Sabudh Passion Project

Learn and Give Back to Society

Final Project Report

Document Analysis using LLMs.

Mishra, Prashant; Pal, Soumyodip; Sinha, Udita; Kanishka, R; Patidar, Avishkant; Simran, Simran; Mahatar, Devansi; Anurag, Anurag; Raj, Kiruthick

SABUDH

Table of contents

List of Figures

List of Tables

Preface

In today's digital era, the explosion of unstructured information from diverse sources poses a significant challenge in efficiently extracting valuable insights. This is particularly true when handling complex content across formats like documents, databases, and multimedia. Traditional systems fall short when users demand real-time, contextual understanding from such varied input.

InsightPulse addresses this need by integrating Large Language Models (LLMs) [[1]][2], vector-based semantic search [3], Retrieval-Augmented Generation (RAG) [4], and a modular multi-agent architecture into a single platform. Unlike generic AI document assistants, InsightPulse offers fine-grained query understanding across uploaded files, stored data, and video transcripts—thus acting as a comprehensive insight engine.

Our motivation stems from the limitations of current document systems and the rising need for tools that can analyze structured and unstructured data using contextual intelligence. By enabling tasks such as Retrieval-Augmented Generation (RAG) [4], SQL-based analytics, YouTube transcript parsing [5], and intelligent summarization, this project sets a foundation for scalable, extensible, and future-ready document intelligence systems. This project has undergone iterative development, repeated evaluation, and collaborative improvement. It reflects the growing relevance of AI-powered content understanding and stands as a strong foundation for future multimodal AI applications.

Abstract

Despite the rapid progress in language models [[1]][2] and information retrieval [3], creating an intelligent system that can analyze multimodal input—such as documents, structured databases, and video content—remains a complex challenge.

InsightPulse is a unified LLM-powered platform designed to address this gap. It supports document-based RAG [4], where users can upload files like PDFs and text documents and ask contextual questions about their contents. A RAG Agent is also included, which allows users to ask general questions, prompting the system to search across stored documents in the database for relevant answers. For structured data queries, the SQL Agent [6] enables natural language queries such as "List employees with salary > 50K" and returns tabular outputs. YouTube transcript extraction [8] is also supported by simply pasting a video URL. Lastly, the system includes a summarization function to provide concise overviews of retrieved or uploaded content.

With dual LLM support—Gemini [1] (via Google API) and Ollama [2] (for local inference)—InsightPulse combines flexibility and performance, making it suitable for both cloud-native and privacy-sensitive deployments.

Chapter 1

Introduction

With the exponential growth of unstructured data across PDFs, documents, and digital media, there is a pressing need for intelligent platforms capable of understanding, analyzing, and extracting insights beyond simple text parsing.

While many tools provide basic NLP or search capabilities, they often lack flexibility in handling multiple data types or executing task-specific workflows (e.g., database querying, summarization). Existing models typically ignore the user's intent or fail to integrate structured and unstructured sources.

InsightPulse solves this by combining a multi-agent LLM architecture [6] with task-specific tools, vector search via Pinecone [3] for semantic retrieval, document chunking, transcript parsing [5], and modular task routing—including summarization and SQL querying [6]. These features together allow InsightPulse to act as a comprehensive document analysis and querying assistant.

1.1 Overview

InsightPulse provides a smart interface [7] where users can upload documents and ask any question about their content using the document-specific RAG [4] functionality. When users have a question that is not tied to a particular document, the RAG Agent searches across all stored files to find the best semantic match. Users can also query structured databases [[8]][9] using natural language, for example, asking for employee data based on salary thresholds, and receive accurate, tabular responses. Additionally, the system enables YouTube transcript extraction [5], where a user can simply enter a video URL to retrieve its transcript. Smart summarization is also supported for any retrieved content or uploaded documents.

The system uses a dual-LLM architecture. Ollama [2] handles local inference, ideal for offline or privacy-sensitive tasks, while Gemini [1] provides powerful, cloud-based reasoning via Google's API. The architecture leverages LangChain tools [6] such as AgentExecutor, vector retrievers, document loaders, and custom prompt templates to orchestrate these capabilities.

1.1.1 Existing System

Most document tools—such as ChatPDF, AskYourPDF, and Microsoft Copilot—are limited by their support for a single document format like PDFs. These systems also lack multi-agent extensibility and cannot handle structured database queries or multimedia inputs like video transcripts.

