A

GEN AI AND LLM ASSIGNMENT REPORT

on

Agricultural Chatbot

Submitted by:

Deepanshu Mehlawat (220633)
Anup Kumar (220459)
Ankit Rathi (220474)
Saurabh (220450)

under mentorship of

Dr. Manisha Saini
(Professor)

Department of Computer Science Engineering
School of Engineering and Technology
BML MUNJAL UNIVERSITY, GURUGRAM (INDIA)

May 2025

ABSTRACT

The agricultural sector continually faces challenges of climate change, pest infestations, or rather lack of systems for personalized advisories, all factors that require innovative solutions for sustainable practitioners. This project presents the building of an Artificial Intelligence-powered assistant for farmers, which relies on local LLMs and RAG to provide localized and precise agricultural advice. This system employs the Mistral 7B Instruct model, served from the Ollama platform, hence specifying the local operation and GPU acceleration, addressing the usual accessibility and data privacy concerns. It features a RAG pipeline that answers farmer queries in natural language based on a curated corpus of agricultural PDF documents, alongside a prompt engineering module for producing simulated field reports and pest suggestions. The system is made available for user interaction through a chatbot interface, which may find an option for translating responses to Hindi from the assistant. The report would then cover the architecture of this system, the implementation of RAG and prompt engineering components, and a discussion on the attained functionality. It would also highlight the major bottlenecks faced during development, especially those concerned with the initial phases of GPU acceleration setup that were eased after migration to Ollama. The project demonstrates the potential of locally deployed generative AI to empower farmers with tailored information, establishing a foundation for a more robust and comprehensive agricultural advisory tool.

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

The world agricultural ecosystem that is the backbone of human consumption and financial stability is under extreme duress. Global warming is inducing irregular climatic patterns that produce more instances of floods and droughts as well as temperature vagaries, all of which play a critical role in determining crop viability and production [1]. Simultaneously, the changed dynamics of pests and the incidence of resistant crop diseases remain a challenge to the production of food. Traditional farming techniques rely on generalized guidelines that may not turn out ideal for the specific farm conditions and therefore yield less efficient resource utilization and less-than-ideal yields. The need for timely, individualized, and data-driven advisory systems for farmers is greater than ever before in such a context in order to preserve food security as well as promote sustainable farming practices.

Farmers throughout the world and particularly in the resource-constrained environment face immense challenges due to the mounting dynamics of climate change, repeat pest infestations, and the lack of accessible and tailored crop management information. While there is a plethora of agricultural information in scientific journals, extension manuals, and websites, it is scattered, not readily accessible, nor presented in a format that is farm understandable. The absence of information limits the enactment of best practices and adaptation techniques that facilitate agricultural sustainability.

The aim of this project is specifically to resolve the challenges mentioned earlier by creating and developing an innovative AI assistant tailored for farmers. First and foremost, the project aims to develop and deploy a straightforward AI assistant that makes full use of the potential of Large Language Models (LLMs) in interpreting naturally worded questions by farmers. Secondly, a primary aim is that the assistant should provide context-sensitive pertinent, trustworthy, and actionable advice related to the chosen specific agricultural domains with an initial emphasis on such topics as the best pest control methods and how common crop diseases might be managed. Thirdly, the process of the project is the utilization of a Retrieval-Augmented Generation (RAG) pipeline that is based on a collection of authoritative agricultural journals and papers as a backing in ensuring all the offered advice is supported by and aligned with valid and credible sources. Moreover, a significant aim is to utilize current advanced prompt engineering techniques in the generation of customized and personalized outputs such as the generation of synthetic field reports that will benefit farm administration.

The aim of this project is the creation of a proof-of-concept AI assistant as a base system on which subsequent changes and additions can be built. The current functional range of the assistant is in a few core domains. In the case of agricultural domains, the system is primarily engaged in response to farmer queries regarding general crop diseases and general pest control methods, drawing its information from a handpicked selection of PDF documents; secondly, it is capable of dispensing simulated reports as a response based on user input. Language-wise, farmer queries receive primary reception and processing in the English language, with Hindi translation for the output of the assistant offered as an extra, optional facility. The Large Language Model on which the system is based is the Mistral 7B Instruct model specifically the variant mistral:7b-instruct-q4_K_M GGUF and is served and deployed using the Ollama framework. The knowledge base used in the Retrieval-Augmented Generation system is presently limited in its depth and range to the materials of the specific PDF files within the project's specific data/ folder.

The following is a comprehensive description of the development and creation process of the Agri AI Assistant. Section 2 presents a concise but relevant review of the current literature concerning the numerous applications of Artificial Intelligence in the agricultural context. Section 3 then offers a thorough analysis of the general system architecture, the individual design decisions taken, and the technologies that support the functionality of the assistant. Section 4 describes the practical details of the most critical software components involved, such as but not limited to the Retrieval-Augmented Generation pipeline and the prompt engineering techniques used. Next, Section 5 shows the test results obtained and discusses extensively the operational functioning and performance aspects of the system. Section 6 is committed to describing the range of challenges that have been encountered throughout the development process and the remedies and solutions put in place. Finally, Section 7 concludes the report by providing an overview of the accomplishments of the project and suggesting possible directions and avenues for subsequent work and further enhancement of the Agri AI Assistant.

LITERATURE REVIEW

The integration of Artificial Intelligence (AI) into agriculture has shown transformative potential, addressing various challenges from precision farming to supply chain optimization. Large Language Models (LLMs) represent a recent advancement in AI, offering sophisticated natural language understanding and generation capabilities that can be harnessed for agricultural advisory systems.

