Insurance Policy Assistant: RAG-based Chatbot

Problem Statement

Understanding insurance policies is challenging for many people due to complex legal language, lengthy documents, and domain-specific terminology. This creates barriers to comprehension and often leads to misunderstandings about coverage details.

The Insurance Policy Assistant addresses this problem by:

1. Providing an intuitive interface for uploading insurance documents
2. Allowing users to query their policies in natural language
3. Presenting accurate, contextual responses extracted directly from policy documents
4. Maintaining conversation context for follow-up questions

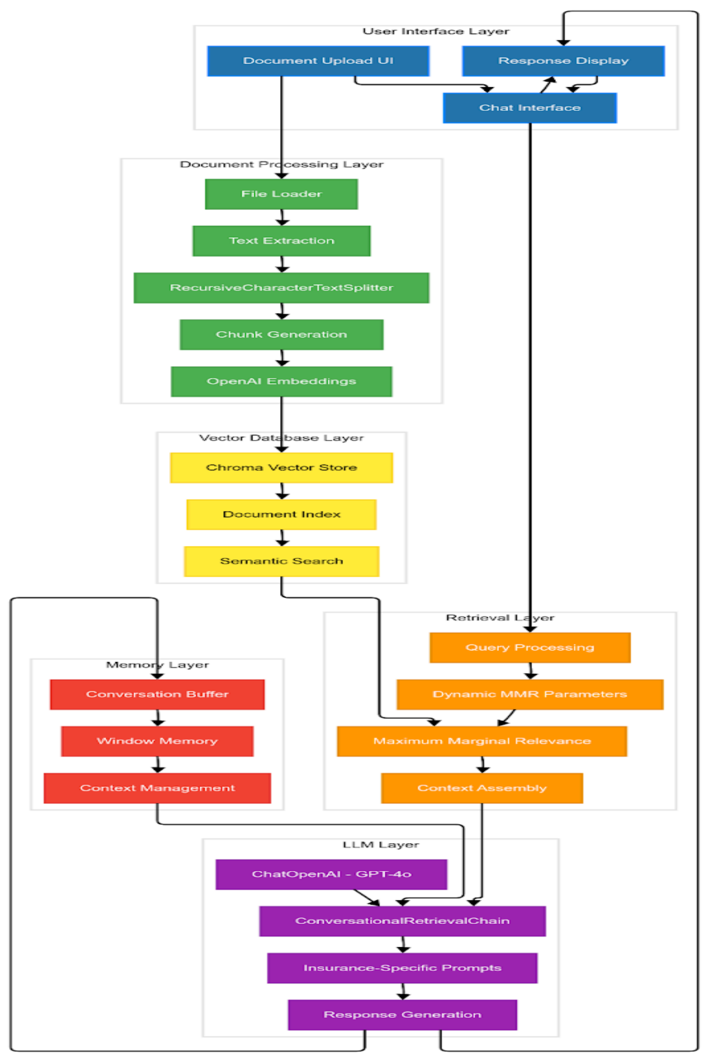
LangChain is an ideal framework for this implementation because it provides:

- Document processing utilities for handling PDFs and text files
- Text chunking mechanisms to properly segment insurance documents
- Vector store integration for semantic retrieval of policy information
- Conversation memory to maintain context across multiple queries
- Retrieval-augmented generation (RAG) patterns to ground LLM responses in source documents