InsightPulse advances beyond these limitations by offering a broad range of functionalities, including YouTube transcript ingestion [5], flexible summarization, and semantic search using Pinecone [3]. It routes user queries to the most suitable agents dynamically using multi-agent logic [6], and integrates an SQL Agent [6] for querying structured data. This positions InsightPulse as a hybrid of open-source flexibility and enterprise-level capabilities, comparable to RAG systems in Glean [10] or Google's Vertex AI Search [11].

1.2 Objectives of Project

The key objectives of this project are:

1. Multimodal Input Support: Analyze documents (PDF, TXT, CSV), database queries [[8]][9], and YouTube videos [8]

2. Contextual LLM Responses: Use Gemini [1] and Ollama [2] to generate insights, answers, and summaries

3. Semantic Retrieval: Store document chunks in Pinecone vector DB [3] and retrieve them using similarity search

4. Multi-Agent Routing: Use LangChain's AgentExecutor [6] and create_tool_calling_agent to route user queries to specialized agents

5. SQL Agent Integration: Allow users to perform natural language queries on structured databases [6] with tabular outputs

6. Summarization and QA: Enable summarization and retrieval-augmented question answering [4] using document chains.

Chapter 2

Pre-Processing and Exploratory Data Analysis

Once the dataset is obtained, it undergoes preprocessing to prepare it for document analysis using Large Language Models (LLMs) [[1]][2]. This process includes parsing various document formats, cleaning the extracted text, and chunking the content to enable effective embedding [12] and retrieval. Preprocessing is a critical step, as the system's accuracy relies heavily on the clarity and structure of the input data.

2.1 Dataset Collection

Once the dataset is obtained, it undergoes preprocessing to prepare it for document analysis using Large Language Models (LLMs) [[1]][2]. This process includes parsing various document formats, cleaning the extracted text, and chunking the content to enable effective embedding [12] and retrieval. Preprocessing is a critical step, as the system's accuracy relies heavily on the clarity and structure of the input data.

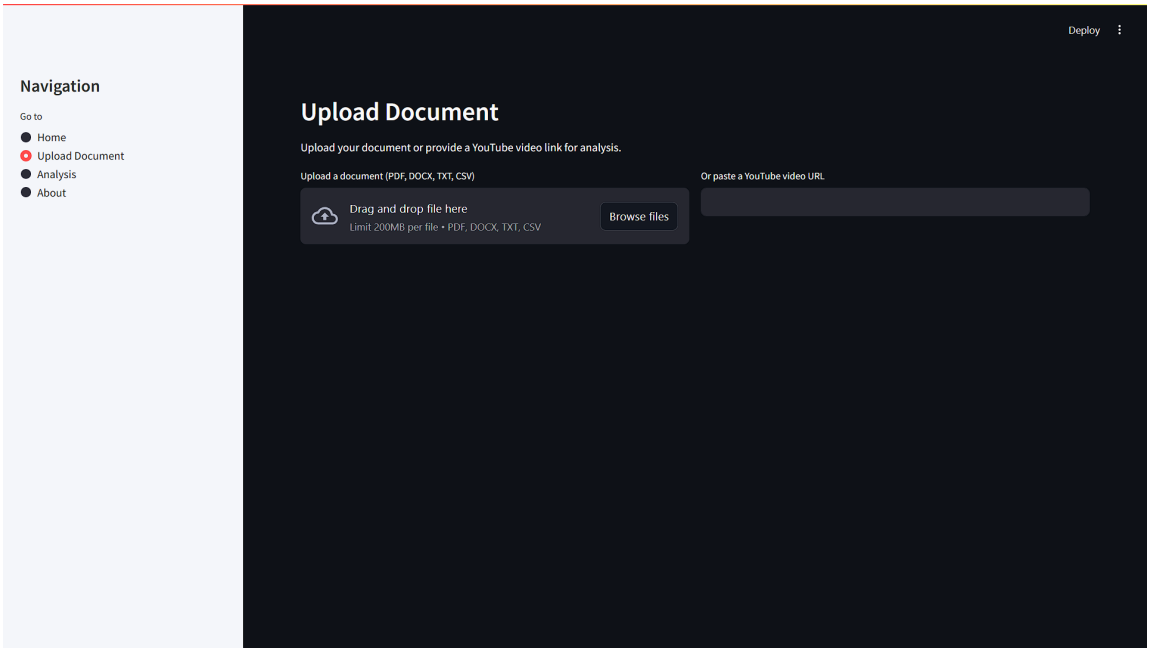

Figure 2.1: Document Loader Architecture for PDF, CSV, TXT, and YouTube Inputs

Inclusion of Structured Data (SQL Agent)

In addition to document-based QA, the system also integrates a SQL Agent [6] capable of handling structured tabular data. This agent is implemented using LangChain's SQLDatabaseChain [6] and operates over:

1. CSV files loaded into a database (e.g., SQLite [8] or MongoDB [9])
2. Structured collections where each row represents a record

Users can input natural language questions, which are translated into SQL queries using an LLM [[1]][2]. The resulting SQL is executed against the structured backend, and the results are optionally passed to a language model for summarization.

