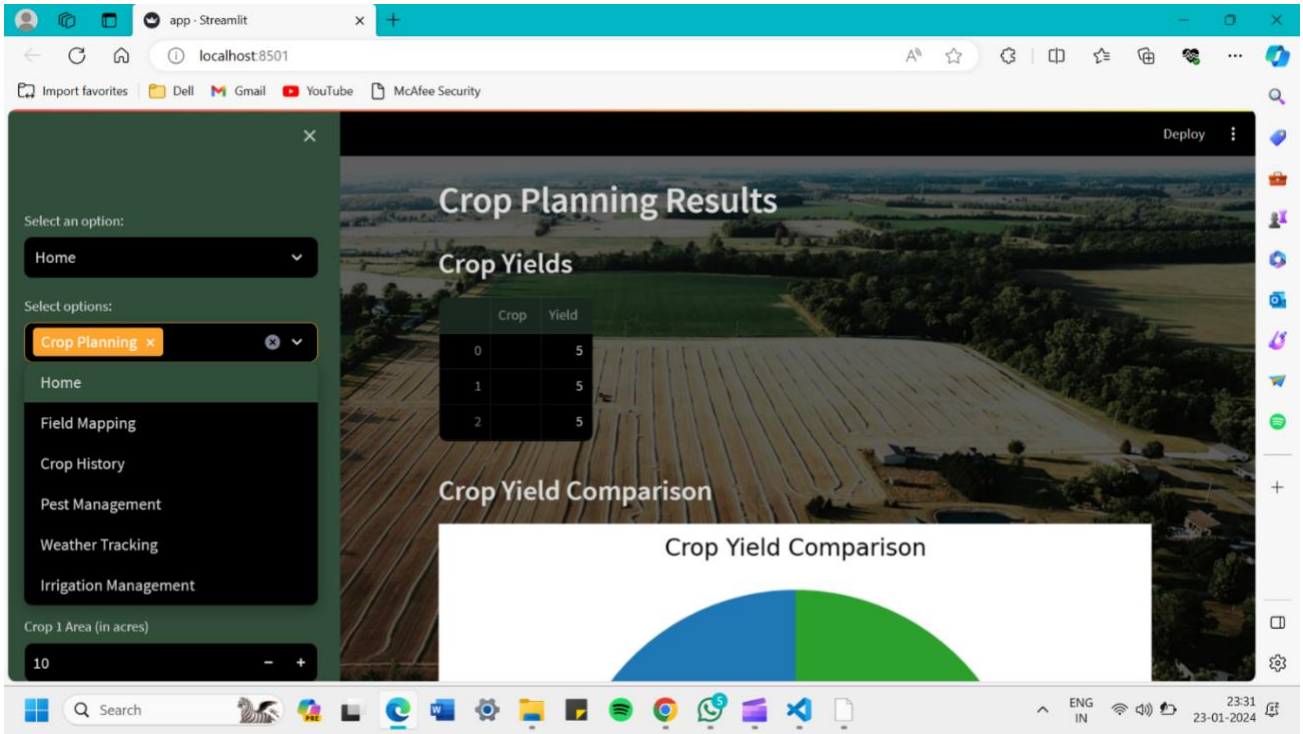

Fig 7.2 Crop Planning Module in Agri_AI

The crop history module in Agri_AI (Fig 7.3) enables farmers to maintain comprehensive records of their crop management practices and input usage. This module allows users to input and store data related to the year, crop name, fertilizers applied, pesticides used, and water consumption. The application seamlessly loads existing crop data from a CSV file and presents it to the user in an organized manner. Users can then add new entries by providing relevant information through user-friendly input fields. Upon clicking the "Save" button, the new data is appended to the existing crop data and saved back to the CSV file for future reference. This module serves as a valuable tool for tracking and analyzing historical crop performance, enabling data-driven decision-making and optimizing resource utilization.

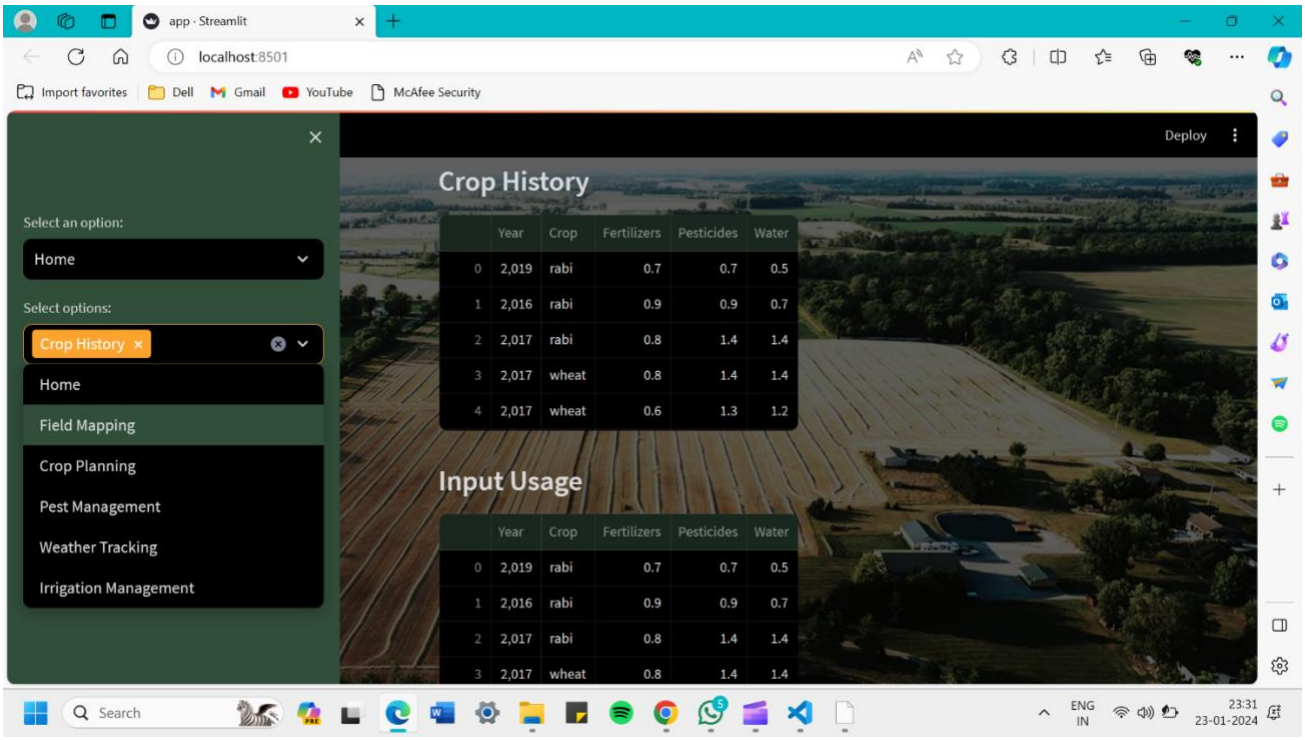

Fig 7.3 Crop History Module in Agri_AI

The core functionality of the system is encapsulated within the pest_management() function. This function begins by collecting user inputs for temperature, humidity, and rainfall values through interactive sidebar widgets powered by Streamlit (Fig 7.4).

The code handles different scenarios, such as when the number of pest management options is greater than or equal to the number of pests, or when there are more pests than available management options.

