Semantic Spotter — IntelliPolicy

LangChain-based Insurance Q&A System

🧠 1. Problem Statement

Insurance policy documents are often long, complex, and filled with technical jargon that makes it challenging for customers and agents to find specific information such as claim limits, exclusions, coverage details, and waiting periods.

Traditional keyword-based searches are inadequate because they do not capture semantic context.

To address this challenge, IntelliPolicy leverages a Retrieval-Augmented Generation (RAG) system powered by LangChain and OpenAI models, capable of accurately answering natural language queries from one or more insurance policy documents.

🎯 2. Objectives

- Build an AI-driven system to query insurance policy documents conversationally.
- Retrieve and summarize relevant policy sections automatically.
- Provide accurate, context-based answers in simple language.
- Demonstrate real-world application of LangChain for RAG.

🚀 3. Project Goals

- Design an intelligent Q&A system for insurance document analysis.
- Implement an end-to-end LangChain-based RAG pipeline.
- Enable policy lookup for exclusions, claim limits, and coverage terms.
- Showcase how embeddings and LLMs can augment retrieval accuracy.

📂 4. Data Sources

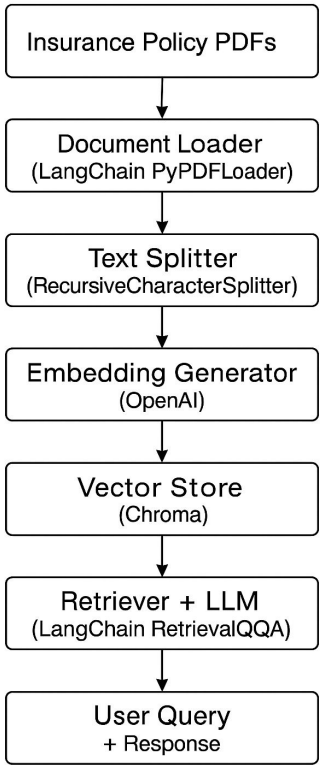
- Input: Sample or publicly available insurance policy PDFs stored in Google Drive.
- Loader: LangChain's `PyPDFLoader` to extract text from the documents.
- Processing: Text split into manageable chunks using `RecursiveCharacterTextSplitter`.
- Embedding Model: `text-embedding-3-small` from OpenAI.
- Vector Store: ChromaDB for similarity search and persistent storage.
-

🏗️ 5. System Design

The IntelliPolicy system follows a modular **RAG architecture** with seven stages:

1. **Document Loading** → Extract text using `PyPDFLoader`.
2. **Text Splitting** → Split into overlapping chunks for semantic continuity.
3. **Embedding Generation** → Convert text chunks into vectors using OpenAI embeddings.
4. **Vector Store Creation** → Store and persist embeddings in **ChromaDB**.
5. **Retriever Setup** → Retrieve top `k` relevant text chunks for each query.
6. **LLM Integration** → Use `ChatOpenAI (GPT-3.5-turbo)` to generate context-based answers.
7. **Query Response** → Display formatted answers via a custom `pretty_print_rag_result()` function.

🧭 6. System Architecture Flowchart



🧩 7. Design Choices

Framework: LangChain
→ Provides modular, scalable RAG pipelines with retrievers and chains.

Vector Store: ChromaDB
→ Lightweight, open-source, and easy to persist locally.

LLM Model: GPT-3.5 Turbo
→ Reliable, cost-effective, and accurate for contextual question answering.

Embedding Model: text-embedding-3-small
→ Efficient embedding generation for semantic similarity.

Chunking Strategy: RecursiveCharacterTextSplitter
→ Maintains sentence continuity with overlapping chunks.

Memory: Excluded
→ Each query is independent; memory is unnecessary for single-turn Q&A.

⚙️ 8. Implementation Highlights

Code Modularity:
Each core component (document loading, text splitting, embedding, retrieval, and generation) was implemented independently, making the pipeline easy to understand, debug, and extend.

