AURA: Augmented Utilities and Response Agent

Kuldeep, Saksham, Udit, Vinay, Yash

November 20, 2025

Abstract

Employees struggle to find information in large document repositories, a problem not solved by standard LLMs due to their upload and context size limits. We introduce AURA (Augmented Utilities and Response Agent), an intelligent document assistant. AURA is a conversational AI that uses a sophisticated pipeline, including RAG and MCP, to process company data (text, images) into an interactive knowledge base. It also retrieves real-time information from the internet (websites, YouTube, etc.) for comprehensive responses.

1 Introduction

Organizations lose productivity manually searching vast document collections, a challenge standard LLMs cannot address due to their size and context limits. AURA (Augmented Utilities and Response Agent) is an intelligent, conversational AI document assistant that uses RAG and MCP to transform company data (manuals, PDFs) and real-time internet sources into a comprehensive, interactive knowledge base.

2 AI Pipeline and Methodology

The core of AURA is its sophisticated AI pipeline, which is designed to handle a variety of tasks, from query understanding to multimodal information retrieval and final response generation.

2.1 Pipeline Overview

The pipeline begins when a user submits a request. The input is parsed, and if it contains images, a vision model generates a description. The user's query is rewritten for clarity, and the conversation history is summarized to maintain context. Based on the user's selection, the pipeline then proceeds through either the Model Context Protocol (MCP) or the Retrieval-Augmented Generation (RAG) process, or both.

2.2 Retrieval-Augmented Generation (RAG)

Our RAG implementation goes beyond simple text retrieval and employs an advanced multimodal approach.

- **Document Parsing with MinerU:** Unstructured documents are parsed using MinerU to identify and extract titles, sections, tables, figures, and content flow, converting complex PDFs into machine-readable formats.
- **Dual Graph Construction:** We implement a dual graph approach to represent the knowledge within the documents.
  1. **Cross-Modal Knowledge Graph:** Non-text elements like images and tables become parent nodes, with entities from their captions as child nodes.
  2. **Text-Based Knowledge Graph:** Built from textual entities and their relations.

  These two graphs are fused to create a unified knowledge graph that connects all modalities into a single semantic map.
- **Implementation Details and Model Selection:** The RAG pipeline balances performance with computational efficiency by using the local bge-large-en-v1.5 model for semantic embeddings.

While initial experiments with local generative models like Mistral-7B and Qwen3-VL encountered hardware memory constraints, the final architecture leverages AWS Bedrock. We currently deploy Claude 3.0 Sonnet in a serverless environment, ensuring robust scalability for both text generation and vision-language tasks.

2.3 Model Context Protocol (MCP) Integration

The Model Context Protocol (MCP) allows AURA to access and process real-time information from the internet. Unlike typical setups that depend on third-party subscriptions for MCP services, we have created a custom server and client framework using LangChain. This implementation offers Google Search capabilities, retrieves YouTube sources, generates relevant Amazon product links and YouTube Videos summarisation.

The workflow operates as follows:

- **Query Refinement and Tool Selection:** The Large Language Model (LLM) refines the user's input. Based on the refined query, the LLM outputs a decision in JSON format to identify the specific tools needed (e.g., YouTube search, Amazon search). However, if MCP is globally enabled, a Google search is automatically performed on all rewritten queries. The Google API are done through the Serper API since they provide free credits, and YouTube API calls were made with the official YouTube API that Google provides.
- **Search and Categorization:** Results from the Google Search are sorted into social media, forums, blogs, and video platforms. Currently, the system prioritizes blog posts and forums for content extraction. We use the `Trafilatura` Python library to scrape text from these sources, while filtering out social media and complex video sites to ensure reliability.
- **Video Integration:** If the LLM finds that visual content can be provided, the system makes a YouTube API call to get a curated list of videos relevant to the query.
- **Product Recommendations:** With context from web content and video descriptions, the LLM assesses whether physical products are relevant. If they are, Amazon links are generated and added to the final response.
- **Synthesised Output:** The user receives a detailed response that combines scraped web content, citation links, relevant YouTube videos, and Amazon product suggestions.
- **YouTube Summarisation :** After generating the output, an optional feature allows for summarising the retrieved YouTube content. The system fetches transcripts through `youtube-transcript-api` for the identified videos and uses the LLM for recursive summarisation, giving the user a brief overview of the video material.

2.4 AI Models

We utilize a suite of AI models, each chosen for a specific purpose in the pipeline:

- **Qwen2.5-1.5B:** Used for query rewriting, history summarization, and chat title generation. Hosted on AWS EC2.
- **Qwen2.5-VL-3B:** Handles image description generation and the creation of specific prompts for the MCP and RAG processes. Hosted on AWS EC2.
- **BGe Large V1.5 300M:** A text embedding model for retrieval purposes.
- **Claude 3 Sonnet:** The final model for generating the polished answer. It is hosted on AWS Bedrock for a serverless and scalable deployment.
- **Inference Engine (Gemma 3 4B):** The query rewriting and summarization tasks for the MCP use the Gemma 3 4B model. This model runs on Ollama in an AWS EC2 instance to ensure efficient processing.

3 Implementation and Tech Stack

The development of AURA leveraged a variety of modern technologies and frameworks to build a comprehensive and efficient system.