Generative AI, particularly LLMs, has demonstrated proficiency in tasks such as knowledge retrieval, summarization, and content creation [2]. In agriculture, these models can process vast amounts of textual data from research articles, farming guides, and expert opinions to provide synthesized information to farmers. For instance, LLMs can be used to create chatbots that answer farmer queries, generate crop-specific advice, or even help in drafting farm management plans [3]. The ability of LLMs to understand and generate human-like text makes them suitable for creating interactive and accessible advisory tools, bridging the gap between complex agricultural science and practical farm-level application. Open-source models, when run locally, further democratize access to these powerful technologies, mitigating concerns about data privacy and connectivity often faced in rural agricultural settings [4].

While LLMs possess extensive general knowledge, they can sometimes "hallucinate" or provide outdated information, especially on specialized topics. Retrieval-Augmented Generation (RAG) is a technique designed to mitigate this by grounding LLM responses in external, verified knowledge sources [5]. In a RAG system, when a query is received, relevant information is first retrieved from a domain-specific corpus (e.g., agricultural research papers, extension documents). This retrieved context is then provided to the LLM along with the original query, enabling the model to generate more accurate, contextual, and reliable answers [5]. For agricultural applications, RAG is particularly valuable as it allows the AI assistant to base its advice on curated, up-to-date, and regionally relevant agricultural information, thereby enhancing the trustworthiness and utility of the advisory system.
The advent of efficient open-source LLMs and runtime environments like Ollama has made it feasible to deploy these models on local hardware, including consumer-grade GPUs [6]. This local deployment model offers several advantages for agricultural applications. Firstly, it reduces dependency on cloud-based services, which can be beneficial in areas with limited or unreliable internet connectivity. Secondly, it addresses data privacy concerns, as farm-specific data or queries do not need to leave the local system. Furthermore, local LLMs allow for greater customization and fine-tuning on specific agricultural datasets if desired, although this project utilizes a pre-trained model with RAG. Platforms like Ollama simplify the process of downloading, managing, and serving these local LLMs, making them accessible to a broader range of developers and researchers aiming to build domain-specific AI tools [7]. Our project leverages this approach to provide a practical and accessible AI assistant solution.

SYSTEM DESIGN AND ARCHITECTURE

The development of the Agri AI Assistant was guided by the need for a robust, accessible, and locally deployable solution. This section outlines the chosen agricultural domains, the overall system architecture, the technology stack selected, and the rationale behind key design choices.

3.1. Chosen Agricultural Domains

To provide focused and practical advice within the project's scope, specific agricultural domains were selected. The primary areas of concentration are **Pest Management Advisory** and **Crop Disease Diagnosis (via textual queries)**. These domains were chosen due to their significant impact on crop yield and the high frequency of farmer queries related to these issues. Furthermore, readily available public information in the form of extension service documents and agricultural guides for these topics allowed for the creation of a foundational knowledge base for the Retrieval-Augmented Generation (RAG) component. The system also supports the generation of **Simulated Field Reports** as an example of LLM-driven content creation.

3.2. Overall System Architecture

The Agri AI Assistant is designed as a modular system, facilitating interaction through a web-based interface powered by Streamlit. The core information processing pipeline is depicted in Figure 1

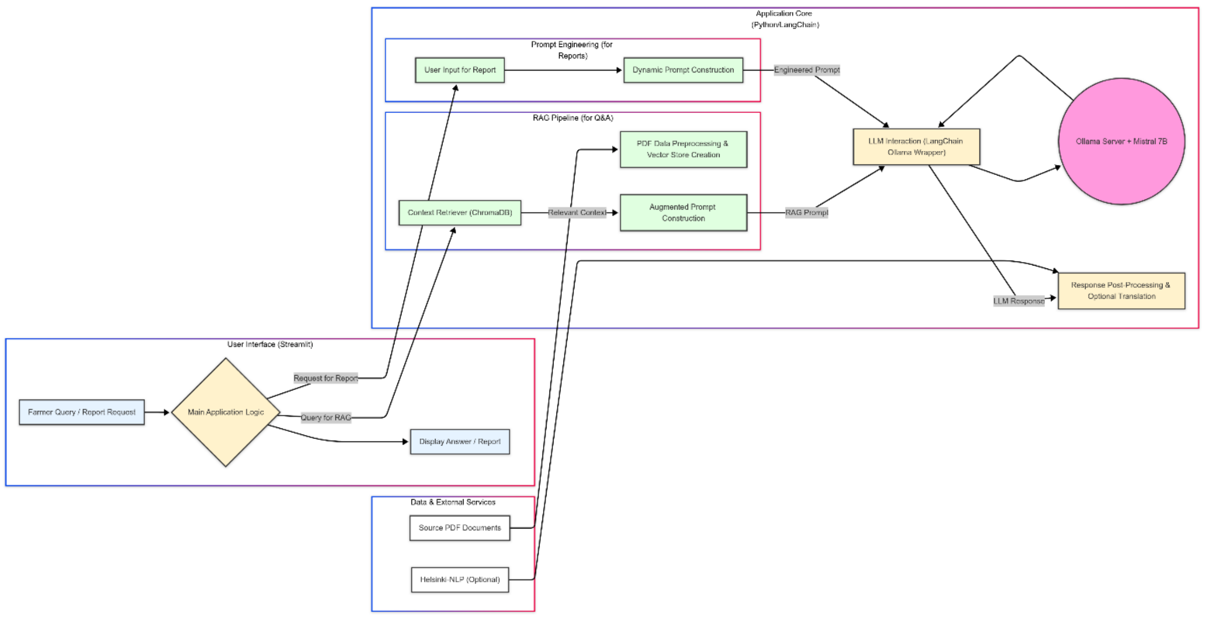

Application Core (Python/LangChain)
Prompt Engineering (for Reports)
User Input for Report
Dynamic Prompt Construction
Engineered Prompt
RAG Pipeline (for Q&A)
PDF Data Preprocessing & Vector Store Creation
Context Retriever (ChromaDB)
Relevant Context
Augmented Prompt Construction
RAG Prompt
LLM Interaction (LangChain Ollama Wrapper)
Ollama Server + Mistral 7B
LLM Response
Response Post-Processing & Optional Translation
User Interface (Streamlit)
Farmer Query / Report Request
Main Application Logic
Request for Report
Query for RAG
Display Answer / Report
Data & External Services
Source PDF Documents
Helsinki-NLP (Optional)

Figure 1 - System Architecture

The operational flow begins when a farmer interacts with the Streamlit user interface, selecting a task such as asking a question or requesting a report. For knowledge-based queries, the user's natural language input is processed by the RAG pipeline. This involves retrieving relevant text chunks from a pre-indexed vector store of agricultural PDF documents. The retrieved context, along with the original query, is then formulated into an augmented prompt. For tasks requiring generative output, such as simulated field reports, a specialized prompt is constructed using predefined templates and user-provided parameters via the prompt engineering module.