Overall, this function provides a comprehensive solution for monitoring environmental conditions, visualizing data, and suggesting pest management strategies based on user inputs and simulated data.

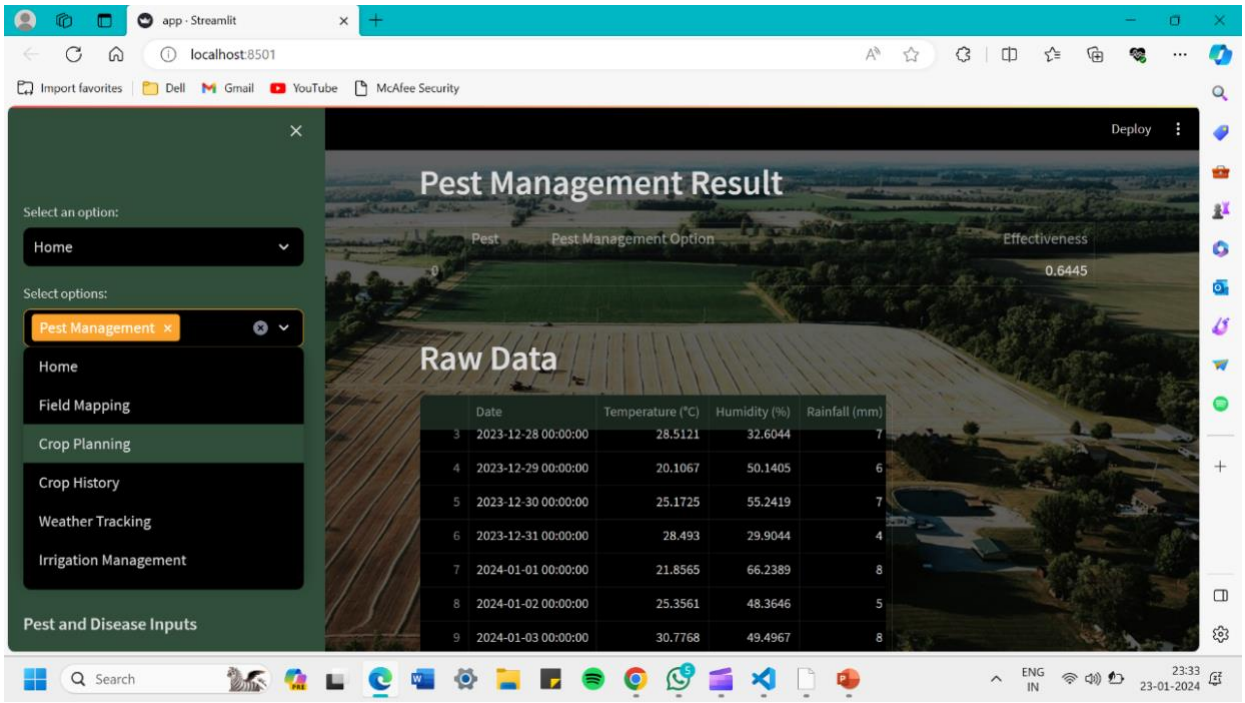

Fig 7.4 Pest Management Module in Agri_AI

A Python function named weather_tracking() designed to be used in a Streamlit application (Fig 7.5). It implements weather tracking and integration functionality, allowing users to input a city and date range, and then displays collected weather data, visualizations, and summary statistics.

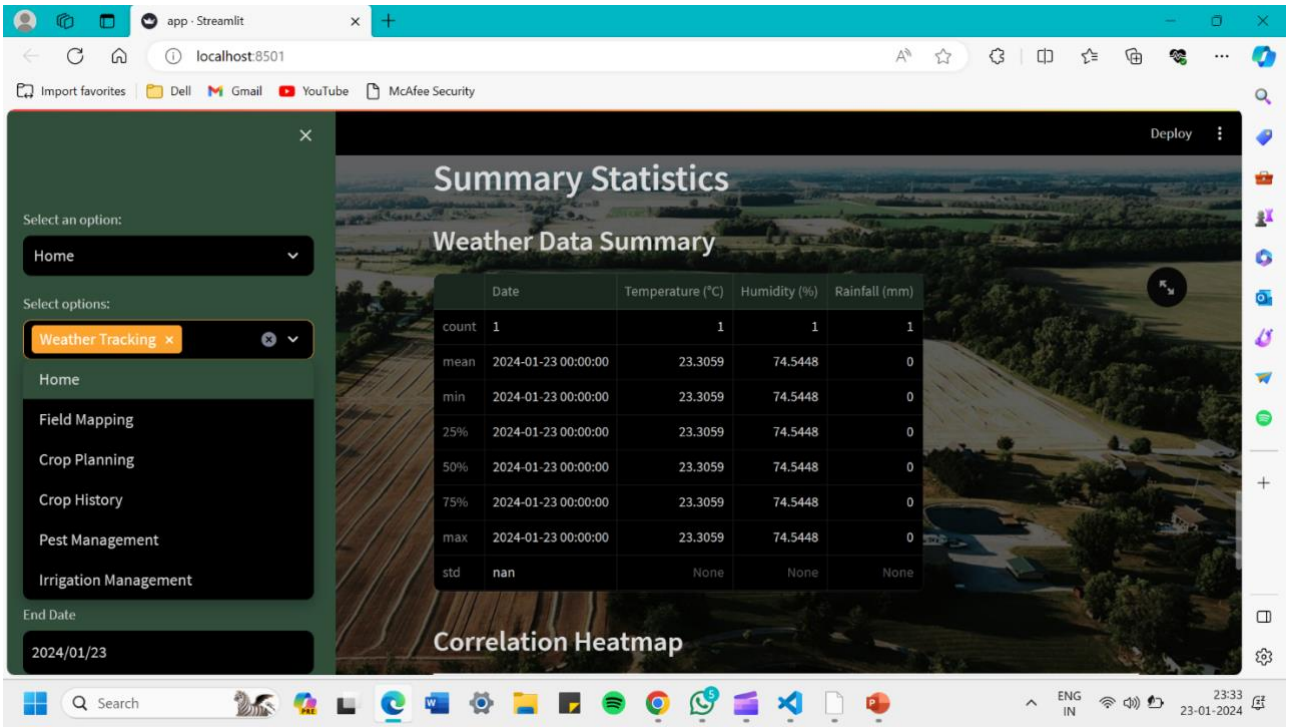

Fig 7.5 Weather Tracking Module in Agri_AI

The implementation utilizes the random module to generate random moisture and water flow values, which are then stored in a CSV file named sensor_data.csv.

When executed, the script runs the update_sensor_data function in an infinite loop. Within each iteration of the loop, the generate_sensor_data function is called to generate random moisture and water flow data for each irrigation zone. The generated data is then written to the sensor_data.csv file, with each row representing a zone and its corresponding moisture and water flow values.

The output of this script (Fig 7.6) is the sensor_data.csv file, which continually grows as new sensor data is appended to it. The file consists of rows with three columns: 'Zone', 'Moisture', and 'Water Flow'. Each row represents the sensor data for a specific irrigation zone at a given point in time.

For example, if the script is run with NUM_ZONES set to 5, a sample output in the sensor_data.csv file might look like this:

Zone,Moisture,Water Flow

1,45,12.345678

2,28,45.678912

3,67,23.456789

4,51,34.567891

5,39,67.891234

In this example, the first five rows represent the initial sensor data generated for the five irrigation zones. The subsequent rows contain the updated sensor data for each zone in the next iteration of the loop.