- **Backend Framework:** Django
- **Frontend/UI:** Streamlit
- **Database:** SQLite
- **AI Microservice:** FastAPI
- **Retrieval-Augmented Generation (RAG):** RAG-Anything, MinerU
- **Model Context Protocol (MCP):** `Serper API`, `Langchain`, `Youtube API`, `Trafilatura`, `youtube-transcript-api`
- **Model Downloads:** Ollama, Hugging Face
- **Cloud Hosting:** AWS (EC2 and Bedrock)

Initially, we explored using React and JavaScript for the frontend and backend. However, we encountered integration challenges with our Python-based AI pipeline, which led us to adopt Streamlit and Django in the final phase of the project for a more seamless integration.

4 Deployment

4.1 Backend Framework

The backend is built using a combination of Django and FastAPI.

- **Django:** Serves as the primary backend framework, handling robust server-side logic, API endpoints, user authentication, and database interactions.
- **FastAPI:** Used as an external microservice to handle the computationally intensive AI pipeline, ensuring that the main application remains responsive.

4.2 Database

We utilize SQLite as our database for its lightweight and embedded nature, making it ideal for rapid development and deployment. The database schema consists of five core tables: User, Chat, Message, Attachment, and PipelineExecution, which store user data, conversation history, and AI pipeline logs.

4.3 High-Level Flow

The user interaction with the system follows a clear and efficient flow:

1. The user sends query, from the frontend the query triggers an API call to the database that sends the conversation history. Both of these are done by Django backend.
2. The combined along with attached images are preprocessed to form a json type structure in which we have user query, converstation history and list of images in base64 strings and option buttons for (MCP, RAG, Youtube summary). Our assumption in this is that the order of images does not matter.
3. The orchestrator is hosted as a port, to which we send this structure and recieve the final answer along with all the possible intermediate information like rewritten prompt, summarised conversation etc for storage in database.
4. What happens at orchestrator port:
    - The query is rewritten to be concise and straightforward. If this is the user's first query, we use title generation on this prompt for generating the chat title.
    - The history is summarised to capture the main points which reducing the size.

- The images are described using a VL model
- All these are combined to form the complete prompt
- If none of them is selected:
  - The Complete prompt along with images is send to claude 3 sonnet for answering.
- If one of MCP or RAG is selected:
  - We use the complete prompt to get the setting selected prompt using rewrite *mcp/rag* prompt function. This prompt is then sent to the selected tool.
  - Then the recieved answer along with complete prompt and images is sent to claude 3 model for making it concise and better.
- If both of the settings are selected: We rewrite the prompt for MCP and RAG and then those prompts are sent to the tools and we get 2 answer from both the tool then we send these 2 answers along with complete prompt and image to claude 3 for combining those answers.
- If the Youtube summary option is selected and the MCP option is not selected, we perform mcp search but only use the youtube links and parse them to make their summary using subtitles. Anything apart from the youtube summary is not used further.

4.4 Frontend/UI

The user interface is developed using Streamlit, which allows for the rapid creation of interactive web applications. The frontend provides features such as:

- Image attachment with drag-and-drop support.
- Automatic chat naming based on the user's first prompt(using Qwen 2.5).
- A voice recorder that converts speech prompt to text prompt.
- User sign-in and sign-up functionality.
- A sidebar to display chat history and user credentials.
- Options to rename and delete chats.

5 Team Contributions

The successful development of AURA was a collaborative effort, with each team member taking responsibility for key components of the project:

- **Kuldeep:** JS Backend & React Frontend, Data Collection (Manual)
- **Saksham:** Pipeline Orchestration, LLM Experiments
- **Udit:** Backend Development, Database Integration
- **Vinay:** Full RAG Pipeline, Orchestration
- **Yash:** MCP, Debugging

6 Conclusion

AURA successfully addresses the critical need for an efficient and intelligent document interaction system. By combining a sophisticated AI pipeline with a user-friendly interface, we have created a powerful tool that can significantly enhance productivity in organizations dealing with large volumes of documents. The modular architecture, leveraging both internal knowledge processing through RAG and external information retrieval via MCP, makes AURA a versatile and comprehensive solution. Future work could focus on expanding the range of supported document types, further refining the retrieval and generation models, and enhancing the user experience with more advanced conversational capabilities.

Attached Demo Video

References

[1] Rag-Anything Github Repo: https://github.com/HKUDS/RAG-Anything .

[2] MinerU Github Repo: https://github.com/opendatalab/MinerU .

[3] LightRag Github Repo: https://github.com/HKUDS/LightRAG .

[4] Internet Archive. Available at: https://archive.org/ (Accessed for downloading car manuals data).

[5] My Car User Manual. Available at: https://www.mycarusermanual.com/ (Accessed for data).

[6] Toyota Owners Manual Portal. Available at: https://www.toyota.com/owners/warranty-owners-manuals/ (Accessed for data).

[7] Ford Support. Available at: https://www.ford.com/support/ (Accessed for data).

[8] Honda Owners. Available at: https://owners.honda.com/ (Accessed for data).

[9] DEG OEM Links (OEM Manual Directory). Available at: https://degweb.org/owners-manual-usa-vehicles/ (Accessed for data).