Both types of prompts are then dispatched to the locally running Large Language Model, served by the Ollama platform. The LLM processes the input and generates a textual response. This response is subsequently formatted for clarity and, if requested by the user, translated into Hindi using an external translation model. Finally, the processed answer or generated report is displayed back to the farmer through the Streamlit interface. This architecture emphasizes leveraging a local LLM for data privacy and accessibility, while the RAG component aims to enhance the reliability and contextuality of the advice provided.

3.3. Technology Stack

The selection of technologies was driven by the requirements for local deployment, ease of use, and effective LLM interaction:

Category	**Technology**	**Justification**
LLM and Serving	Mistral 7B Instruct Q4_K_M (GGUF)	Strong performance-to-size ratio.
	Ollama	Simplicity in downloading, managing, and serving LLMs locally. Automatic GPU acceleration, circumventing build complexities.
Orchestration Framework	LangChain	Primary framework for building the application, especially the RAG pipeline. Extensive modules for document loading, text splitting, embeddings, vector store interaction, and chain management.
Document Loading	PyPDFLoader, DirectoryLoader	LangChain modules used for loading documents.
Text Splitting	RecursiveCharacterTextSplitter	LangChain module used for splitting text.
LLM Wrapper	Ollama LLM wrapper	LangChain component for interacting with the locally served Ollama model.
Embedding Model	Sentence Transformers (all-MiniLM-L6-v2)	Balance of performance and efficiency, suitable for local execution. Used for generating document and query embeddings.
Vector Store	ChromaDB	Local vector database for storing and retrieving document embeddings. Ease of setup and integration with LangChain.

User Interface	Streamlit	Interactive web-based user interface, enabling easy query input and response viewing for farmers.
Programming Language	Python (v3.10.16)	Extensive support for AI/ML libraries.
Translation (Bonus)	Hugging Face transformers	Library used for implementing Hindi translation.
	Helsinki-NLP/opus-mt-en-hi	Specific model from Hugging Face transformers used for English to Hindi translation.

Table 1 - Tech Stack Details

3.4. Design Choices

Several key design decisions shaped the development of the assistant:

- **Local First with Ollama:** The initial attempts to compile llama-cpp-python with CUDA support directly on Windows proved to be exceedingly complex due to toolchain and environment conflicts. Consequently, a strategic shift was made to utilize **Ollama**. This platform abstracts the intricacies of LLM serving and GPU management, providing a stable and reliable local LLM endpoint. This choice prioritized functional GPU acceleration and development velocity over granular control of the C++ backend.
- **RAG for Grounded Responses:** To enhance the reliability and factual accuracy of the LLM's advice, a **Retrieval-Augmented Generation (RAG)** pipeline was implemented. This ensures that responses to farmer queries are primarily based on the content of curated agricultural documents rather than solely on the LLM's parametric knowledge, thereby reducing the likelihood of hallucinations.
- **Modular Component Implementation:** The system was designed with distinct modules for data ingestion, RAG, prompt engineering, and UI, allowing for easier development, testing, and potential future expansion.
- **User-Friendly Interface:** Streamlit was selected to create an accessible and intuitive interface, requiring minimal technical expertise from the end-user (farmer).
- **Focus on Core Functionality:** Given the project timeline, the emphasis was placed on establishing a working RAG system and a prompt engineering module as per the assignment requirements, with translation as a bonus. More advanced features like vision integration were deferred.

IMPLEMENTATION DETAILS

This section details the practical implementation of the core components of the Agri AI Assistant, specifically the document-based Q&A system using the RAG pipeline and the use of prompt engineering for generating simulated agricultural reports.

4.1. Component 1: Document-Based Q&A (RAG Pipeline)

The RAG pipeline is central to providing contextual and reliable answers to farmer queries. Its implementation involved several stages:

- **4.1.1. Dataset Curation and Preparation:**

  The knowledge base for the RAG system was constructed from a collection of publicly available agricultural PDF documents. For this proof-of-concept, three primary documents were utilized:

  1. A 7-page University of California Integrated Pest Management (UC IPM) fact sheet on Aphid control.
  2. A 9-page Oregon State University Extension Service document on Managing Powdery Mildew in Greenhouse Tomato (EM9413.pdf).
  3. A comprehensive 134-page document on Soil Health Management. These documents were chosen for their authoritative content on common agricultural challenges and their varied lengths, allowing for testing the system's ability to handle both concise fact sheets and more extensive guides. The PDFs were placed in a local data/ directory.

- **4.1.2. Data Ingestion and Preprocessing:**

  The LangChain framework was employed for ingesting and processing these PDF documents. The DirectoryLoader was configured to load all .pdf files from the specified data/ path, using PyPDFLoader for individual file parsing. Once loaded, the raw text from the documents was segmented into smaller, manageable chunks using RecursiveCharacterTextSplitter. A chunk_size of 1000 characters and a chunk_overlap of 200 characters were configured. This chunking strategy aims to maintain semantic coherence within chunks while allowing for some contextual overlap between adjacent segments, which can be beneficial for retrieval.