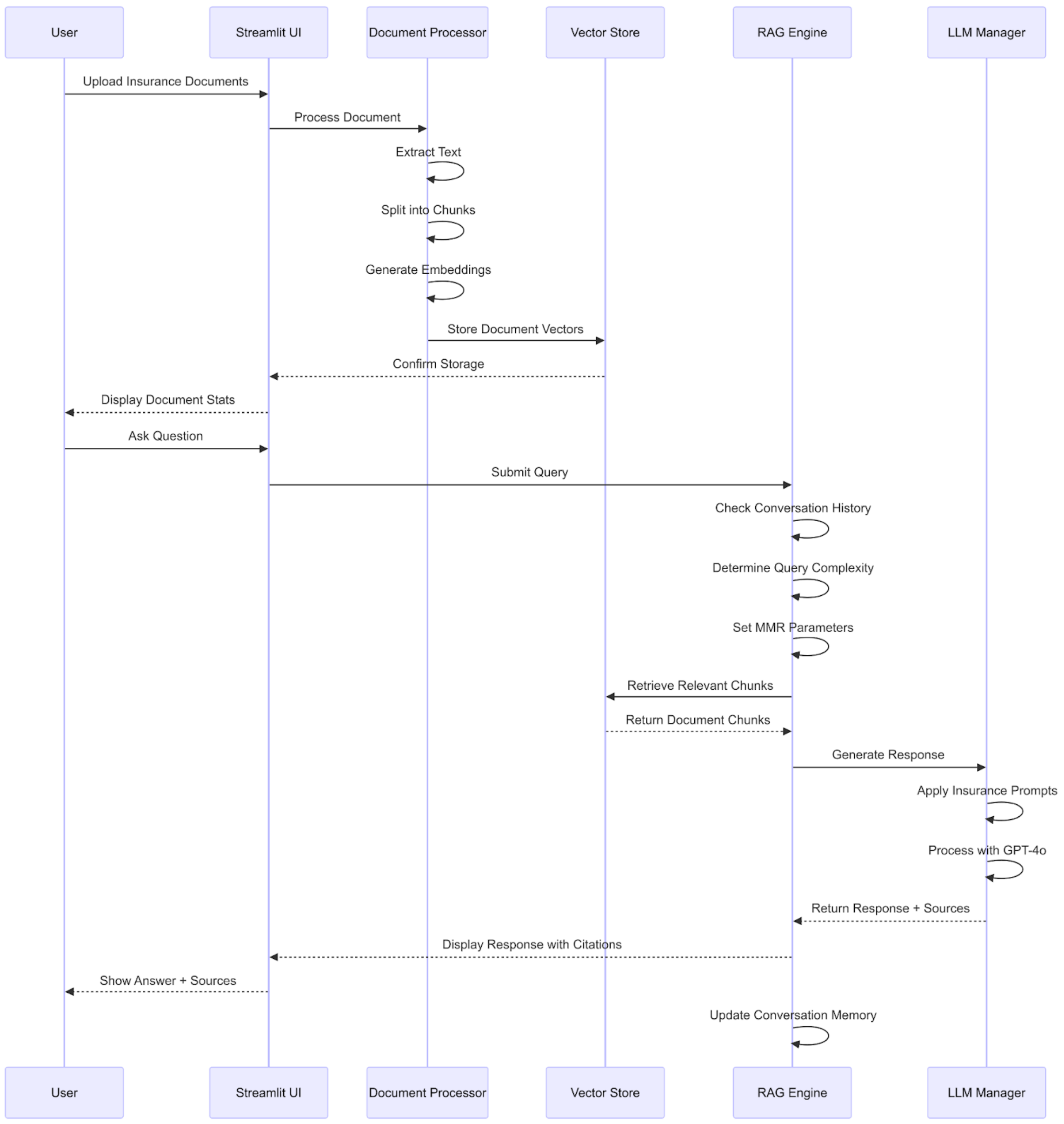
System Design

Architecture Overview

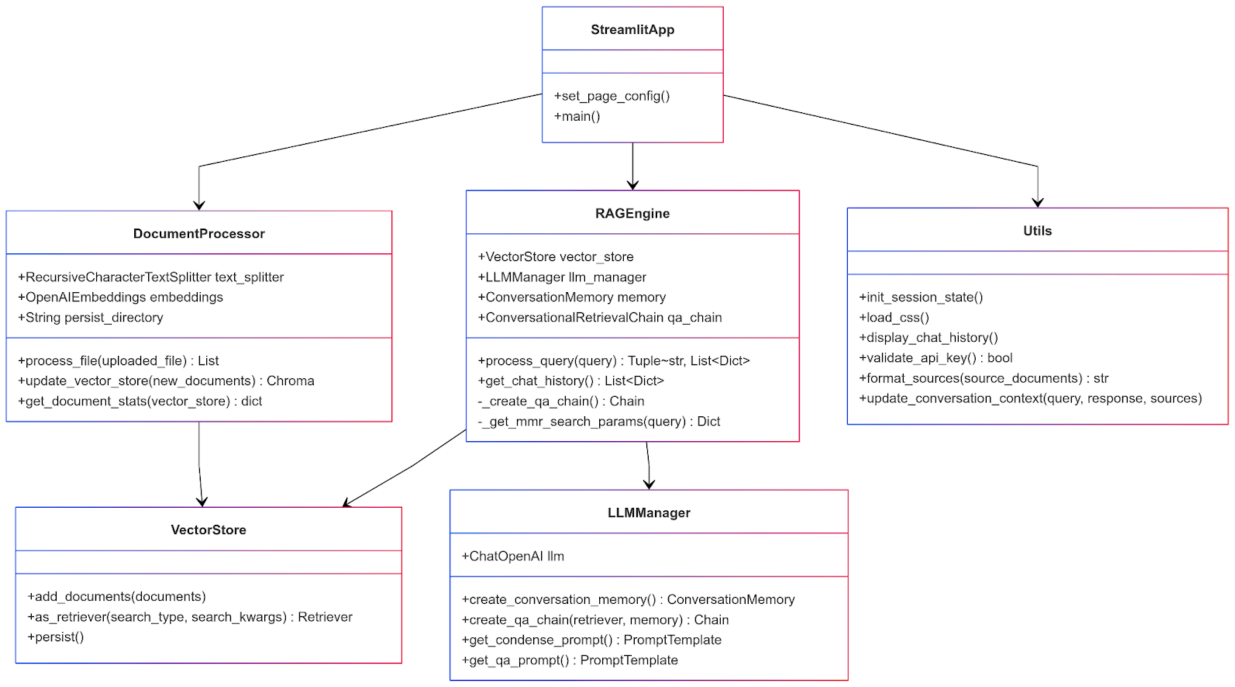
Layered System Architecture - Shows the distinct layers of the system (UI, Document Processing, Vector Database, Retrieval, LLM, and Memory) with color-coded components and their interactions.



Sequence Diagram - Illustrates the temporal flow of operations for both document upload and query processing, showing exactly how data moves between components.



1. **Component Diagram** - Provides a UML-style class diagram showing the key classes, their important methods, and their relationships.


StreamlitApp
+set_page_config()
+main()
DocumentProcessor
+RecursiveCharacterTextSplitter text_splitter
+OpenAIEmbeddings embeddings
+String persist_directory
+process_file(uploaded_file) : List
+update_vector_store(new_documents) : Chroma
+get_document_stats(vector_store) : dict
RAGEngine
+VectorStore vector_store
+LLMManager llm_manager
+ConversationMemory memory
+ConversationalRetrievalChain qa_chain
+process_query(query) : Tuple~str, List<Dict>
+get_chat_history() : List<Dict>
-_create_qa_chain() : Chain
-_get_mmr_search_params(query) : Dict
Utils
+init_session_state()
+load_css()
+display_chat_history()
+validate_api_key() : bool
+format_sources(source_documents) : str
+update_conversation_context(query, response, sources)
VectorStore
+add_documents(documents)
+as_retriever(search_type, search_kwargs) : Retriever
+persist()
LLMManager
+ChatOpenAI llm
+create_conversation_memory() : ConversationMemory
+create_qa_chain(retriever, memory) : Chain
+get_condense_prompt() : PromptTemplate
+get_qa_prompt() : PromptTemplate

