Internship Final Report

Development of Conversational AI Systems Following an 8-Week Structured Plan

Submitted by: Ramchandra Satyawan Rane (AI/ML Intern)

Date: July 25, 2025

Company: **Credenca Data Solutions Private Limited**

Table of Contents

1. Introduction

1.1. Purpose of the Report

This document serves as the final report for the 8-week internship program focused on the development of AI-powered applications. Its purpose is to detail the activities undertaken, the skills acquired, and the deliverables produced, strictly following the structured training and work plan provided by my industry mentor. This report demonstrates the successful translation of the planned training modules and assignments into tangible, functional software projects.

1.2. Internship Overview and Adherence to Plan

The internship was structured as an intensive 8-week program designed to build a comprehensive skill set in conversational AI, from foundational concepts to advanced systems. The plan systematically guided my learning journey, beginning with core LangChain training, progressing through a series of Proof-of-Concept (POC) assignments, and culminating in system enhancement, testing, and final documentation.

This report is structured chronologically, with each section corresponding to a specific week's focus area and expected deliverables as outlined in the training plan. It provides a clear narrative of how the theoretical knowledge gained in the initial weeks was progressively applied to build, enhance, and validate a suite of sophisticated chatbot applications. The projects completed, including a basic chatbot, a RAG-enabled website assistant, and two specialized assistants for tabular data and vision-language understanding, directly reflect the successful completion of the assignments for Weeks 3, 4, and 5. The subsequent weeks were dedicated to implementing the planned enhancements for robustness and performance, followed by rigorous testing and final reporting, ensuring full adherence to the mentor's plan.

2. Week 1-2: Foundational Training in LangChain and LLM Applications

2.1. Objectives and Training Modules

As per the training plan, the first two weeks were dedicated to intensive, foundational learning. The primary objective was to master the core concepts of building LLM-powered applications using Python and the LangChain framework. This involved completing several training modules, including "LangChain MasterClass" and "Master LangChain Build #15 AI Apps," which covered the entire ecosystem for interacting with models like OpenAI's GPT series.

2.2. Key Concepts Mastered

The training provided a strong theoretical and practical grounding in the essential components required for building conversational AI:

- LLM Wrappers: Understanding how LangChain provides a standardized interface to interact with different Large Language Models, abstracting away the complexity of direct API calls.
- Prompt Engineering: Learning to use PromptTemplates and ChatPromptTemplates to create dynamic, reusable, and context-rich prompts. This is fundamental to guiding the LLM's behaviour effectively.
- Chains: Grasping the concept of Chains as sequences of operations. I learned to use simple chains like LLMChain for direct model interaction and laid the groundwork for more complex chains to be used in later weeks.
- Memory: Understanding the stateless nature of LLMs and the importance of Memory components. I explored different strategies like ConversationBufferMemory to enable chatbots to remember past interactions and maintain conversational context.
- Vector Databases: The training introduced the concept of vector databases for semantic search, specifically covering Pinecone. This was a critical prerequisite for the RAG assignment in Week 4.

2.3. Initial Technical Exploration

To solidify this learning, I began by building small-scale applications. The first step was creating a script to test OpenAI API connectivity, ensuring the development environment was correctly configured. This initial exploration involved writing basic Python scripts to send prompts to the GPT-3.5 model via LangChain and print the responses, confirming a successful setup and a foundational understanding of the core request-response loop. This practical step ensured that the theoretical knowledge from the training modules was immediately translated into working code, paving the way for the POC assignment in Week 3.

3. Week 3: Mini POC - Basic Chatbot with LangChain and OpenAI

3.1. Deliverable: Working Chatbot Interface

The primary deliverable for Week 3 was a "Working chatbot interface with documentation of model choice and LangChain usage." I successfully met this objective by developing the gpt_chat_memory_app and chatgpt_role_assistant projects. These applications provided a functional user interface built with Streamlit and demonstrated the core principles of conversational AI.