- **4.1.3. Embedding and Vector Storage:**

  Each text chunk was then converted into a dense vector representation using the SentenceTransformerEmbeddings class, specifically loading the all-MiniLM-L6-v2 model. This model generates 384-dimensional embeddings that capture the semantic meaning of the text.

  The generated embeddings, along with their corresponding text chunks and metadata (source PDF, page number), were indexed and stored in a local ChromaDB vector store. ChromaDB was chosen for its lightweight nature and seamless integration with LangChain, persisting the vector database to the vectorstores/db_chroma directory. This allows the embeddings to be loaded quickly on subsequent app runs without re-processing the PDFs, unless the source documents change or the vector store is intentionally rebuilt.

- **4.1.4. Retrieval and Question Answering:**

  When a user submits a query through the "Chat about Crops (RAG)" interface, the RetrievalQA chain from LangChain orchestrates the question-answering process.

  1. The user's query is first embedded using the same all-MiniLM-L6-v2 model.
  2. The ChromaDB vector store is queried to find the top k (defaulted to 3 in this project) most semantically similar document chunks to the user's query embedding.
  3. These retrieved document chunks constitute the "context."
  4. A specific prompt template, designed for RAG, is then populated with this retrieved context and the original user question. The prompt instructs the LLM to answer based *only* on the provided context:

```
SYSTEM: You are an AI assistant providing agricultural advice. Use the following pieces of context ONLY to answer the user's question. If you don't know the answer from the context, just say that you don't know, don't try to make up an answer. Provide concise and actionable advice.

CONTEXT: {context}

QUESTION: {question}

ANSWER:
```

Figure 2 - Prompt for QA

  5. This augmented prompt is sent to the Ollama LLM instance (Mistral 7B).
  6. The LLM generates an answer, which is then returned to the user. The system also displays the source document snippets that contributed to the answer.

4.2. Component 2: Prompt Engineering for Simulated Reports

Beyond Q&A, the system utilizes prompt engineering to leverage the LLM's generative capabilities for creating structured text outputs, such as simulated field reports or pest suggestions. This component does not rely on the RAG pipeline's document retrieval but directly prompts the Ollama-served LLM.

- **4.2.1. Field Report Generation:**

  A Python function generate_crop_report_prompt was created to dynamically construct a detailed prompt. This function takes user inputs such as crop name, growth stage, recent weather conditions, and field observations. It then embeds these details into a structured prompt template:

```
SYSTEM: You are an agricultural expert simulating a field report.

TASK: Generate a brief field report based on the following details. Focus on potential risks and
next steps. Be concise and practical for a farmer. DETAILS:

- Crop: {crop_input}

- Growth Stage: {stage_input}

- Recent Weather: {weather_input}

- Observations: {observations_input}

REPORT:
```

Figure 3 - Prompt for crop report generation

When the user fills in the corresponding fields in the Streamlit UI and clicks "Generate Report," this complete prompt is sent to the llm.invoke() method of the Ollama LangChain wrapper. The LLM then generates a simulated field report based on the provided scenario.

- **4.2.2. Pest Suggestion Generation:**

  Similarly, a function generate_pest_suggestion_prompt constructs prompts for pest predictions. This prompt takes inputs like crop type, region, and weather forecast:

```
SYSTEM: You are an agricultural expert simulating a field report.

TASK: Generate a brief field report based on the following details. Focus on potential risks and
next steps. Be concise and practical for a farmer. DETAILS:

- Crop: {crop_input}

- Growth Stage: {stage_input}

- Recent Weather: {weather_input}

- Observations: {observations_input}

REPORT:
```

Figure 4 - Prompt for pest control report

The LLM uses its general knowledge, guided by the prompt structure and input details, to suggest potential pests and preventative measures. This demonstrates how prompt engineering can elicit specific types of structured information from the LLM without fine-tuning.

4.3. (Bonus) Hindi Translation Integration

To enhance accessibility, an optional Hindi translation feature was implemented for the LLM's outputs.

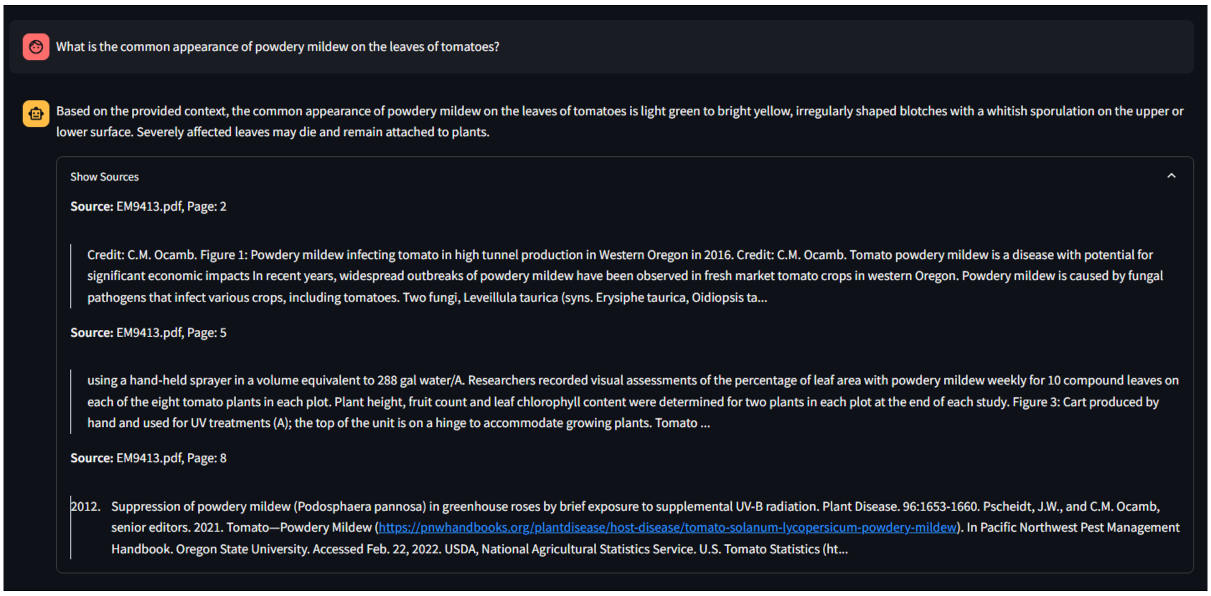
- **Model and Library:** This feature uses the Hugging Face transformers library and its pipeline API. The specific translation model employed is Helsinki-NLP/opus-mt-en-hi, a pre-trained model optimized for English-to-Hindi translation.
- **Implementation:** A function translate_text(text, translator) was created. It takes the English text generated by the LLM and the loaded translator pipeline as input.
- **Handling Long Text:** Initial testing revealed that the translation model might truncate very long responses. To address this, the translate_text function was modified to split the input English text into sentences using regular expressions (re.split(r'([.!?])\s*', text)). Each sentence is then translated individually, and the translated sentences are subsequently re-joined to form the complete Hindi output. This sentence-by-sentence approach helps to ensure that the entire content is translated without exceeding the model's input length limitations per translation call.
- **User Interface:** A checkbox in the UI allows users to toggle Hindi translation for the RAG Q&A responses.