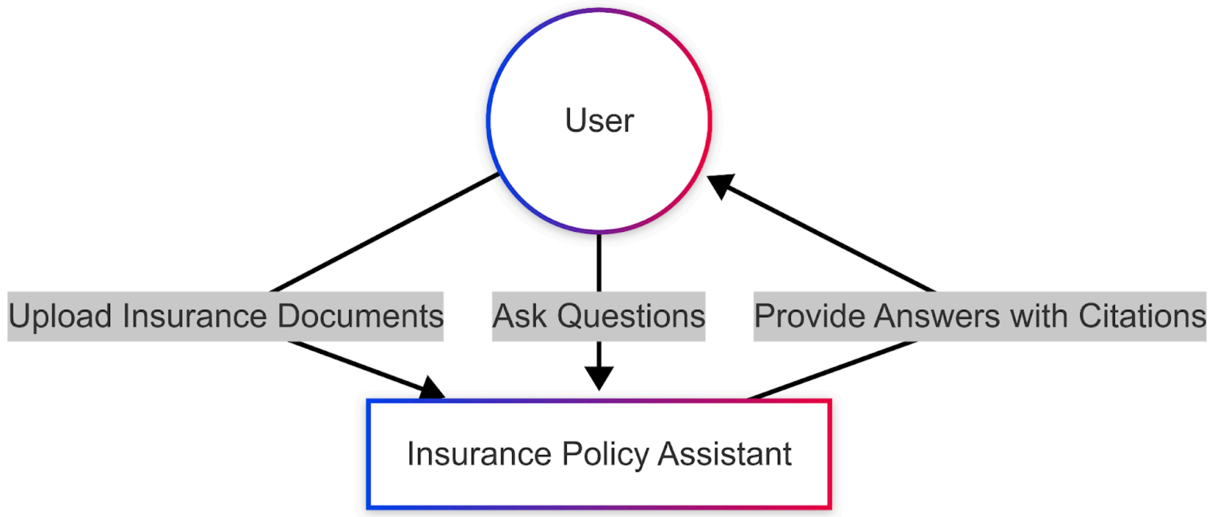
The system follows a modular design with these primary components:

1. **Document Processor**: Handles file uploads, text extraction, chunking, and embedding generation
2. **Vector Store**: Stores embedded document chunks for semantic search
3. **RAG Engine**: Processes user queries through a retrieval-augmented pipeline
4. **LLM Manager**: Manages LLM configuration and specialized prompting for insurance context
5. **User Interface**: Streamlit-based interface for document upload and chatbot interaction