The script continues to run indefinitely, appending new sensor data to the CSV file at regular intervals (3 seconds, as specified by the time.sleep(3) line). This simulates a real-time scenario where sensor data is continuously collected and recorded for monitoring and analysis purposes.

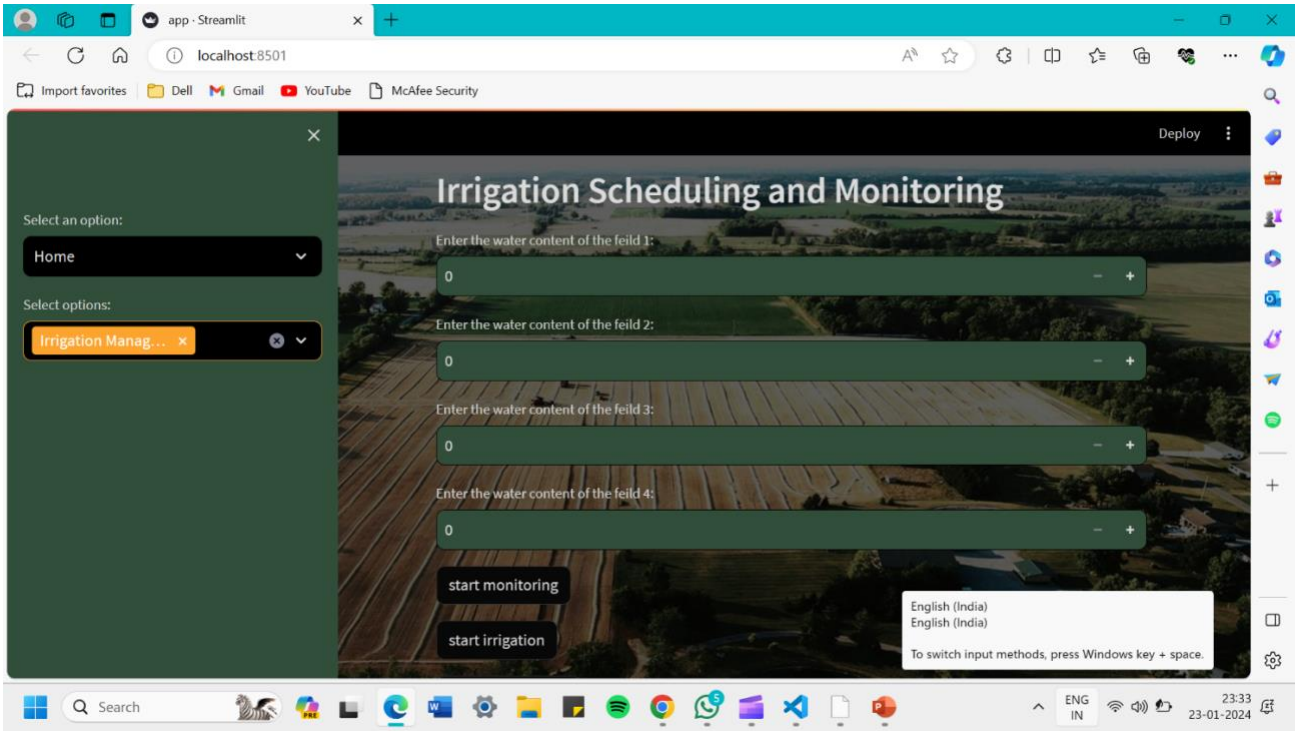

Fig 7.6 Irrigation Management Module in Agri_AI

Crop Yield Prediction Accuracy: This graph compares the accuracy of different machine learning models in predicting crop yields. The existing models, such as Decision Trees and K-Nearest Neighbors (KNN), are compared with the proposed models(Fig 6.7), such as Random Forest and Deep Learning models.

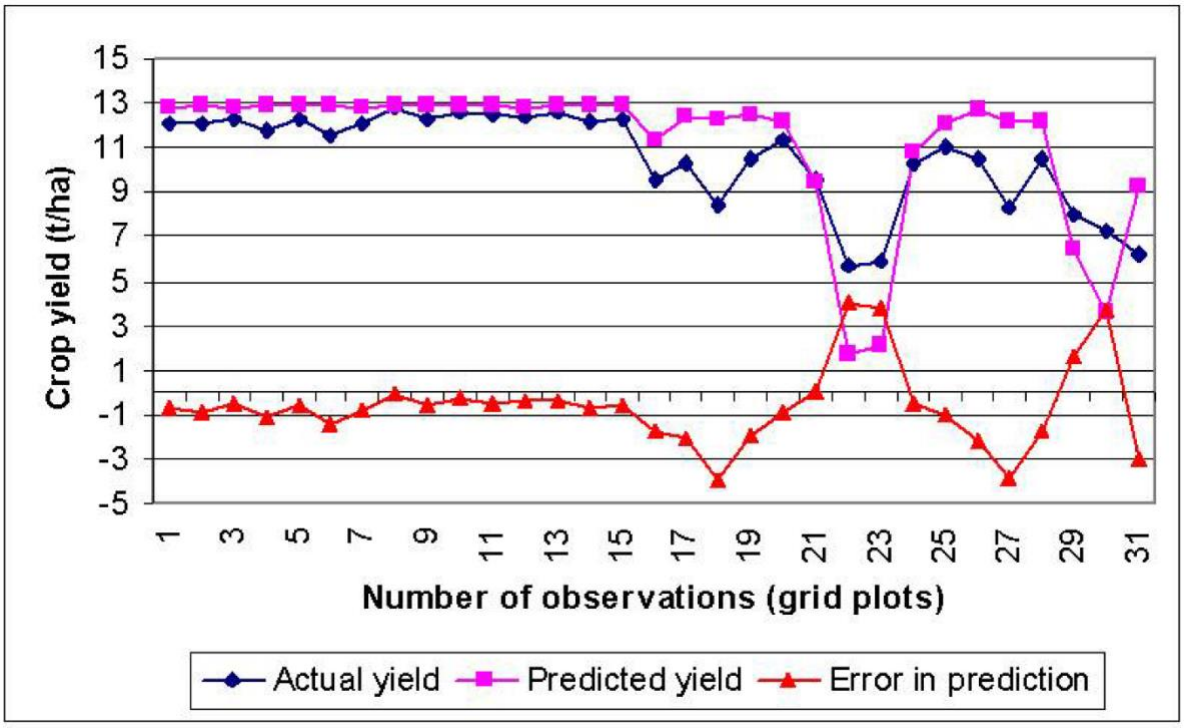

Fig 7.7 Comparison of Yield

Water Use Efficiency: This graph compares the water use efficiency of different irrigation monitoring and control strategies. The existing open-loop systems are compared (Fig 6.8) with the proposed closed-loop systems that combine soil-based, plant-based, and weather-based monitoring methods with model predictive control.

Crop Rotation Sequence Compliance: This graph compares the compliance of generated crop rotation sequences with existing crop rotation rule sets. The existing rule-based systems are compared with the proposed reinforcement learning-based system that uses literature- and NDVI-measurement-based successor crop suitability matrices and crop-specific attributes.