3.2. Technical Implementation

- Model Choice: I selected OpenAI's gpt-3.5-turbo model for this POC due to its strong balance of performance and ease of integration via LangChain's ChatOpenAI wrapper. Its conversational fine-tuning made it ideal for a chatbot application.
- LangChain Usage:
    - ConversationChain: This was the central component used to manage the dialogue. It seamlessly integrates the LLM with a memory module.
    - ConversationBufferMemory: To fulfill the requirement of a stateful conversation, I used this memory type to store the entire chat history. This allowed the chatbot to refer to earlier parts of the conversation, providing contextually relevant responses.
    - Streamlit UI: I used the Streamlit library to rapidly build an interactive web interface. Key components included st.text_input for user queries, st.button to submit, and a container to display the alternating user and AI messages, creating an intuitive chat experience.

3.3. Outcome and Learnings

The outcome was a fully functional chatbot that not only responded to user queries but also maintained the context of the conversation. The chatgpt_role_assistant project further extended this by allowing the user to define a persona for the chatbot (e.g., "Legal Expert") using SystemMessage prompts, demonstrating an early grasp of prompt engineering.

This week was crucial for transitioning from theory to practice. I learned how to manage application state in Streamlit using st.session_state, how to structure a LangChain application with Chains and Memory, and how to build a user-friendly interface. This successfully completed the mini-POC deliverable and set a strong foundation for the more complex RAG assignment in the following week.

4. Week 4: Expansion to Retrieval-Augmented Generation (RAG)

4.1. Deliverable: Chatbot Integrated with Document Retrieval

The objective for Week 4 was to expand the basic chatbot to support Retrieval-Augmented Generation (RAG), integrating it with document retrieval capabilities using Pinecone. I achieved this by developing the website_ai_assistant project, which functioned as a chatbot capable of answering questions based on the content of a specific website.

4.2. Architectural Deep Dive: RAG with Pinecone

This project required the implementation of a full end-to-end RAG pipeline, a significant step up in complexity. The architecture was as follows:

1. Data Ingestion and Processing:
    - A SitemapLoader was used to automatically discover and scrape all pages of a target website.
    - The scraped HTML content was parsed, and the raw text was split into smaller, overlapping chunks using RecursiveCharacterTextSplitter. This is essential for fitting the content within the LLM's context window and ensuring no semantic meaning is lost at the boundaries of chunks.
2. Vectorization and Indexing:
    - Each text chunk was converted into a numerical vector using an embedding model from HuggingFace (all-MiniLM-L6-v2). These embeddings capture the semantic meaning of the text.
    - These vectors, along with their corresponding text metadata, were then uploaded to a Pinecone index. Pinecone, as a managed vector database, provides the infrastructure for performing highly efficient similarity searches at scale.
3. Retrieval and Generation:
    - When a user asks a question, the question itself is first converted into a vector using the same embedding model.
    - This query vector is used to search the Pinecone index, which returns the 'k' most semantically similar text chunks from the original website content.
    - These retrieved chunks are then dynamically injected into a prompt and sent to the LLM, which generates a final answer based on this provided, factual context.

4.3. Project Outcome: The Website AI Assistant

The result was a powerful information retrieval system. Instead of answering from its generic training data, the chatbot's knowledge was strictly limited to the content of the provided website, making its answers accurate and contextually relevant. This project successfully demonstrated a practical application of RAG, fulfilling the week's deliverable and showcasing the ability to ground LLM responses in an external, custom knowledge base.

5. Week 5: POCs - Specialized AI Assistants for Data and Vision

5.1. Deliverable: AI Assistants with File Input and Query Capability

Following the plan, Week 5's goal was to prototype an AI assistant for knowledge base Q&A. I accomplished this by developing two distinct, specialized Proof-of-Concept applications, each tailored to a different data modality. This approach demonstrated a deeper understanding of building fit-for-purpose AI tools.

5.2. The Tabular Data Analyst (CSV/Excel)

This application was designed to allow users to have a natural language conversation with their spreadsheets.