RESULTS AND DISCUSSION

The evolution of the Agri AI Assistant ended with a functional prototype that could respond to farmer questions using a Retrieval-Augmented Generation (RAG) system and produce simulated agricultural articles using prompt engineering, both of which run using a locally executed Large Language Model served over the web using Ollama. This section describes the functionality observed and the performance of the system.

5.1 Functionality Demonstration

The primary user interface for the RAG system is a chatbot that is intended for straightforward interactivity. When given questions within the purview of the PDFs that have been ingested, the system evinced a capacity for retrieving applicable context and offering appropriate answers. An example question such as, "What is the common appearance of powdery mildew on the leaves of tomatoes?" might reasonably receive a response summarizing leaf coloration, white powdery growths, and possible leaf deformation, with the system offering an option for the user to see the resultant PDF snippets from which that information is taken. An example query regarding aphid control might entail the user requesting, "Organic ways that I might control aphids?" The assistant based on the UC IPM document might respond with such methods as introducing beneficial insects or using insecticidal soap, again with respectably attributing sources. The quality of such RAG responses corresponded directly with the clarity of information in the base documents.



What is the common appearance of powdery mildew on the leaves of tomatoes?

Based on the provided context, the common appearance of powdery mildew on the leaves of tomatoes is light green to bright yellow, irregularly shaped blotches with a whitish sporulation on the upper or lower surface. Severely affected leaves may die and remain attached to plants.

Show Sources

Source: EM9413.pdf, Page: 2

> Credit: C.M. Ocamb. Figure 1: Powdery mildew infecting tomato in high tunnel production in Western Oregon in 2016. Credit: C.M. Ocamb. Tomato powdery mildew is a disease with potential for significant economic impacts In recent years, widespread outbreaks of powdery mildew have been observed in fresh market tomato crops in western Oregon. Powdery mildew is caused by fungal pathogens that infect various crops, including tomatoes. Two fungi, Leveillula taurica (syns. Erysiphe taurica, Oidiopsis ta...

Source: EM9413.pdf, Page: 5

> using a hand-held sprayer in a volume equivalent to 288 gal water/A. Researchers recorded visual assessments of the percentage of leaf area with powdery mildew weekly for 10 compound leaves on each of the eight tomato plants in each plot. Plant height, fruit count and leaf chlorophyll content were determined for two plants in each plot at the end of each study. Figure 3: Cart produced by hand and used for UV treatments (A); the top of the unit is on a hinge to accommodate growing plants. Tomato ...

Source: EM9413.pdf, Page: 8

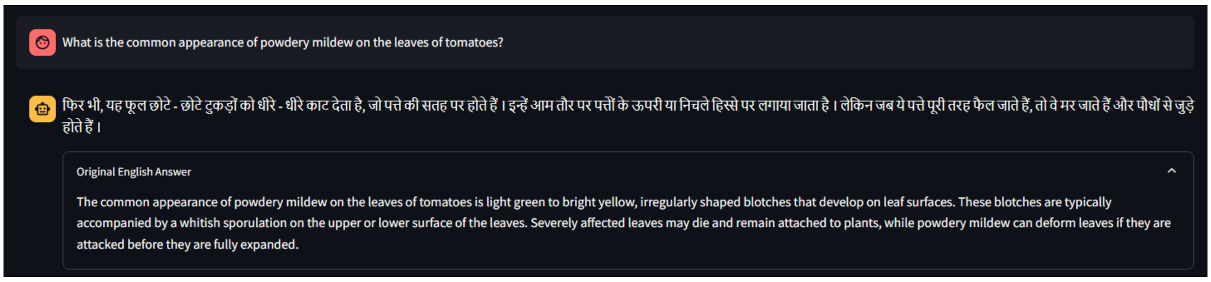
> 2012. Suppression of powdery mildew (Podosphaera pannosa) in greenhouse roses by brief exposure to supplemental UV-B radiation. Plant Disease. 96:1653-1660. Pscheidt, J.W., and C.M. Ocamb, senior editors. 2021. Tomato—Powdery Mildew (https://pnwhandbooks.org/plantdisease/host-disease/tomato-solanum-lycopersicum-powdery-mildew). In Pacific Northwest Pest Management Handbook. Oregon State University. Accessed Feb. 22, 2022. USDA, National Agricultural Statistics Service. U.S. Tomato Statistics (ht...