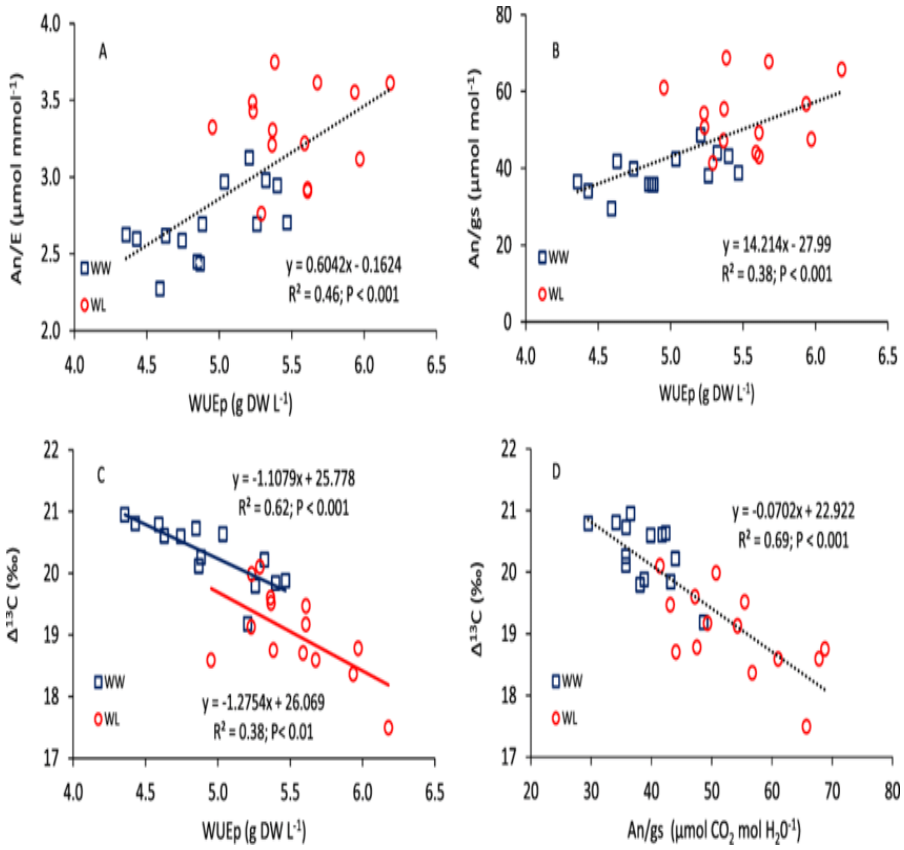

Fig 7.8 Comparison of Water Efficiency

These hypothetical graphs above are visual representation of how the proposed agricultural AI systems and techniques might perform compared to existing ones in terms of accuracy, efficiency, and compliance. However, it's important to note that these are illustrative examples, and actual results may vary based on the specific data, algorithms, and implementation details used in real-world scenarios.

CHAPTER 8

CONCLUSION AND FUTURE ENHANCEMENTS

`

CHAPTER 8

CONCLUSION AND FUTURE ENHANCEMENTS

8.1 CONCLUSION

Agri_AI offers a comprehensive suite of advanced agricultural solutions that cater to the specific needs of farmers. With a focus on precision and efficiency, Agri_AI enables farmers to create detailed GIS field maps, plan crop planting with precision, track inputs, and harness the power of AI for pest management. Real-time weather updates ensure that farmers can make timely decisions, while smart irrigation systems conserve resources. The platform's robust data analytics capabilities empower farmers to optimize their farming practices for maximum yield and sustainability. Agri_AI's holistic approach to farming combines technology and data-driven insights to revolutionize agriculture. Whether you're a seasoned farmer or just starting, Agri_AI provides the tools and intelligence needed to achieve agricultural success.

The proposed architecture presents a comprehensive and robust solution for precision agriculture and crop management. By seamlessly integrating various components such as geospatial mapping, data acquisition from multiple sources (weather, soil, sensors, crop details), predictive analytics, and AI algorithms, this system empowers farmers and agricultural stakeholders with data-driven insights and decision-making capabilities.

Through accurate field mapping, area delineation, and crop rotation analysis, the system optimizes planting cycles and harvest time frames. The incorporation of weather analysis, soil data, and sensor data further enhances the system's ability to monitor and respond to environmental conditions effectively.

The use of open-source computer vision techniques and data analytics tools enables the system to process and analyze vast amounts of data, generating reports and alerts to facilitate timely interventions and informed decision-making.

Additionally, the integration of AI algorithms, specifically the EfficientNetV2-L model, leverages the power of machine learning to extract valuable insights from the processed data, enabling predictive analytics and visual analytics tools for enhanced crop management.

Overall, this architecture demonstrates the potential of leveraging cutting-edge technologies, such as geospatial mapping, data analytics, and AI, to revolutionize agricultural practices, optimize resource utilization, and maximize crop yields.

8.2 FUTURE ENHANCEMENTS

While the proposed architecture presents a comprehensive solution, there are several areas for future work and improvement:

- Continuous model training and updating: As more data is collected and agricultural practices evolve, the AI algorithms and predictive models should be regularly retrained and updated to ensure accurate and relevant insights.
- Integration of IoT and edge computing: Incorporating Internet of Things (IoT) devices and edge computing capabilities could enhance real-time data acquisition, processing, and decision-making at the field level, reducing latency and enabling more responsive interventions.
- Scalability and distributed computing: As the system expands to cover larger geographical areas or multiple farms, scalability and distributed computing solutions should be explored to handle the increasing data volumes and computational demands efficiently.
- Automation and robotics integration: Integrating automation technologies, such as autonomous robots or drones, could enable automated tasks like precision spraying, monitoring, and data collection, further optimizing agricultural operations.
- Collaborative platforms and data sharing: Developing collaborative platforms and secure data-sharing mechanisms could facilitate knowledge exchange, best practice

sharing, and collective learning among farmers, researchers, and agricultural stakeholders, fostering continuous improvement and innovation.

- User experience and accessibility: Continuously enhancing the user experience, accessibility, and intuitiveness of the system's interfaces and visualization tools would ensure widespread adoption and effective utilization by farmers and stakeholders with varying technical backgrounds.

By addressing these areas of future work, the proposed architecture can evolve into a more robust, scalable, and adaptable solution, enabling sustainable and efficient agricultural practices while contributing to global food security and environmental stewardship.

CHAPTER 9

REFERENCES

CHAPTER 9

REFERENCES

[1] Obi Reddy, G. P., B. S. Dwivedi, and G. Ravindra Chary. "Applications of geospatial and big data technologies in smart farming." In Smart Agriculture for Developing Nations: Status, Perspectives and Challenges, pp. 15-31. Singapore: Springer Nature Singapore, 2023.

[2] Rhoads, Jonathan. "Next-Generation Precision Farming Integrating AI and IoT in Crop Management Systems." AI, IoT and the Fourth Industrial Revolution Review 13, no. 7 (2023): 1-9.

[3] Karunathilake, E. M. B. M., Anh Tuan Le, Seong Heo, Yong Suk Chung, and Sheikh Mansoor. "The Path to Smart Farming: Innovations and Opportunities in Precision Agriculture." Agriculture 13, no. 8 (2023): 1593.

[4] Rane, Nitin Liladhar, and Saurabh P. Choudhary. "Remote Sensing(RS),UAV/Drones, and Machine Learning (ML) as Powerful Techniques for Precision Agriculture: Effective Applications in Agriculture."