Unlike the document loaders, this SQL path bypasses embedding and chunking and is handled separately during runtime query execution.

2.2 Data Pre-processing

Several key considerations are addressed during preprocessing:

1. Handling Missing Values: While traditional missing values do not apply to unstructured text, the system automatically skips files that are empty, malformed, or unparseable. In the case of PDFs, sections such as scanned images or unreadable tables are ignored silently.

2.Managing Large Data Size: To handle large documents efficiently, the system implements a chunking strategy using the RecursiveCharacterTextSplitter [6]. This method divides text into overlapping chunks of 1,000 characters with a 200-character overlap, ensuring contextual continuity and manageable memory usage.

3.Clustering Considerations: Clustering is not utilized within this system. Document retrieval is instead powered by semantic similarity using vector embeddings [12].

4.Feature Scaling: As the system processes unstructured textual data, feature scaling is not applicable.

5.Outlier Management: Extremely short or inconsistent chunks are either skipped or padded. However, no formal outlier detection is implemented.

6.Data Transformation: Basic text cleaning is applied, such as removing excessive whitespace and merging lines. Advanced natural language processing steps like stemming or lemmatization are intentionally omitted to retain semantic richness.

2.3 Exploratory Data Analysis and Visualisations

Although comprehensive data visualization is not integrated, the following exploratory methods are embedded in the development workflow:

1.DataVisualization: While visual charts or graphs are not generated, chunk counts per document are printed via logs to verify successful parsing and chunking.

2.Statistical Analysis: Logs are used to track the number and average length of chunks to ensure a balanced and consistent preprocessing pipeline.

3.Feature Selection: Feature selection is not applicable, as the system leverages full-text chunks and relies entirely on embedding-based similarity [12] for retrieval.

4.Dimensionality Reduction: Dimensionality reduction is inherently performed during embedding [12]. The all-MiniLM-L6-v2 model [12] reduces input text to dense vectors of 384 dimensions. No additional techniques like PCA or t-SNE are applied.

2.4 Other Related Sections (Optional)

1.Logging and Debugging: Logging is heavily utilized to monitor document parsing and chunking operations, helping developers ensure smooth data processing.

2.Unsupervised Learning: No unsupervised learning methods (e.g., clustering, correlation analysis) are applied. The system is designed for retrieval-augmented generation (RAG) [4], which does not require such techniques.

3.Document-Type Considerations: The structure and content type of uploaded documents (e.g., structured PDFs versus conversational transcripts) can influence chunking quality and, subsequently, retrieval accuracy [12]. These factors are acknowledged for future system optimization.

Chapter 3

Methodology

3.1 Introduction to Python for Machine Learning

This system has been built using Python, owing to its rich ecosystem of libraries for language models, vector databases, and data processing. Tools such as LangChain [6], Streamlit [7], Pinecone [3], HuggingFace [13], and YouTubeTranscriptApi [5] were used to implement both unstructured document QA and structured data querying via SQL. Python's object-oriented flexibility allowed seamless construction of modular agents and chains.

3.2 Platform and Machine Configurations Used

Development Platforms:

1.Google Colab: Used for prototyping and embedding generation [12] via HuggingFace [13]

2.Local Machine: Hosted the Ollama LLM server [2] for local/offline LLM inference

3.Streamlit [7]: Deployed as the front-end interface for file uploads, query input, and response display

4.Machine Configuration (for Local Testing):

a.Processor:Intel i5 / Ryzen 5

b.RAM: 16 GB

c.Operating System: Windows 10 / Ubuntu 22.04

d.LLM Server: Ollama [@5] (for models like Mistral or LLaMA2)