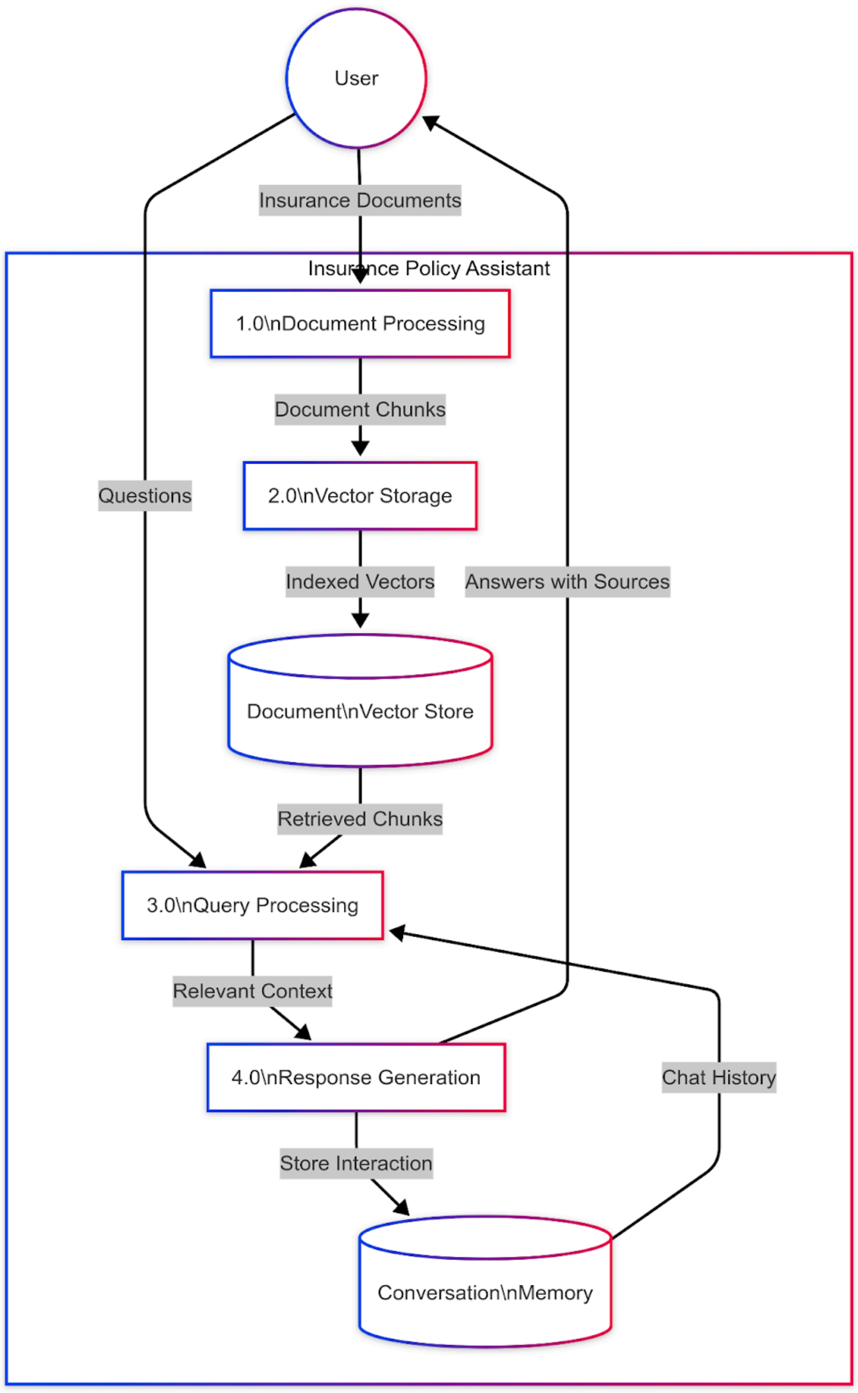
Data Flow

1. User uploads insurance policy documents (PDF/TXT)
2. Documents are processed:
    - Text extraction
    - Chunking into semantic units
    - Embedding generation
    - Storage in Chroma vector database
3. User submits a natural language query
4. System processes the query:
    - Retrieves relevant document chunks using Maximum Marginal Relevance (MMR)
    - Incorporates conversation history for context
    - Generates response using GPT-4o
    - Provides source attribution back to specific policy documents
5. User receives response with citations

DFD :
Context Level DFD - Shows the system as a single process with external entities


User
Upload Insurance Documents
Ask Questions
Provide Answers with Citations
Insurance Policy Assistant

DFD - Breaks down the system into major processes and data stores.