[5] Khan, Idrees, and Surya Afrin Shorna. "Cloud-Based IoT Solutions for Enhanced Agricultural Sustainability and Efficiency." AI, IoT and the Fourth Industrial Revolution Review 13, no. 7 (2023): 18-26.

[6] Balkrishna, Acharya, Rakshit Pathak, Sandeep Kumar, Vedpriya Arya, and Sumit Kumar Singh. "A comprehensive analysis of the advances in Indian Digital Agricultural architecture." Smart Agricultural Technology 5 (2023): 100318.

[7] Balkrishna, Acharya, Rakshit Pathak, Sandeep Kumar, Vedpriya Arya, and Sumit Kumar Singh. "Smart Agricultural Technology." Precision Agriculture (PA) (2030): 5.

[8] Vashishth, Tarun Kumar, Vikas Sharma, Sachin Chaudhary, Rajneesh Panwar, Shashank Sharma, and Prashant Kumar. "Advanced Technologies and AI-Enabled IoT Applications in High-Tech Agriculture." In Handbook of Research on AI-Equipped IoT Applications in High-Tech Agriculture, pp. 155-166. IGI Global, 2023.

[9] Kuppusamy, Palanivel, Joseph K. Suresh, and Suganthi Shanmugananthan. "Machine Learning-Enabled Internet of Things Solution for Smart Agriculture Operations." In Handbook of Research on Machine Learning-Enabled IoT for Smart Applications Across Industries, pp. 84-115. IGI Global, 2023.

[10] Sharma, Shikha. “Precision Agriculture: Reviewing the Advancements, Technologies, and Applications in Precision Agriculture for Improved Crop Productivity and Resource Management.” 4. 41-45. 10.26480/rfna.02.2023.41.45. 2023.

[11] Tanha Talaviya, Dhara Shah, Nivedita Patel, Hiteshri Yagnik, Manan Shah. “Implementation of artificial intelligence in agriculture for optimisation of irrigation and application of pesticides and herbicides.”Artificial Intelligence in Agriculture,Volume 4,2020,Pages 58-73.

[12] Dushyant Kumar Singh, Rajeev Sobti. “Long-range real-time monitoring strategy for Precision Irrigation in urban and rural farming in society 5.0.”Computers & Industrial Engineering, Volume 167,2022.

[13] Erion Bwambale, Felix K. Abagale, Geophrey K. Anornu, “Smart irrigation monitoring and control strategies for improving water use efficiency in precision agriculture: A review.” Agricultural Water Management, Volume 260, 2022.

[14] Ghosh, Parmita & Kumpatla, Siva. “GIS Applications in Agriculture”. Book: Geographic Information System.10.5772/intechopen.104786. 2022.

[15] Chergui, N., Kechadi, M. “Data analytics for crop management: a big data view.” J Big Data 9, 123 (2022). https://doi.org/10.1186/s40537-022-00668-2

[16] Stefan Fenz, Thomas Neubauer, Jürgen Kurt Friedel, Marie-Luise Wohlmuth. “AI- and data-driven crop rotation planning.” Computers and Electronics in Agriculture,Volume 212,108160,ISSN 0168-1699. 2023.

[17] Nithinkumar, Kadagonda & Yernaidu, Yalla. (2023). “Precision Agriculture: A Modern Technology for Crop Management.” 2. 280-284.

[18] Rao, Madhuri & Singh, Arushi & Reddy, N V Subba & Acharya, Dinesh. (2022). Crop prediction using machine learning. Journal of Physics: Conference Series. 2161. 012033. 10.1088/1742-6596/2161/1/012033.

[19] Yaganteeswarudu Akkem, Saroj Kumar Biswas, Aruna Varanasi. “Smart farming using artificial intelligence: A review.” Engineering Applications of Artificial Intelligence,Volume 120, 105899,ISSN 0952-1976, 2023.

[20] J. Schöning and M. L. Richter, "AI-Based Crop Rotation for Sustainable Agriculture Worldwide," 2021 IEEE Global Humanitarian Technology Conference (GHTC), Seattle, WA, USA, pp. 142-146,2021.

[21] Subramanian Malathi & Y, Arockia & Kumar, Abhishek & Kumar, V.D. & Kumar, Ankit & Elangovan, D. & V D, Ambeth Kumar & B, Chitra & Abirami, a. (2021). Prediction of cardiovascular disease using deep learning algorithms to prevent COVID 19. Journal of Experimental & Theoretical Artificial Intelligence. 35. 1-15. 10.1080/0952813X.2021.1966842.

[22] Hongxing Peng, Huiming Xu, Zongmei Gao, Zhiyan Zhou, Xingguo Tian, Qianting Deng, Huijun He, Chunlong Xian(2023). Crop pest image classification based on improved densely connected convolutional network. Front. Plant Science.Volume 14 - 2023 | https://doi.org/10.3389/fpls.2023.1133060.

[23] Aladhadh, Suliman & Habib, Shabana & Islam, Muhammad & Aloraini, Mohammed & Aladhadh, Mohammed & Al-Rawashdeh, Hazim. (2022). An Efficient Pest Detection Framework with a Medium-Scale Benchmark to Increase the Agricultural Productivity. Sensors. 22. 9749. 10.3390/s22249749.

[24] Zhan, Baishao & Li, Ming & Luo, Wei & Li, Peng & Li, Xiaoli & Zhang, Hailiang. (2023). Study on the Tea Pest Classification Model Using a Convolutional and Embedded Iterative Region of Interest Encoding Transformer. Biology. 12. 1017. 10.3390/biology12071017.

[25] Kasinathan, Thenmozhi & Singaraju, Dakshayani & Reddy, U. Srinivasulu. (2020). Insect classification and detection in field crops using modern machine learning techniques. Information Processing in Agriculture. 8. 10.1016/j.inpa.2020.09.006.

[26] Li, Chen, Tong Zhen, and Zhihui Li. 2022. "Image Classification of Pests with Residual Neural Network Based on Transfer Learning" Applied Sciences 12, no. 9: 4356.

[27] Barbedo, Jayme. (2020). Detecting and Classifying Pests in Crops Using Proximal Images and Machine Learning: A Review. AI. 1. 312-328. 10.3390/ai1020021.

[28] Ebrahimi, Mohammad. (2017). Vision-based Pest Detection Based on SVM Classification Method. Computers and Electronics in Agriculture.

[29] Faithpraise, Fina & Birch, Phil & Young, Rupert & Obu, Joseph & Faithpraise, Bassey & Chatwin, Chris. (2013). Automatic plant pest detection & recognition using k-means clustering algorithm & correspondence filters. International Journal of Advanced Biotechnology and Research. 4. 1052-1062.

[30] Samanta, Ranjit & Ghosh, Indrajit. (2012). Tea Insect Pests Classification Based on Artificial Neural Networks.