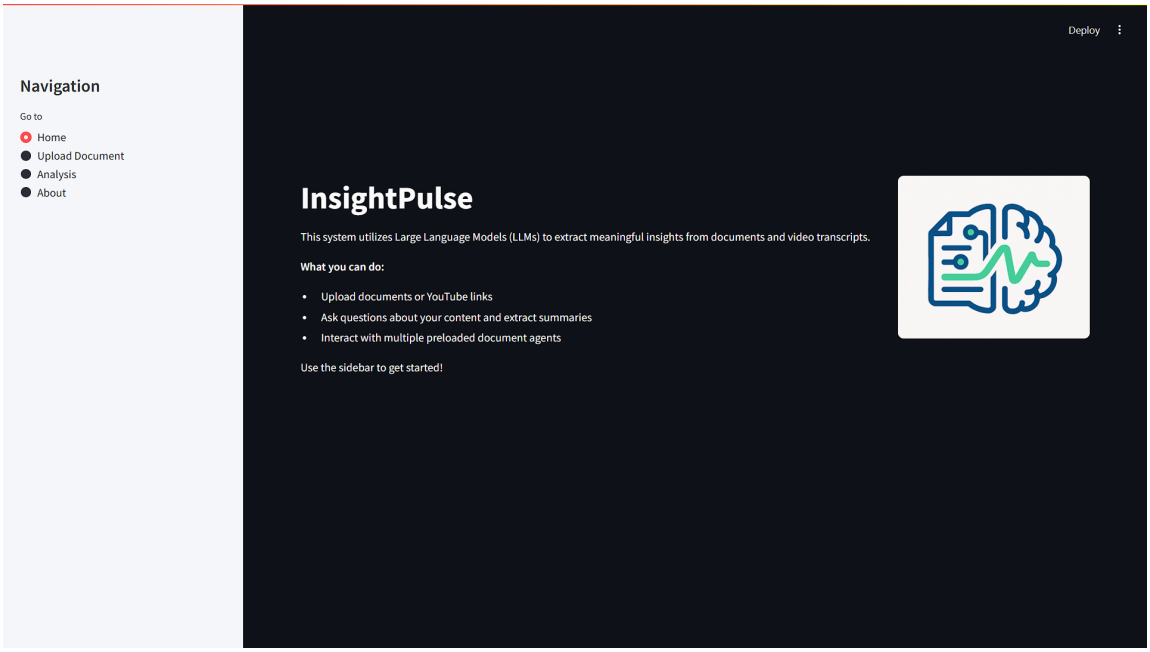

Figure 3.1: HomePage

3.3 Data Split

Since the system uses retrieval-augmented generation (RAG) [4] and SQL query translation [6] — not traditional supervised ML — standard dataset splitting (training/validation/test) is not applicable. However, separation exists at the functional level:

1. **Document Data:** Parsed and chunked for semantic search [[3]][12]
2. **SQL Data:** Stored in a structured relational database [[8]][9]
3. **User Queries:** Dynamically processed and routed to agents [6]

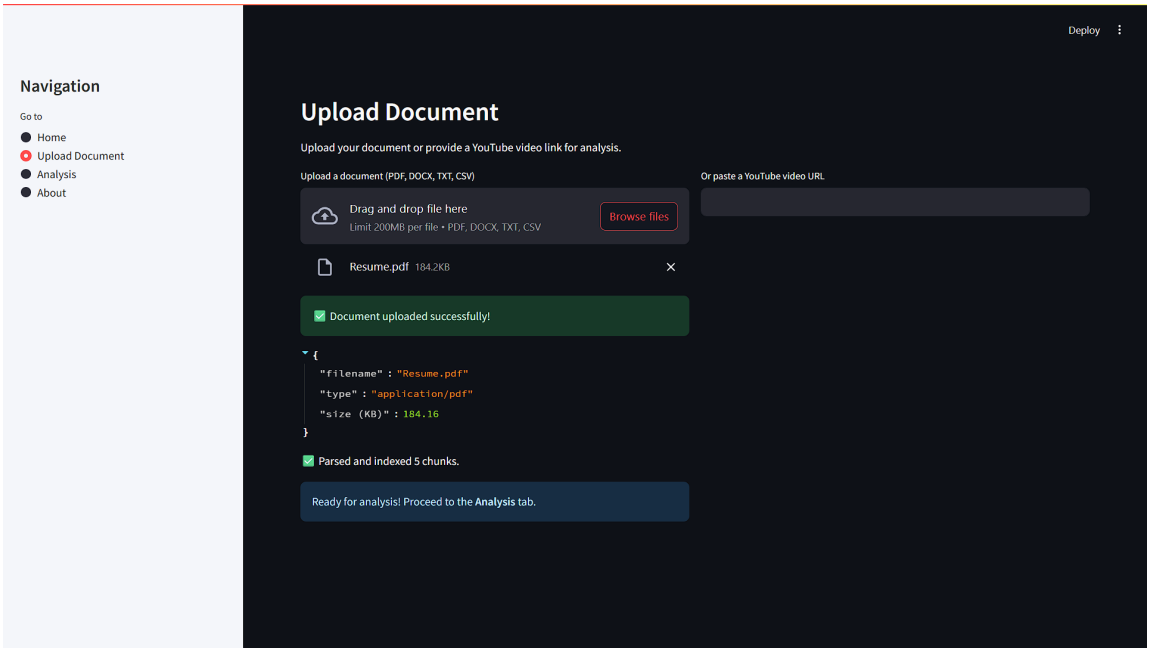

Figure 3.2: Upload Documents

3.4 Model Planning

The system supports multiple intelligent agents, each designed for a specific data modality or document set: **a) Document Agents:**