Flexibility:
The system supports multiple policy documents and can be extended to include metadata filters or new vector stores in the future.

Transparency:
All intermediate steps — such as embeddings, chunk outputs, and retrieved results — are easily inspectable for debugging and optimization.

Reproducibility:
The project runs entirely in **Google Colab**, ensuring consistent results and easy re-execution for evaluators and reviewers.

Output Formatting:
A custom `pretty_print_rag_result()` function formats query responses neatly, making them ready for screenshots and inclusion in the project repo

💡 9.Challenges Faced

PDF Text Extraction:
Extracting clean, structured text from insurance PDFs was challenging due to tables, footnotes, and inconsistent formatting.

Chunk Size Optimization:
Finding the right balance between chunk size and overlap to retain context while staying within token limits required experimentation.

Embedding Performance:
Generating embeddings for long documents introduced latency and cost challenges when using OpenAI's API.

Retrieval Accuracy:
Ensuring the retriever fetched contextually relevant and accurate chunks was crucial for improving the model's final responses.

Model Latency & Rate Limits:
Encountered occasional slow responses and API rate limits while testing multiple queries in Colab.

🚀10. Future Enhancements

Multi-Document Support:
Extend the system to handle multiple policy documents simultaneously for broader coverage.

Metadata-Based Retrieval:
Enable filtering by insurer, policy type, or category to improve query precision.

Web Interface Integration:
Develop a user-friendly **Streamlit** or **Gradio** interface for real-time interaction.

Hybrid Retrieval:
Combine semantic and keyword-based search to improve retrieval accuracy.

Summarization & Confidence Scoring:
Incorporate concise answer summarization and add confidence scores for reliability.

Local/Open-Source Models:
Explore models like **Llama 3** or **Mistral** for cost-effective, offline deployments.

🧾 11. Conclusion

The **IntelliPolicy** project demonstrates how **LangChain** can power a full Retrieval-Augmented Generation (RAG) pipeline for document Q&A.
It bridges the gap between unstructured insurance policy text and user-friendly question answering.

Key takeaways:

- Developed an end-to-end RAG workflow using LangChain and OpenAI.
- Enabled semantic retrieval for better information accuracy.
- Showcased modular, reproducible design ideal for scalable applications.

This project lays the groundwork for building intelligent, domain-specific AI assistants across industries.

🧰 12. How to Run the Project

1. Open `Semantic_Spotter_Sunil.ipynb` in **Google Colab**.
2. Mount Google Drive and provide the path to the insurance PDF.
3. Run cells sequentially to:
    a. Install dependencies
    b. Load and split documents
    c. Create embeddings
    d. Build and query the RAG pipeline
4. Test queries using:

```
pretty_print_rag_result(rag_pipeline({"query": "What is the claim limit for pre-existing diseases?"}), "What is the claim limit for pre-existing diseases?")
```

5. View formatted output directly in Colab and take screenshots for the report.

🧮 13. Lessons Learned

- Learned how **LangChain** abstracts retrieval and generation pipelines.
- Understood the trade-offs in chunking, embedding, and retrieval accuracy.
- Gained practical experience integrating OpenAI APIs in a real-world use case.

Sample Output Screenshot

```
query = "What are the eligibility requirements for employees under this group insurance policy?"
result = rag_pipeline({"query": query})
pretty_print_rag_result(result, query)
```

```
================================================================================================================================
🧠 INTELLIPOLICY: INSURANCE POLICY Q&A SYSTEM
================================================================================================================================

🔷 Question:
What are the eligibility requirements for employees under this group insurance policy?

💬 Assistant Answer:

Employees must enroll in the insurance policy to be eligible for coverage. At least 75% of all
eligible employees must enroll to maintain eligibility under this group insurance policy.

================================================================================================================================
✅ Response Generated Successfully!
================================================================================================================================
```