- Functionality: Users can upload a CSV or Excel file. The application then uses a LangChain Agent to answer questions about the data.
- Technical Implementation: The core of this bot is the create_pandas_dataframe_agent. This agent is provided with the user's DataFrame and an LLM (GPT-3.5). When a user asks a question like, "What is the average sales for the North region?", the agent's LLM component generates and executes Python code using the Pandas library (e.g., df[df['Region'] == 'North']['Sales'].mean()) to find the answer directly from the data. This is a powerful demonstration of an agentic workflow where the AI uses tools to accomplish a task.

5.3. The Vision-Language Document Assistant (PDF/DOCX)

- Functionality: Users can upload a document (PDF, DOCX, PPTX). The assistant extracts all text and analyses all images to answer questions about the entire document.
- Technical Implementation: This bot uses a sophisticated multi-model architecture:
    1. Image Understanding with Gemini: The application first extracts all images from the document. Each image is then sent to Google's Gemini 1.5 Flash model, a state-of-the-art vision-language model, which generates a rich, detailed text description of the image's content.
    2. Context Combination: The original text from the document is combined with the Gemini-generated image descriptions into a single, comprehensive knowledge base.
    3. RAG with OpenAI: This combined text is then chunked, vectorized using OpenAI Embeddings, and stored in a FAISS vector store. When the user asks a question, the system uses this vector store to retrieve the most relevant text or image descriptions and provides them to the OpenAI GPT-3.5 model to generate a final, contextually-aware answer.

This dual-bot approach fulfilled the week's deliverable by creating powerful, specialized assistants and demonstrated the advanced skill of integrating multiple leading AI models (Gemini and OpenAI) to solve a complex problem.

6. Week 6: System Enhancement and Optimization

6.1. Deliverable: Improved Performance, Logging, and Retry Mechanisms

The focus for Week 6, as per the plan, was to move beyond core functionality and enhance the system's robustness and performance. The goal was to implement features that would make the application more reliable and efficient in a real-world scenario. This was applied to the advanced chatbot prototype from the previous week.

6.2. Implementation of Smart Memory

While the initial chatbot used ConversationBufferMemory, which stores the entire chat history, this can be inefficient and costly for long conversations due to high token usage. To optimize this, I explored and understood how to implement smarter memory strategies:

- ConversationSummaryMemory: This strategy uses the LLM itself to create a running summary of the conversation. Instead of passing the full chat history, only the concise summary is sent, significantly reducing the number of tokens required for each API call while still preserving the core context of the dialogue.
- ConversationBufferWindowMemory: This provides a balance by only keeping the last 'k' interactions in memory. This is effective for conversations where recent context is more important than very old messages.

Implementing these memory types demonstrated an understanding of performance optimization and cost management in LLM applications.

6.3. Implementation of Caching and Robust Error Handling

- Caching with @st.cache_data: To improve the perceived performance of the Streamlit application, I implemented caching. The function responsible for generating the LLM response was decorated with @st.cache_data. This means that if the same input is received again, Streamlit serves the cached result instead of making another expensive API call, making the application feel much faster and reducing redundant API usage.
- Logging: A logging mechanism was set up using Python's built-in logging module. All user inputs, model responses, and critical errors were logged to a file (chatbot_log.txt). This is an essential practice for debugging, monitoring application behavior, and understanding user interaction patterns.
- Retry Mechanism: Network calls to external APIs like OpenAI can sometimes fail due to transient issues. To handle this gracefully, I created a robust_invoke wrapper function. This function wraps the LLM call in a try...except block and implements an exponential backoff retry loop. If an API call fails, it waits for a short period and tries again, attempting this up to three times before returning a user-friendly error message. This makes the application significantly more resilient to temporary network or API issues.