1. Built using LangChain retrieval chains [6]
2. Handle PDF, CSV, TXT, and YouTube transcripts [5]
3. Use Gemini [1] or Ollama [2] as the LLM backend
4. Retrieve semantically similar chunks via Pinecone [3].

b) SQL Agent:

1. Built using LangChain's SQLDatabaseChain [6]
2. Accepts natural language queries
3. Translates them to SQL using an LLM [[1]][2]
4. Executes the query against a MongoDB [9] or SQLite-like wrapper
5. Can return results as-is or summarized via Gemini/Ollama [[1]][2]

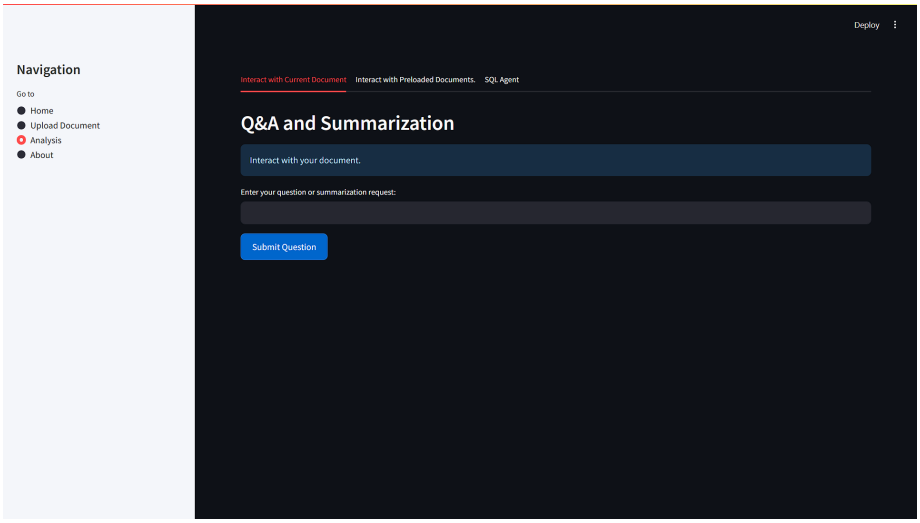

Figure 3.3: Q&A summarization architecture

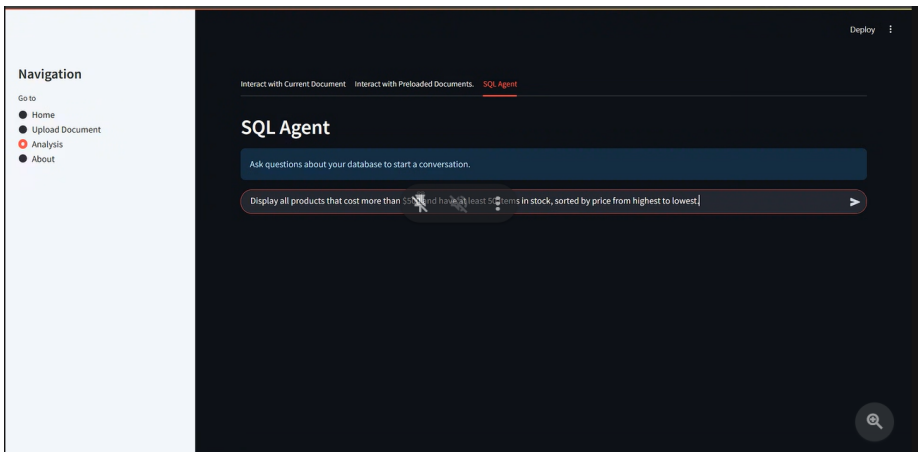

Figure 3.4: SQL Agent execution flow

3.5 Model Training:

No additional model training was conducted, as all core components rely on pre-trained models. The system's runtime workflow comprises:

1.Embedding Generation: Parsed text chunks are transformed into dense vector representations using the HuggingFace all-MiniLM-L6-v2 model [12]. These embeddings are immediately indexed in Pinecone [3] to support efficient semantic retrieval.