Figure 5 - cahtbot answer with RAG implemented

The prompt engineering module effectively produced structured textual outputs. With inputs for a sample field report such as "Tomato crop flowering stage, recent hot and humid weather conditions, observations of yellowing of leaves and whitefly presence," the system provided a succinct report. The report usually mentioned the crop stage, recorded the weather conditions, and raised possible risks such as the development of fungus due to high humidity or elevated pest pressures from whiteflies with a monitoring or specific action generally suggested. Likewise, the pest suggestion functionality with parameters such as "Cotton crop in Central

India with a monsoon approaching" would supply a list of probable pests such as aphids or bollworms along with a short prevention tip or a monitoring tip for each.

The Hindi translation facility, when active, exhibited the ability to translate the LLM's responses in English. For instance, an English response explaining the methods of improving soil health would be translated into the corresponding Hindi output. Though largely functional, the early versions had issues with translating lengthy multi-sentence responses fully, which was then rectified by adopting a sentence-by-sentence translation process within the translate_text method.

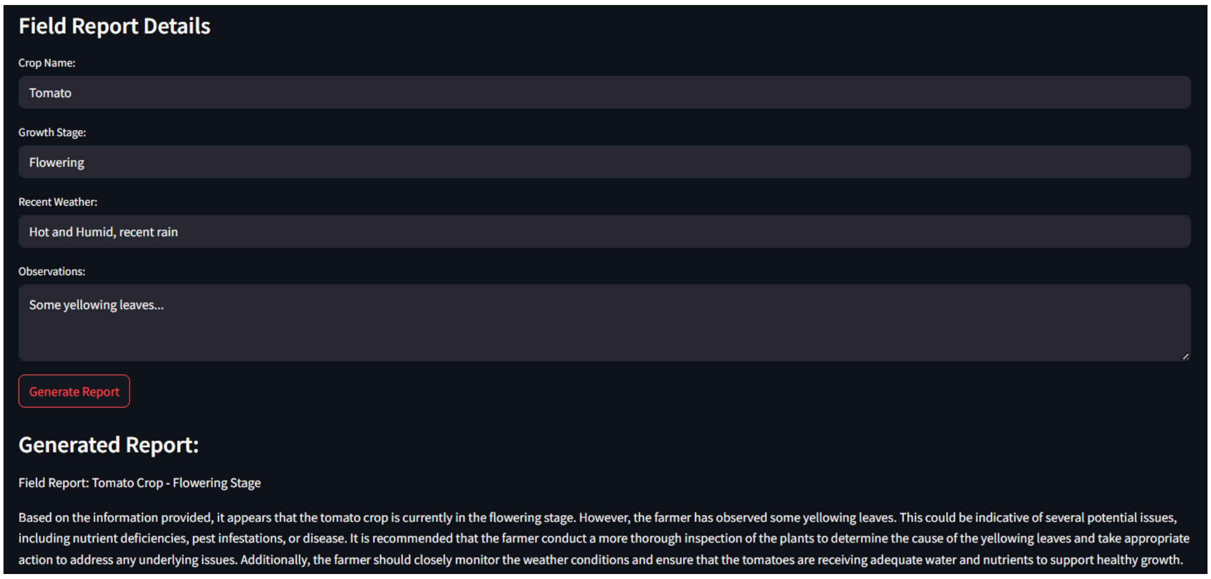

What is the common appearance of powdery mildew on the leaves of tomatoes?

फिर भी, यह फूल छोटे - छोटे टुकड़ों को धीरे - धीरे काट देता है, जो पत्ते की सतह पर होते हैं । इन्हें आम तौर पर पत्तों के ऊपरी या निचले हिस्से पर लगाया जाता है । लेकिन जब ये पत्ते पूरी तरह फैल जाते हैं, तो वे मर जाते हैं और पौधों से जुड़े होते हैं ।

Original English Answer

The common appearance of powdery mildew on the leaves of tomatoes is light green to bright yellow, irregularly shaped blotches that develop on leaf surfaces. These blotches are typically accompanied by a whitish sporulation on the upper or lower surface of the leaves. Severely affected leaves may die and remain attached to plants, while powdery mildew can deform leaves if they are attacked before they are fully expanded.

Figure 6 - Hindi Translated response


Field Report Details

Crop Name: Tomato

Growth Stage: Flowering

Recent Weather: Hot and Humid, recent rain

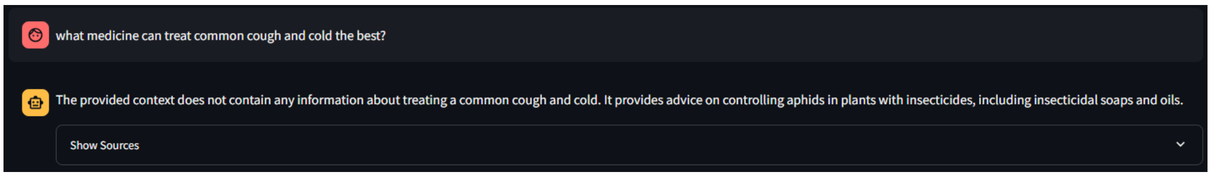
Observations: Some yellowing leaves...

Generated Report:

Field Report: Tomato Crop - Flowering Stage

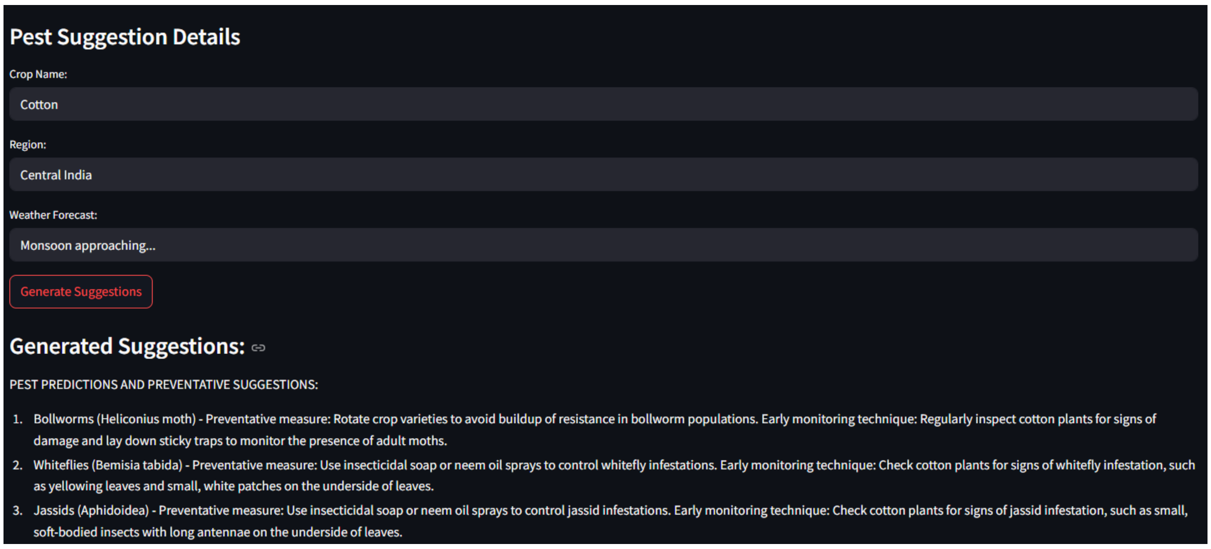
Based on the information provided, it appears that the tomato crop is currently in the flowering stage. However, the farmer has observed some yellowing leaves. This could be indicative of several potential issues, including nutrient deficiencies, pest infestations, or disease. It is recommended that the farmer conduct a more thorough inspection of the plants to determine the cause of the yellowing leaves and take appropriate action to address any underlying issues. Additionally, the farmer should closely monitor the weather conditions and ensure that the tomatoes are receiving adequate water and nutrients to support healthy growth.

Figure 7 - Crop report generation output


what medicine can treat common cough and cold the best?

The provided context does not contain any information about treating a common cough and cold. It provides advice on controlling aphids in plants with insecticides, including insecticidal soaps and oils.

Figure 8 - Rejecting out of context questions


Pest Suggestion Details

Crop Name: Cotton

Region: Central India

Weather Forecast: Monsoon approaching...

Generated Suggestions:

PEST PREDICTIONS AND PREVENTATIVE SUGGESTIONS:

1. Bollworms (Heliconius moth) - Preventative measure: Rotate crop varieties to avoid buildup of resistance in bollworm populations. Early monitoring technique: Regularly inspect cotton plants for signs of damage and lay down sticky traps to monitor the presence of adult moths.
2. Whiteflies (Bemisia tabida) - Preventative measure: Use insecticidal soap or neem oil sprays to control whitefly infestations. Early monitoring technique: Check cotton plants for signs of whitefly infestation, such as yellowing leaves and small, white patches on the underside of leaves.
3. Jassids (Aphidoidea) - Preventative measure: Use insecticidal soap or neem oil sprays to control jassid infestations. Early monitoring technique: Check cotton plants for signs of jassid infestation, such as small, soft-bodied insects with long antennae on the underside of leaves.

Figure 9 - Pest control report generation

5.2 Performance Considerations

The responsiveness of the system, an essential consideration for user experience, depended mainly on the performance of the Ollama server and on the local hardware (NVIDIA RTX 3050 6GB Laptop GPU, 16GB RAM). With GPU acceleration enabled through Ollama, response generation for RAG queries or prompt-crafted outputs took between 5 and 20 seconds, depending on the query complexity and response length. Initial loading and caching of models (LLM, embeddings, translator) and the one-time ingestion of PDF data into the ChromaDB vector store added some initial delay, but subsequent app starts were thereafter greatly accelerated for these components. Sentence-by-sentence translation imposed a significant but generally acceptable response-time overhead when enabled.

5.3 Qualitative Evaluation

Qualitatively evaluating the system shows a number of strengths and weaknesses. The RAG pipeline with the input of appropriate documents greatly anchored the responses of the LLM such that they became more context-driven and factual compared with the base model's outcomes solely based on its parametric information. The facility of tracing the answers back to the original documents greatly increased the sense of reliability. The query-based interactions in the chatbot interface remained straightforward. The prompt engineering components effectively directed the LLM towards creating outputs in desired structures, but the creativity and specificity of these outputs depended both on the LLM's capabilities and the prompt quality.

The system's awareness is nonetheless strictly limited to the PDFs they've ingested; queries beyond this realm generally prompted the LLM to respond that it lacked the information (as directed by the RAG prompt) or in some instances for the direct prompt engineering, producing less pertinent output if the prompt wasn't properly constrained. The quality of the RAG search and resultant answer generation was additionally found to be highly susceptible to the formulation of the user query and the coherence of information across the source papers.

CHALLENGES FACED AND SOLUTIONS

The journey of developing the Agri AI Assistant involved a series of key technical challenges, the most of which pertained to the creation of a stable local GPU-enabled LLM environment on the Windows platform, and optimizing the output quality of the system.

A substantial initial hurdle involved the attempt to directly compile and utilize llama-cpp-python and ctransformers libraries with CUDA support. This process was fraught with difficulties, including persistent CMake errors, failures in CUDA toolset detection by the build system, and conflicts arising from interactions between the Conda Python environment, system-installed Visual Studio Build Tools, and the NVIDIA CUDA Toolkit. Numerous troubleshooting steps were undertaken, such as meticulously setting environment variables (CMAKE_ARGS, CUDAToolkit_ROOT, FORCE_CMAKE), experimenting with different Developer Command Prompts (x86 vs. x64), ensuring the global CUDA Toolkit was correctly installed, and attempting to install specific CUDA compiler components (cuda-nvcc) within the Conda environment. Despite these efforts, achieving a stable and consistently working GPU-accelerated build proved elusive, often resulting in the libraries defaulting to CPU-only operation, which was unacceptably slow. The primary reason for these difficulties appeared to be the intricate and often fragile nature of the C++/CUDA compilation toolchain on Windows, especially when interfacing with Python environments.