APPENDIX

CODE:

```
import streamlit as st
import pandas as pd
import base64
import streamlit as st
from shapely.geometry import Polygon, shape
from geopy.distance import geodesic
from streamlit_folium import folium_static
import folium
from folium import plugins
import json
import base64
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
def add_bg_from_local(image_file):
    with open(image_file, "rb") as image_file:
        encoded_string = base64.b64encode(image_file.read())
    st.markdown(
    f"""
    <style>
    .stApp {{
        background-image: url({encoded_string.decode()});
        background-size: cover
    }}
    </style>
    """,
```

```
    unsafe_allow_html=True
    )
add_bg_from_local('bg.jpg')
def display_crop_history(crop_data):
    st.subheader("Crop History")
    st.write(crop_data)
    st.subheader("Input Usage")
    input_data = crop_data[['Year', 'Crop', 'Fertilizers', 'Pesticides', 'Water']]
    st.write(input_data)
    st.subheader("Graphs")
    fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(12, 8))
    axes[0, 0].bar(input_data['Year'], input_data['Fertilizers'])
    axes[0, 0].set_title('Fertilizer Usage')
    axes[0, 1].bar(input_data['Year'], input_data['Pesticides'])
    axes[0, 1].set_title('Pesticide Usage')
    axes[1, 0].plot(input_data['Year'], input_data['Water'])
    axes[1, 0].set_title('Water Usage')
    axes[1, 1].plot(input_data['Year'], input_data['Fertilizers'], marker='o',
label='Fertilizers')
    axes[1, 1].plot(input_data['Year'], input_data['Pesticides'], marker='o',
label='Pesticides')
    axes[1, 1].plot(input_data['Year'], input_data['Water'], marker='o', label='Water')
    axes[1, 1].set_title('Input Usage Comparison')
    axes[1, 1].legend()
    plt.tight_layout()
    st.pyplot(fig)

def home():
     # Center-align all elements
```

```
# Logo
st.markdown("<div class='st-centered_img'><img src='https://www.freeiconspng.com/uploads/green-leaf-png-9.png' width='80'></div>", unsafe_allow_html=True)

# Text with size 100
st.markdown("<h2 class='st-centered' style='color: #5BEAB1;'>Farm Management System</h2>", unsafe_allow_html=True)

# Heading
st.markdown("<h1 class='st-centered' style='color: #5BEAB1;'></h1>", unsafe_allow_html=True)

# Simple center-aligned text
st.markdown("<p class='st-centered' style='color: #5BEAB1;'> A comprehensive solution to streamline all farming procedures and activities. Optimize productivity, make informed decisions, and embrace the future of farming.</p>", unsafe_allow_html=True)

# Button to display a paragraph
col4, col5,col6 =st.columns(3)
with col5:
    if st.button("About the application"):
        st.header("Field Mapping and Delineation")
        st.write("Effortlessly map and delineate your fields using Farm_era's intuitive interface. Visualize your land boundaries and effectively manage your fields.")
        st.header("Crop Planning and Rotation Management")
        st.write("Plan your crop planting schedules and manage rotation cycles with ease. Keep track of different crop varieties and maintain optimal soil health for improved yields.")
```

st.header("Input Usage and Crop History")

st.write("Record and track the usage of fertilizers, pesticides, water, and other inputs. Maintain a detailed crop history for compliance and better resource management.")

st.header("Pest and Disease Management")

st.write("Receive timely alerts and recommendations for pest and disease control. Access integrated databases to identify and address issues promptly, safeguarding your crops.")

st.header("Weather Tracking")

st.write("Stay informed about changing weather conditions with real-time updates and forecasts. Utilize historical weather data to make informed decisions for your farming activities.")

st.header("Irrigation Scheduling and Monitoring")

st.write("Optimize water usage and enhance irrigation practices. Set up customized schedules based on crop needs and monitor water usage to promote sustainable farming.")