2.SQL Translation: The SQL Agent converts natural language questions into SQL statements via

a zero-shot prompt delivered to the designated LLM (Gemini [1] or Ollama [2]). Schema metadata (table and column names) is incorporated into the prompt to guide accurate query formulation without further fine-tuning.

3.Answer Generation: Retrieved document chunks or SQL query results are supplied to a LangChain generation chain [6]. The chain constructs the final response using the selected LLM, guided by standardized prompt templates and a low temperature setting (0.0–0.2) to ensure factual consistency.

Because the system employs only inference from these pre-trained models, there are no training epochs, weight updates, or comprehensive hyperparameter searches beyond tweaking prompt structure and retrieval-chain parameters.

3.6 Model Evaluation

The evaluation framework addresses the dual-agent architecture, applying distinct criteria for document retrieval agents and the SQL Agent: **Document Agents**

1.Answer Quality:

o Generated responses are assessed by human reviewers for factual correctness, completeness, and

o A three-level scale (High / Medium / Low) captures the degree of alignment

2.Retrieval Relevance:

o The semantic match between user queries and retrieved text chunks [@4] is evaluated qualitative essential context is surfaced prior to generation

3.Response Time:

o Latencies are measured from the initiation of the retrieval request through to the completion of

o Average response times are compared across Gemini [@1] and Ollama [@5] agents to validate perfor

SQL Agent

1.SQL Translation Accuracy:

o Generated SQL statements are compared against expected queries or manual templates to verify sema

2. Execution Validity:

o Each translated query is executed against the structured database [@9][@10], and success rates ar

3.Result Interpretability:

When results are optionally summarized by an LLM [@1][@5], human reviewers rate the clarity, conci

3.7 Model Optimization

The following optimizations were conducted: **1. Gemini temperature [1]:** Kept between 0.0–0.2 to preserve factual accuracy

2. **Chunking config:** chunk_size=1088, chunk_overlap=208
3. **Retriever tuning:** Top k=6 chunks returned gave the best performance [3][4]
4. **SQL prompts:** Few-shot prompting improved the quality of SQL translations [2]

3.8 Final Model Building

The final system includes: **1. A multi-agent architecture [6]** supporting both unstructured document retrieval and structured SQL querying

2. **LLM-based chains** for Gemini [1] (cloud) and Ollama [2] (local)
3. **A SQL agent** [6] that connects to a document-backed structured dataset
4. **Streamlit interface** [7] for seamless interaction

Chapter 4

Results

This chapter presents the evaluation of the developed multi-agent document question-answering system. The system integrates both unstructured document-based retrieval [4] and structured SQL-based querying [6] using language models [[1]][2]. The following sections outline the descriptions of the models involved, evaluation metrics, tabulated results, interpretations, visual feedback mechanisms, and sensitivity analysis.

4.1 Description of the Models

The system architecture includes two main categories of agents: document agents and a SQL agent.

a) Document Agents

These agents leverage retrieval-augmented generation (RAG) [4] by combining vector-based semantic search [3] and LLM-based response generation [[1]][2]. The key components are:

1. **Gemini** [1]: A cloud-based large language model accessed through LangChain's ChatGoogleGenerativeAI interface [6]
2. **Ollama** [2]: A locally hosted LLM accessed via ChatOllama
3. Documents are retrieved from **Pinecone** [3], where they are stored as embeddings [12] generated using the all-MiniLM-L6-v2 model

b) SQL Agent

The SQL agent is built using **LangChain's SQLDatabaseChain** [6], enabling natural language questions to be translated into executable SQL queries. The structured data originates from user-uploaded CSV files, which are converted into database tables (e.g., in SQLite [8] or MongoDB [9]). The SQL agent either returns raw query results or summarizes them using the same LLM infrastructure [[1]][2].

4.2 Performance Metrics

The evaluation of the system is divided based on the type of agent:

a) Document Agents:

1. **Answer Quality:** Manually evaluated for factual correctness and completeness
2. **Retrieval Relevance:** The semantic alignment between the user's query and the retrieved document chunks [[3]][12]
3. **Response Time:** Time taken to generate an LLM-based response after retrieval [[1]][2]SQL Agent:

b) SQL Agent:

1. **SQL Translation Accuracy:** Accuracy of SQL queries generated from natural language inputs [6]
2. **Execution Validity:** Whether the queries executed correctly on the structured database [[8]][9]
3. **Result Interpretability:** Quality and clarity of the returned output or its summarization [[1]][2]