This enduring challenge prompted a crucial design choice: the transition to using Ollama as the backend for hosting the LLM. The architecture of Ollama, which packages the required LLM execution and CUDA dependencies together, decouples the complexity of local compilation. This transition greatly eased the configuration for GPU acceleration and enabled the project to continue with a stable and performant local LLM endpoint.

After the LLM serving had stabilized, output quality issues were handled. The RAG system, as helpful as it was, at times pulled out less pertinent document chunks if user questions were unclearly worded or if the exact information wasn't represented in the most optimal way within the source PDFs. Adjusting the RAG prompt to make it specifically command the LLM to draw inferences solely from the given context and to respond with "I don't know" if the answer wasn't there helped alleviate responses that lacked grounding.

The prompt generation module produced excessively generic or even irrelevant responses at the beginning (for instance, writing code). This was addressed by greatly improving the prompt templates. This included defining the persona of the LLM explicitly (for example, "expert in agriculture"), stating the task explicitly, and most importantly introducing negative constraints (for instance, "Don't write code," "Don't ask a question") that would direct the LLM more accurately. The Hindi translation facility used to struggle with long text passages and translate the first sentence alone. This was recognized as a limitation of the translation model's input/output length per pass per sentence. The solution lay in changing the translate_text function to perform sentence-by-sentence translation: the English text needs to get segmented, each of these segments get translated separately, and finally get reassembled. Last but not least, handling of Python package dependencies and infrequent versioning issues (like with the packaging library used by Streamlit and LangChain) involved a close consideration of the requirements.txt file as well as the occasional specific version pinning and up/downgrade actions within the Conda environment.

CONCLUSION AND FUTURE SCOPE

7.1. Conclusion

This project successfully developed a proof-of-concept AI assistant for farmers, leveraging the capabilities of local Large Language Models served via Ollama, Retrieval-Augmented Generation, and prompt engineering. The system demonstrates the feasibility of creating accessible AI tools that can provide contextual agricultural advice and generate relevant content. The core objectives of understanding natural language queries, providing grounded advice through RAG from PDF documents, and generating simulated reports were achieved. The transition to Ollama was a critical step in overcoming initial GPU acceleration challenges, enabling a functional and performant local LLM setup. The chatbot interface provides a user-friendly means of interaction, and the optional Hindi translation feature expands its potential accessibility. Despite the limitations inherent in using a 7B parameter model and a curated document set, the project establishes a solid foundation for a valuable agricultural advisory tool.

7.2. Future Work

The Agri AI Assistant offers numerous avenues for future development and enhancement. A primary focus would be to expand and diversify the knowledge base by ingesting a much larger corpus of agricultural documents, covering a wider range of crops, regions, farming practices, and updated research findings. This would significantly improve the breadth and depth of advice the RAG system can offer.

Implementing more sophisticated RAG techniques could further enhance answer quality. This could include incorporating re-ranking models to better score retrieved documents, using query transformation techniques to rephrase user queries for optimal retrieval, or exploring hybrid search methods.

Expanding multilingual support beyond Hindi translation for outputs to include understanding queries in various regional languages would greatly increase the assistant's utility for a broader farming community. This might involve integrating more advanced multilingual embedding models or input translation layers.

Exploring the possibility of fine-tuning a smaller, agriculture-specific LLM (if suitable base models and datasets become available and accessible for local fine-tuning) could potentially improve domain-specific knowledge and reduce reliance on very large general-purpose models for certain tasks.

Finally, incorporating a user feedback mechanism within the application would be invaluable for iteratively improving the system's responses and identifying areas where the knowledge base or LLM performance needs enhancement. The deferred vision model integration for image-based disease identification remains a significant area for future development, which would require sourcing or training a suitable lightweight vision model and robustly integrating its output with the LLM for treatment planning.

REFERENCES

[1] J. Saini and R. Bhatt, “Global Warming -Causes, Impacts and Mitigation Strategies in Agriculture,” *Curr. J. Appl. Sci. Technol.*, vol. 39, pp. 93–107, Apr. 2020, doi: 10.9734/CJAST/2020/v39i730580.

[2] Aakrity, Aishwarya, H. Jazmyne, and R. Rucker, “Data Driven Organization Problem Solving in Information Management,” 2025, *Unpublished*. doi: 10.13140/RG.2.2.33877.74728.

[3] R. Biswas and N. Goel, “Intelligent Chatbot Assistant in Agriculture Domain,” in *Agriculture-Centric Computation*, M. K. Saini, N. Goel, H. S. Shekhawat, J. L. Mauri, and D. Singh, Eds., Cham: Springer Nature Switzerland, 2023, pp. 180–194. doi: 10.1007/978-3-031-43605-5_14.

[4] “what is the reason why you even want to run llm locally? : r/LocalLLaMA.” Accessed: May 14, 2025. [Online]. Available: https://www.reddit.com/r/LocalLLaMA/comments/18ra7w8/what_is_the_reason_why_you_even_want_to_run_llm/

[5] S. Zhao, Y. Yang, Z. Wang, Z. He, L. K. Qiu, and L. Qiu, “Retrieval Augmented Generation (RAG) and Beyond: A Comprehensive Survey on How to Make your LLMs use External Data More Wisely,” Sep. 23, 2024, *arXiv*: arXiv:2409.14924. doi: 10.48550/arXiv.2409.14924.

[6] “Why Ollama is Good for Running LLMs on Computer.” Accessed: May 14, 2025. [Online]. Available: https://analyticsindiamag.com/deep-tech/why-ollama-is-good-for-running-llms-on-computer/

[7] “Mistral 7B LLM: Run Locally with Ollama | by Parmar shyamsinh | Medium.” Accessed: May 14, 2025. [Online]. Available: https://medium.com/@parmarshyamsinh/mistral-7b-llm-run-locally-with-ollama-bf10494be857