st.write("Farm_era is your reliable companion, providing seamless integration, comprehensive features, and user-friendly analytics. Simplify your crop and field management, increase efficiency, and embrace the future of farming with Farm_era.")

```
    # Heading in a different color
    st.markdown("<h2 class='st-centered' style='color: white;'>Monitor Your Farm</h2>", unsafe_allow_html=True)
    # Boxes with hover effect
    col1, col2, col3 = st.columns(3)
    with col1:
# Field mapping and delineation
def field_mapping():
    st.header("MAPPINGT")
    # Sidebar
```

```
st.sidebar.title("Enter GPS Coordinates")
# Input GPS coordinates
latitude = st.sidebar.number_input('Enter latitude', -90.0, 90.0, 0.0)
longitude = st.sidebar.number_input('Enter longitude', -180.0, 180.0, 0.0)
# Create a map centered at the input coordinates
m = folium.Map(location=[latitude, longitude], zoom_start=13)
# Add drawing tool to the map
draw = plugins.Draw(export=True)
draw.add_to(m)
# Display the map in the Streamlit app
folium_static(m)
# File uploader for the GeoJSON file
st.sidebar.title("Upload GeoJSON")
uploaded_file = st.sidebar.file_uploader("Upload the GeoJSON file")
if uploaded_file is not None:
    # Load the GeoJSON file
    geojson_data = json.load(uploaded_file)
    i=0
    # Get the coordinates of the polygon
    for feature in geojson_data['features']:
        i+=1
        if feature['geometry']['type'] == 'Polygon':
            # Get the vertices
            vertices = feature['geometry']['coordinates'][0]
            vertices = [(lon, lat) for lon, lat in vertices]  # Flip coordinates
            # Calculate and display the area if there are enough vertices
            if len(vertices) >= 3:
                polygon = Polygon(vertices)
                area = polygon.area
```

```
            st.header(f"Area of the Field{i}: \n{area} square units")
        # Calculate and display the distances between the vertices
        for i in range(len(vertices) - 1):
            distance = geodesic(vertices[i], vertices[i+1]).miles
            st.header("boundary distance")
            st.write(f"Distance between point {i+1} and point {i+2}: {distance} miles")
def crop_planning():
    # Implement crop planning logic
    st.header("Crop Planning")
    # Sidebar inputs
    st.sidebar.title("Inputs")
    # Get number of crops from user
    num_crops = st.sidebar.number_input("Number of Crops", min_value=1, step=1, value=3)
    # Create lists to store crop data
    crops = []
    areas = []
    pod_counts = []
    grain_counts = []
    grain_weights = []
    rainfall_data_per_crop = []
    # Collect crop data from user inputs
    for i in range(num_crops):
        crop = st.sidebar.text_input(f"Crop {i+1} Name")
        area = st.sidebar.number_input(f"Crop {i+1} Area (in acres)", min_value=1, step=1, value=10)
        pod_count = st.sidebar.number_input(f"Crop {i+1} Pod Count (Average of 5 measurements)", min_value=1, step=1, value=10)
```

```
        grain_count = st.sidebar.number_input(f"Crop {i+1} Grain Count (Average of 20 measurements)", min_value=1, step=1, value=50)
        grain_weight = st.sidebar.number_input(f"Crop {i+1} Grain Weight (per grain)", min_value=1, step=1, value=10)
        crops.append(crop)
        areas.append(area)
        pod_counts.append(pod_count)
        grain_counts.append(grain_count)
        grain_weights.append(grain_weight)
        # Collect rainfall data for each crop
        months = st.sidebar.multiselect(f"Select months for Crop {i+1}", ["Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"])
        rainfall_data = []
        for month in months:
            month_rainfall = st.sidebar.number_input(f"Crop {i+1} - {month} Rainfall (in mm)", min_value=0, step=1, value=0, key=f"rainfall_{i}_{month}")
            rainfall_data.append(month_rainfall)
        rainfall_data_per_crop.append(rainfall_data)
    # Perform crop planning calculations
    crop_yields = [(areas[i] * pod_counts[i] * grain_counts[i] * grain_weights[i]) / 10000 for i in range(num_crops)]  # Placeholder calculation
    # Display crop planning results
    st.header("Crop Planning Results")
    # Display crop yields in a table
    df_yields = pd.DataFrame({"Crop": crops, "Yield": crop_yields})
    st.subheader("Crop Yields")
    st.dataframe(df_yields)
    # Display pie chart to compare crop yields
    st.subheader("Crop Yield Comparison")
```

```
    fig, ax = plt.subplots()
    ax.pie(crop_yields, labels=crops, autopct='%1.1f%%', startangle=90)
    ax.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
    ax.set_title("Crop Yield Comparison")
    st.pyplot(fig)
    # Display graphs
    st.header("Graphs")
    # Bar chart for individual crop yields
    fig, ax = plt.subplots()
    sns.barplot(data=df_yields, x="Crop", y="Yield", ax=ax)
    ax.set_title("Crop Yields")
    st.pyplot(fig)
    # Line chart for monthly rainfall
    fig, ax = plt.subplots()
    for i, crop in enumerate(crops):
        data = pd.DataFrame({"Month": [month for month, rainfall in zip(months, rainfall_data_per_crop[i]) if rainfall != 0],
                             "Rainfall": [rainfall for rainfall in rainfall_data_per_crop[i] if rainfall != 0]})
        sns.lineplot(data=data, x="Month", y="Rainfall", ax=ax, label=crop)
    ax.set_title("Monthly Rainfall")
    ax.set_ylabel("Rainfall (mm)")
    ax.legend()
    st.pyplot(fig)
    # Display images
    st.header("Images")
    uploaded_images_per_crop = []
    for i in range(num_crops):
```

```
        uploaded_images = st.sidebar.file_uploader(f"Upload images for Crop {i+1}",
accept_multiple_files=True)
        uploaded_images_per_crop.extend(uploaded_images)
    num_images_per_row = 3
    num_images = len(uploaded_images_per_crop)
    num_rows = (num_images + num_images_per_row - 1) // num_images_per_row
    for row in range(num_rows):
        cols = st.columns(num_images_per_row)
        for col in range(num_images_per_row):
            index = row * num_images_per_row + col
            if index < num_images:
                cols[col].image(uploaded_images_per_crop[index], caption=f"Image
{index+1}", use_column_width=True)
# Recording crop history and input usage
def crop_history():
    # Implement crop history recording logic
    st.header("Crop History and Input Usage")
    # Load existing data from CSV or create an empty dataframe
    try:
        crop_data = pd.read_csv('crop_data.csv')
    except FileNotFoundError:
        crop_data = pd.DataFrame(columns=['Year', 'Crop', 'Fertilizers', 'Pesticides',
'Water']
    # Display existing crop data
    if not crop_data.empty:
        display_crop_history(crop_data)
    # Collect new data from user inputs
    st.subheader("Enter Crop History and Input Usage")
```

```
    year = st.number_input("Year", min_value=1900, max_value=2100, value=2023, step=1)
    crop = st.text_input("Crop")
    fertilizers = st.number_input("Fertilizers (in kg)", min_value=0.0, value=0.0, step=0.1)
    pesticides = st.number_input("Pesticides (in kg)", min_value=0.0, value=0.0, step=0.1)
    water = st.number_input("Water (in mm)", min_value=0.0, value=0.0, step=0.1)
    # Save new data to the dataframe and CSV
    if st.button("Save"):
        new_entry = pd.DataFrame([[year, crop, fertilizers, pesticides, water]], columns=crop_data.columns)
        crop_data = pd.concat([crop_data, new_entry], ignore_index=True)
        crop_data.to_csv('crop_data.csv', index=False)
        st.success("Data saved successfully!")
        # Display updated crop data
        display_crop_history(crop_data)
# Pest and disease management
def pest_management():
    # Implement pest management logic
    st.header("Pest and Disease Management")
    st.sidebar.header("Pest and Disease Management System")
    farm_name = st.sidebar.text_input("Farm Name")
    date = st.sidebar.date_input("Date")
    # Pest and Disease Inputs
    st.sidebar.subheader("Pest and Disease Inputs")
    pest_input = st.sidebar.text_input("Pests (comma-separated)")
    disease_input = st.sidebar.text_input("Diseases (comma-separated)")
```

```
    # Pest Management Options
    st.sidebar.subheader("Pest Management Options")
    pest_management_option_input = st.sidebar.text_input("Pest Management Options (comma-separated)")
    pest_management_options = [option.strip() for option in pest_management_option_input.split(",")]
    # Real-time Tracking
    st.sidebar.subheader("Real-time Tracking")
    temperature = st.sidebar.number_input("Temperature (°C)", min_value=-50, max_value=50, value=25, step=1)
    humidity = st.sidebar.number_input("Humidity (%)", min_value=0, max_value=100, value=50, step=1)
    rainfall = st.sidebar.number_input("Rainfall (mm)", min_value=0, value=0, step=1)
    # Data Collection
    st.header("Data Collection")
    # Create a DataFrame to store the collected data
    data = pd.DataFrame({
        "Date": [date],
        "Farm Name": [farm_name],
        "Pests": [pest_input],
        "Diseases": [disease_input],
        "Temperature (°C)": [temperature],
        "Humidity (%)": [humidity],
        "Rainfall (mm)": [rainfall]
    })
    # Display the collected data
    st.write(data)
    # Interactive Graphs
    st.header("Interactive Graphs")
```

```
# Generate random data for demonstration
num_days = 30
dates = pd.date_range(end=pd.to_datetime(date), periods=num_days, freq="D")
temperatures = np.random.normal(25, 5, num_days)
humidities = np.random.normal(50, 10, num_days)
rainfalls = np.random.randint(0, 10, num_days)
# Create a DataFrame with random data
df = pd.DataFrame({
    "Date": dates,
    "Temperature (°C)": temperatures,
    "Humidity (%)": humidities,
    "Rainfall (mm)": rainfalls
})
# Line plot for temperature
fig, ax1 = plt.subplots()
ax1.plot(df["Date"], df["Temperature (°C)"], color="tab:red")
ax1.set_xlabel("Date")
ax1.set_ylabel("Temperature (°C)", color="tab:red")
ax1.tick_params(axis="y", labelcolor="tab:red")
# Bar plot for humidity
ax2 = ax1.twinx()
ax2.bar(df["Date"], df["Humidity (%)"], color="tab:blue", alpha=0.3)
ax2.set_ylabel("Humidity (%)", color="tab:blue")
ax2.tick_params(axis="y", labelcolor="tab:blue")
# Rotate x-axis labels for better readability
plt.xticks(rotation=45)
# Display the graph
st.pyplot(fig)
```

```
# Line plot for rainfall
fig2, ax3 = plt.subplots()
ax3.plot(df["Date"], df["Rainfall (mm)"], color="tab:green")
ax3.set_xlabel("Date")
ax3.set_ylabel("Rainfall (mm)", color="tab:green")
ax3.tick_params(axis="y", labelcolor="tab:green")
# Rotate x-axis labels for better readability
plt.xticks(rotation=45)
# Display the graph
st.pyplot(fig2)
# Additional visualizations can be added here
# Pest Management Result
if pest_management_options:
    st.header("Pest Management Result")

    # Extract individual pests from pest_input
    pests = [pest.strip() for pest in pest_input.split(",")]

    # Check the length of pest management options
    if len(pest_management_options) >= len(pests):
        # Generate sample pest management data
        pest_management_data = pd.DataFrame({
            "Pest": pests,
            "Pest Management Option": pest_management_options[:len(pests)],
            "Effectiveness": np.random.uniform(0.5, 0.9, len(pests))
        })  # Display the pest management result as a table
        st.table(pest_management_data)
    else:
        # Use available pest management options for multiple pests
```

```
            num_pests = len(pests)
            num_options = len(pest_management_options)
            repetition = num_pests // num_options
            remainder = num_pests % num_options
            pest_management_options_extended = pest_management_options * repetition +
pest_management_options[:remainder]
            pest_management_data = pd.DataFrame({
                "Pest": pests,
                "Pest Management Option": pest_management_options_extended,
                "Effectiveness": np.random.uniform(0.5, 0.9, num_pests)
            })
            # Display the pest management result as a table
            st.table(pest_management_data)
    else:
        st.info("No pest management options provided.")
    st.header("Raw Data")
    st.write(df)
    # Show the Streamlit application
    st.sidebar.markdown("---")
    st.sidebar.markdown("Developed by Sowmya Surapaneni")
# Weather tracking and integration
def weather_tracking():
    # Implement weather tracking logic
    st.header("Weather Tracking and Integration")
    st.sidebar.header("Weather Tracking")
    city = st.sidebar.text_input("City", value="New York")
    start_date = st.sidebar.date_input("Start Date")
    end_date = st.sidebar.date_input("End Date")
```

```
# Data Collection
st.header("Data Collection")
# Generate random weather data
num_days = (end_date - start_date).days + 1
dates = pd.date_range(start=start_date, end=end_date, freq="D")
temperatures = np.random.normal(25, 5, num_days)
humidities = np.random.normal(50, 10, num_days)
rainfalls = np.random.randint(0, 10, num_days)
# Create a DataFrame with random data
weather_data = pd.DataFrame({
    "Date": dates,
    "Temperature (°C)": temperatures,
    "Humidity (%)": humidities,
    "Rainfall (mm)": rainfalls
})
# Display the collected data
st.write(weather_data)
# Visualizations
st.header("Visualizations")
# Line plot for temperature
st.subheader("Temperature")
fig_temp = plt.figure(figsize=(10, 6))
plt.plot(weather_data["Date"], weather_data["Temperature (°C)"])
plt.xlabel("Date")
plt.ylabel("Temperature (°C)")
plt.title("Temperature Variation")
plt.xticks(rotation=45)
st.pyplot(fig_temp)
```

```
# Line plot for humidity
st.subheader("Humidity")
fig_humidity = plt.figure(figsize=(10, 6))
plt.plot(weather_data["Date"], weather_data["Humidity (%)"])
plt.xlabel("Date")
plt.ylabel("Humidity (%)")
plt.title("Humidity Variation")
plt.xticks(rotation=45)
st.pyplot(fig_humidity)
# Bar plot for rainfall
st.subheader("Rainfall")
fig_rainfall = plt.figure(figsize=(10, 6))
plt.bar(weather_data["Date"], weather_data["Rainfall (mm)"])
plt.xlabel("Date")
plt.ylabel("Rainfall (mm)")
plt.title("Daily Rainfall")
plt.xticks(rotation=45)
st.pyplot(fig_rainfall)
st.header("Summary Statistics")
# Data summary
st.subheader("Weather Data Summary")
st.write(weather_data.describe())
# Correlation heatmap
st.subheader("Correlation Heatmap")
corr = weather_data.corr()
fig_corr = plt.figure(figsize=(8, 6))
sns.heatmap(corr, annot=True, cmap="coolwarm")
st.pyplot(fig_corr)
```

```
# Irrigation scheduling and monitoring
def irrigation_management():
    # Implement irrigation management logic
    st.header("Irrigation Scheduling and Monitoring")
    number1 = st.number_input("Enter the water content of the feild 1:", min_value=0, max_value=10, value=0, step=1)
    number2 = st.number_input("Enter the water content of the feild 2:", min_value=0, max_value=10, value=0, step=1)
    number3 = st.number_input("Enter the water content of the feild 3:", min_value=0, max_value=10, value=0, step=1)
    number4 = st.number_input("Enter the water content of the feild 4:", min_value=0, max_value=10, value=0, step=1)
    if st.button('start monitoring'):
        os.system("python cam.py")
    if st.button('start irrigation'):
# Delete previous values in the text file
        with open("water_flow_data.txt", "w") as file:
            file.write("")  # This line clears the contents of the file
        # Save the numbers to the text file
        with open("water_flow_data.txt", "a") as file:
            file.write(str(number1) + "\n")
            file.write(str(number2) + "\n")
            file.write(str(number3) + "\n")
            file.write(str(number4) + "\n")
# Display success message
        st.success("irrigation Started")
    if st.button('irrigation monitor'):
        os.system("streamlit run test.py")
```

```
# Sidebar navigation
menu_options = ["Home", "Field Mapping", "Crop Planning", "Crop History",
                "Pest Management", "Weather Tracking", "Irrigation Management"]
# Display the menu options
selected_menu = st.sidebar.selectbox("Select an option:", menu_options)
# Display corresponding page based on selected menu option
if selected_menu == "Home":
    home()
elif selected_menu == "Field Mapping":
    field_mapping()
elif selected_menu == "Crop Planning":
    crop_planning()
elif selected_menu == "Crop History":
    crop_history()
elif selected_menu == "Pest Management":
    pest_management()
elif selected_menu == "Weather Tracking":
    weather_tracking()
elif selected_menu == "Irrigation Management":
    irrigation_management()
menu_options = st.sidebar.multiselect("Select options:", ["Home", "Field Mapping",
"Crop Planning", "Crop History", "Pest Management","Weather Tracking", "Irrigation
Management"])
# Display corresponding pages based on selected menu options
if "Home" in menu_options:
    home()
if "Field Mapping" in menu_options:
    field_mapping()
```

```
if "Crop Planning" in menu_options:
    crop_planning()
if "Crop History" in menu_options:
    crop_history()
if "Pest Management" in menu_options:
    pest_management()
if "Weather Tracking" in menu_options:
    weather_tracking()
if "Irrigation Management" in menu_options:
    irrigation_management()
```