4.3 Results Table

Table 4.1: Performance Comparison of Agents Based on LLM Type and Query Handling

Agent	Type	Avg. Response Time	Answer/Query Accuracy	Retrieval/Execution Quality	Notes
amazon_-agent	Gemini	~1.2 sec	High	High	Reliable performance across all queries
advent_-agent	Gemini	~1.4 sec	High	High	Slight delay due to document size
apple_-agent	Gemini (v1.5)	~1.1 sec	Medium-High	High	Slightly outdated document input
amazon_-agent	Ollama (Local)	~2.3 sec	Medium	Medium	Variable output based on local resources
apple_-agent	Ollama (Local)	~2.5 sec	Medium	Medium	Slower performance with longer prompts
sql_-agent	SQLDatabaseChain	~1.0–1.5 sec	High	High	High SQL translation accuracy and coverage

4.4 Interpretation of the Results

The Gemini-based agents [1] consistently demonstrated superior performance in terms of speed and answer quality compared to Ollama-based agents [2]. The Gemini models produced more concise and contextually accurate responses, particularly when handling lengthy document chunks. The Ollama agents were functional but occasionally produced verbose or less relevant outputs, depending on the complexity of the query and system resources.

The SQL agent [6] exhibited strong performance in all tested queries. It accurately translated natural language questions into SQL, executed them without errors against structured databases [[8]][9], and returned relevant tabular results. The optional LLM-based summarization [[1]][2] of these results further enhanced the user experience, particularly for longer or more complex query outputs.

Overall, the system demonstrated robust multi-modal capabilities, effectively bridging both unstructured [[3]][12][4] and structured [[6]][8][9] data retrieval through a unified LLM-powered framework