User
Insurance Documents
Insurance Policy Assistant
1.0\nDocument Processing
Document Chunks
Questions
2.0\nVector Storage
Indexed Vectors
Answers with Sources
Document\nVector Store
Retrieved Chunks
3.0\nQuery Processing
Relevant Context
4.0\nResponse Generation
Chat History
Store Interaction
Conversation\nMemory

Key Innovations

1. **Dynamic MMR Search Parameters**: The system adjusts retrieval parameters based on query complexity, increasing document diversity for comparison questions
2. **Specialized Insurance Prompting**: Custom prompts designed specifically for insurance domain questioning
3. **Context-Aware Conversation**: Maintains topic awareness across the conversation
4. **Source Attribution**: All responses include references to specific policy documents

Code Implementation

The implementation consists of five main modules:

1. Document Processor (`document_processor.py`)

Handles document ingestion pipeline:

- File upload and temporary storage
- Document loading using appropriate loaders (PDF/Text)
- Text chunking with RecursiveCharacterTextSplitter
- Vector embedding with OpenAI embeddings
- Chroma vector database management

Key optimization: Persistent storage to maintain vector database between sessions

2. RAG Engine (`rag_engine.py`)

Core retrieval and generation pipeline:

- Manages the Retrieval-Augmented Generation process
- Implements dynamic MMR parameter tuning based on query characteristics
- Provides memory-augmented query processing
- Returns both answers and source documents for attribution

Key innovation: Dynamic retrieval parameter adjustment based on query complexity

3. LLM Manager (`llm.py`)

Handles LLM configuration and prompting:

- Sets up the ChatOpenAI instance with GPT-4o
- Creates specialized prompt templates for insurance domain
- Manages conversation memory with window buffering
- Constructs the ConversationalRetrievalChain

Key feature: Domain-specific prompting for insurance policy understanding

4. Utilities (`utils.py`)

Provides helper functions:

- Streamlit session state management
- Chat history display formatting
- API key validation
- Source document formatting
- Conversation context tracking

5. Main Application (`app.py`)

Integrates all components into a Streamlit interface:

- Document upload interface
- Chat interface with history display
- Document statistics display
- CSS styling for better user experience

Installation and Usage

Prerequisites

- Python 3.11 or higher
- OpenAI API key

Installation

1. Clone the repository:

git clone https://github.com/kajalmahata123/policy-pilot.git

cd insurance-policy-assistant

2. Install dependencies:

pip install -r requirements.txt

3. Set OpenAI API key:

export OPENAI_API_KEY="your-api-key-here"

4. Run the application:

```
streamlit run app.py
```

5. Access the application at http://localhost:8501

Usage Guide

1. Upload insurance policy documents using the sidebar
2. Ask questions about your policies in natural language
3. Review responses with source citations
4. Follow up with additional questions - the system maintains conversation context

Challenges and Solutions

Challenge 1: Document Chunking

Insurance documents contain complex hierarchical structures that can be difficult to chunk effectively.

Solution: Used RecursiveCharacterTextSplitter with hierarchical separators to preserve logical structure.

Challenge 2: Context Management

Insurance queries often require context from previous exchanges.

Solution: Implemented ConversationBufferWindowMemory with a window of 5 exchanges to maintain relevant context without overwhelming the system.

Challenge 3: Query Complexity Variation

Insurance queries range from simple coverage questions to complex scenario comparisons.

Solution: Dynamic MMR parameter adjustment based on query complexity to optimize retrieval for different question types.

Challenge 4: Source Attribution

Trustworthiness requires clear source attribution.

Solution: Integrated source document tracking throughout the pipeline and displayed source references with each response.

Future Improvements

1. **Multi-document comparison**: Enhance the system to explicitly compare clauses across multiple policies
2. **Named entity recognition**: Extract and index key entities (coverage limits, policy periods, etc.)
3. **User feedback loop**: Implement feedback mechanisms to improve retrieval performance over time
4. **Custom embeddings**: Train domain-specific embeddings for insurance terminology
5. **Edge deployment**: Optimize for local deployment using smaller models for private data processing

Conclusion

The Insurance Policy Assistant demonstrates how LangChain and RAG can be applied to practical document understanding problems. By combining best practices in document processing, retrieval optimization, and conversation management, the system provides a user-friendly interface for navigating complex insurance documents.