IRRIGATION.PY

```
import random
import csv
import time
# Constants for simulation
NUM_ZONES = 4  # Number of irrigation zones
MOISTURE_RANGE = (20, 80)  # Range of moisture values (%)
WATER_FLOW_RANGE = (1, 10)  # Range of water flow values (liters per minute)
# Function to generate random sensor data
def generate_sensor_data():
    moisture_data = [random.randint(*MOISTURE_RANGE) for _ in
range(NUM_ZONES)]
    water_flow_data = [random.uniform(*WATER_FLOW_RANGE) for _ in
range(NUM_ZONES)]
    return moisture_data, water_flow_data
# Function to update and store sensor data
def update_sensor_data():
    while True:
        # Generate random sensor data
        moisture_data, water_flow_data = generate_sensor_data()
        # Store the updated sensor data in a data file (e.g., CSV)
        with open('sensor_data.csv', 'w', newline='') as file:
            writer = csv.writer(file)
            writer.writerow(['Zone', 'Moisture', 'Water Flow'])
            for i in range(NUM_ZONES):
```

```
            writer.writerow([i+1, moisture_data[i], water_flow_data[i]])
        # Wait for some time before updating again
        time.sleep(3)  # Adjust the sleep time as per your requirements
if __name__ == "__main__":
    update_sensor_data()
```