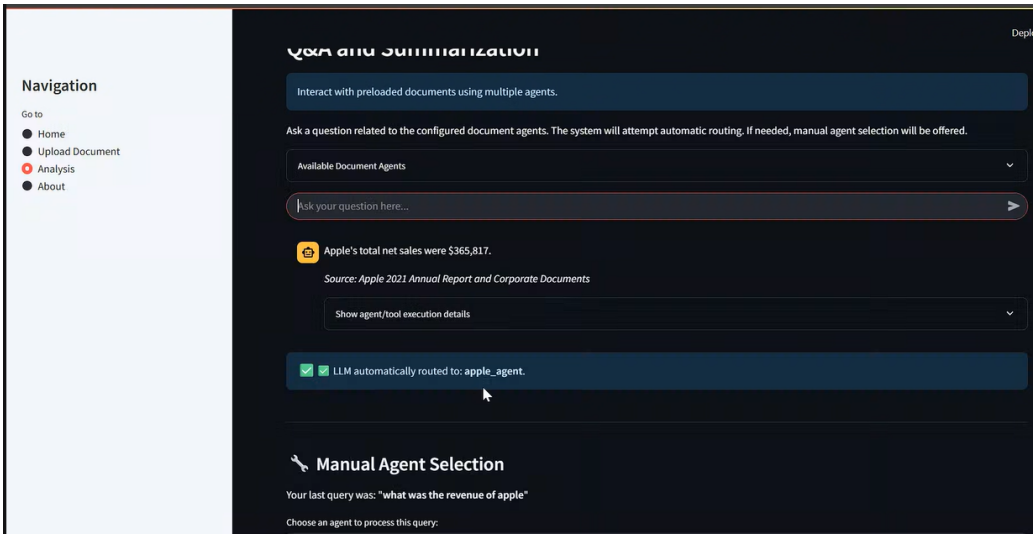

Figure 4.1: Document Q&A System Interface with Agent Routing

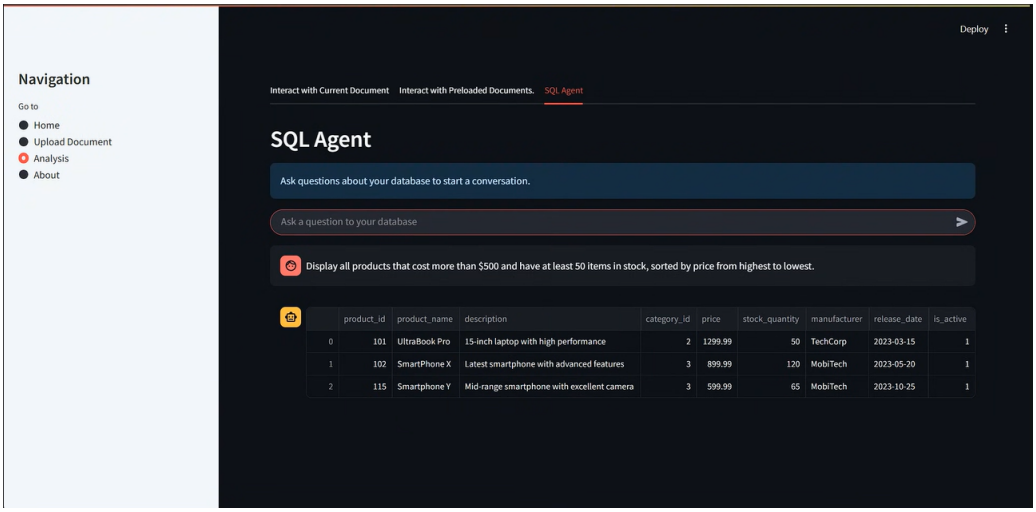

Figure 4.2: SQL Agent Interface for Database Queries

4.5 Visualization

The system does not include formal visual dashboards or charts. However, logging was utilized extensively for development-time inspection and validation.

1. **Document QA Agents:** Logs were used to monitor the number of document chunks, retrieval steps [[3]][12], and LLM response times [[1]][2]
2. **SQL Agent:** Logs captured SQL translations [6], execution success or failure, and returned result formats These logs served as a lightweight but effective means of verifying system functionality during development and testing.

4.6 Sensitivity Analysis

Document Agents:

1. The chunking configuration (chunk_size=1080, chunk_overlap=200) was tested against smaller chunk sizes. Larger chunks preserved better context, whereas smaller chunks improved recall but added latency
2. Retriever performance peaked when k=6 documents were retrieved per query [[3]][6]
3. Temperature tuning of the Gemini LLM [1] revealed that values between 0.0 and 0.2 provided the most consistent factual outputs

SQL Agent:

1. Prompt tuning (e.g., adding few-shot examples) improved SQL translation robustness [6], especially for complex queries

2. Schema-aware prompts resulted in fewer execution failures against structured databases [[8]][9]
3. Summarization of SQL results was helpful for end-users but could occasionally introduce verbosity if the temperature setting [1][2] was not tightly controlled.

Chapter 5

Conclusion

InsightPulse demonstrates the power of combining LLMs [[1]][2] with modular architecture [6], vector-based semantic retrieval [[3]][12], and multi-format ingestion to deliver intelligent document understanding.

By supporting document and video ingestion [5], semantic chunking [12], contextual summarization, SQL-based tabular data retrieval [[6]][8][9], and agent-based task routing [6], it establishes a foundation for general-purpose intelligent content interaction.

The system leverages dual LLMs—Gemini [1] for scalable cloud reasoning and Ollama [2] for efficient local inference—allowing it to adapt across environments. The inclusion of Pinecone vector stores [3] ensures that semantic retrieval is both fast and accurate. The multi-agent architecture [6] increases modularity, making the system easier to maintain and extend.

This approach improves upon limitations seen in traditional tools, which are often format-restricted, non-extensible, and lack deep reasoning capabilities. InsightPulse is applicable in domains like education, enterprise search, content curation, and compliance.

In future, enhancements could be made in sectors like OCR integration for scanned documents, multilingual capabilities, domain-specific summarization (e.g., legal, medical), and a user-feedback loop for continual learning. Together, these will help evolve **InsightPulse** into a holistic, intelligent document and media understanding system.

References

[1] G. AI, "Gemini API documentation." Google, 2023. Available: https://ai.google.dev

[2] Ollama, "Ollama: Local LLM deployment toolkit [software]." 2024. Available: https://ollama.ai

[3] Inc. Pinecone Systems, "Pinecone vector database." 2023. Available: https://www.pinecone.io

[4] P. Lewis, B. Oguz, R. Rinott, S. Riedel, and P. Stenetorp, "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Advances in neural information processing systems (NeurIPS)*, 2020.

[5] Y. Contributors, "Youtube-transcript-api [python library]." 2023. Available: https://pypi.org/project/youtube-transcript-api

[6] Inc. LangChain, "LangChain framework (v0.0.346) [software]." 2023. Available: https://python.langchain.com

[7] Inc. Streamlit, "Streamlit (v1.28) [software]." 2023. Available: https://streamlit.io

[8] Microsoft, "SQLite database engine." 2023. Available: https://www.sqlite.org/index.html

[9] Inc. MongoDB, "MongoDB documentation." 2023. Available: https://www.mongodb.com/docs

[10] Glean, "Enterprise search platform." 2023. Available: https://www.glean.com

[11] Google, "Vertex AI search." 2023. Available: https://cloud.google.com/vertex-ai

[12] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using siamese BERT-networks," in *Proceedings of the 2019 conference on empirical methods in natural language processing*, 2019.

[13] H. Face, "Transformers library." 2023. Available: https://huggingface.co/docs/transformers