7. Week 7: Testing and Validation

7.1. Deliverable: Test Cases and Performance Documentation

With an enhanced and robust application in place, Week 7 was dedicated to systematic testing and validation, as required by the training plan. The objective was to "conduct testing, simulate team usage, and document errors/failures." This was achieved by creating a formal unit testing suite.

7.2. Unit Testing Strategy with Pytest

I created a dedicated test file, Testapp.py, to house a suite of unit tests for the advanced chatbot's core logic. The key challenge was testing the application code, which relies on Streamlit's runtime and external APIs, in a standalone Python environment.

- Framework: I used pytest, a standard and powerful testing framework for Python.
- Mocking Dependencies: To isolate the code under test, I employed extensive mocking:
    - monkeypatch: This pytest fixture was used to replace external dependencies at runtime. For instance, time.sleep() was patched to do nothing, allowing retry logic tests to run instantly. Most importantly, the ChatOpenAI class was patched to return a DummyLLM object instead of making real API calls.
    - DummyLLM Class: I created a mock LLM class that could be programmed to return predictable responses or raise specific exceptions on demand. This was crucial for testing the robust_invoke function's behavior under various success and failure conditions.
    - Faking Session State: A helper function was written to create a mock version of Streamlit's st.session_state, allowing the functions that depend on it to be tested without errors.

7.3. Simulating Usage and Documenting Failures

The test suite was designed to simulate different user interactions and potential failure modes:

- Testing the Retry Logic: I wrote specific tests for the robust_invoke function to ensure it succeeded immediately, succeeded after one failure, and returned a proper error message after all retries were exhausted.
- Testing the Dual-Path Logic: I created separate tests for the application's main response-generating function. One test simulated a user uploading a CSV file, verifying that the pandas agent logic was triggered correctly. Another test simulated a document upload, confirming that the ConversationalRetrievalChain was invoked as expected.

This suite of automated tests serves as living documentation of the application's expected behavior and provides a safety net to catch regressions as new features are added, fulfilling the week's deliverable for rigorous testing.

8. Week 8: Finalization and Knowledge Transfer

8.1. Deliverable: Final Report, Presentation, and Documentation

The final week of the internship was dedicated to consolidating all the work done, documenting the projects, and preparing for knowledge transfer, in line with the plan's final deliverable.

- Final Report: This document serves as the primary written deliverable. It has been carefully structured to align with the 8-week training plan, detailing the objectives, activities, and outcomes for each week.
- Presentation: A concise 10-slide presentation was created to visually summarize the internship journey. It highlights the key concepts learned, the progression of projects from a simple chatbot to an advanced RAG system, and the final architecture of the capstone project.
- Technical Documentation: The code produced throughout the internship is hosted in a GitHub repository. The code itself is well-commented, and the repository includes a README.md file that explains the purpose of each project and how to run the applications. The unit test suite (Testapp.py) also serves as technical documentation for the application's core logic.

8.2. Summary of Achievements

This 8-week internship was a highly successful and intensive learning experience. I successfully progressed from a student of AI theory to a practical builder of sophisticated conversational AI systems. By adhering to the structured plan, I systematically acquired skills in LangChain, RAG architecture, vector databases. The final POCs—a tabular data analyst and a vision-language document assistant—stand as a testament to the comprehensive knowledge gained, demonstrating the ability to build specialized tools and integrate multiple state-of-the-art models like Google's Gemini and OpenAI's GPT.

8.3. Future Recommendations

The projects developed serve as excellent prototypes with significant potential for future expansion. Key recommendations include:

1. Cloud Deployment: Deploy the advanced chatbot application to a cloud service like Streamlit Community Cloud or AWS to make it a shareable, live tool.
2. Model Upgrade: Enhance the application's reasoning capabilities by integrating more powerful models like GPT-4 and adding a UI selector for the user to choose.
3. Expand Agent Capabilities: For the tabular data agent, provide it with more tools, such as the ability to generate plots and charts using libraries like altair, which is already imported in the project.

This concludes the work completed during the internship, successfully meeting all deliverables outlined in the training plan.