Government Policy Analysis using Retrieval Augmented Generation (RAG)

A Retrieval-Augmented Pipeline for Contextual Summarization and Policy Evaluation

Paavni Singh

1. Problem Statement

- Government policy documents are often **lengthy**, **dense**, and **difficult to analyze** at scale.
- Manual analysis of effectiveness, sentiment, and impact is **time-consuming**.
- There is a need for **automated, explainable systems** that can:
  - Summarize policies
  - Extract relevant information
  - Avoid dependency on external APIs (OpenAI, etc.)

2. Objective

To build an **AI-powered document analysis system** that can:

- Summarize policy PDFs (like *LaQshay*) using tree-based techniques.
- Retrieve specific policy content using semantic search.
- Route queries intelligently between summarization and retrieval engines.
- Run **entirely offline** using **local LLMs (LLaMA2)**.

3. Tools and Technologies

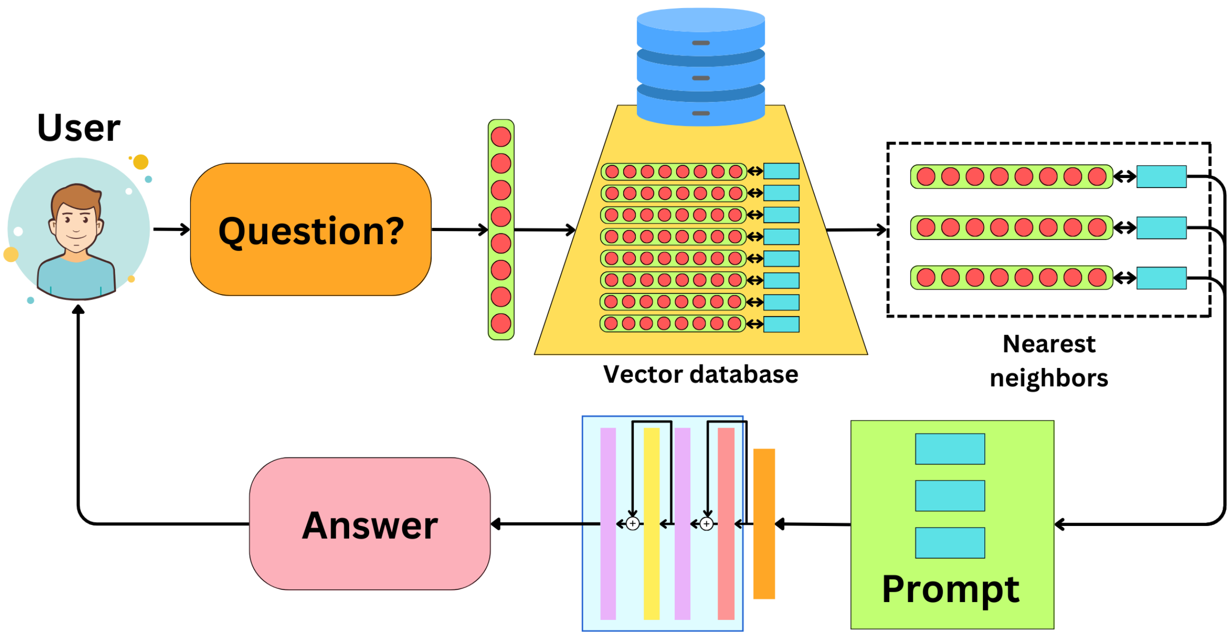
Component	Tool/Framework
Document Parsing	SimpleDirectoryReader, LlamaParse
Vector Indexing	LlamaIndex
LLM	LLaMA2 via Ollama
Summarization	SummaryIndex + tree_summarize
Retrieval	VectorStoreIndex
Routing Logic	RouterQueryEngine + LLMSingleSelector
Programming Language	Python

4. Architecture Diagram

A simple diagram showing the pipeline:

PDF → Sentence Splitter → Index → Router → Response

- Summarization Route → `SummaryIndex`
- Retrieval Route → `VectorStoreIndex`



5. Implementation

i) Document Ingestion:

```
documents = SimpleDirectoryReader(input_files=["/Users/paavnisingh/govt-pol/RKSK.pdf"]).load_data()
```

ii) Sentence Splitting + Indexing + LLM Setup:

```

splitter = SentenceSplitter(chunk_size=1024)
nodes = splitter.get_nodes_from_documents(documents)
llm = Ollama(model="llama2")

```

iii) Create Summary & Vector Indices:

```
summary_index = SummaryIndex(nodes)
vector_index = VectorStoreIndex(nodes)

```

iv) Router Engine with Query Tools:

6. Results & Observations

- Achieved **accurate contextual summaries** without internet dependency.

```python
summary_query_engine = summary_index.as_query_engine(
    response_mode="tree_summarize",
    use_async=True,
)
vector_query_engine = vector_index.as_query_engine()
from llama_index.core.tools import QueryEngineTool


summary_tool = QueryEngineTool.from_defaults(
    query_engine=summary_query_engine,
    description=(
        """Useful for summarization questions related to RKSK policy from multiple papers
        Also useful for making a list of all the analysis done based on the query."""
    ),
)

vector_tool = QueryEngineTool.from_defaults(
    query_engine=vector_query_engine,
    description=(
        "Useful for retrieving specific context from the RKSK policy from multiple papers"
    ),
)



query_engine = RouterQueryEngine(
    selector=LLMSingleSelector.from_defaults()  ,
    query_engine_tools=[
        summary_tool,
        vector_tool,
    ],
    verbose=True
)
```

- The ***RouterQueryEngine*** handled intent switching intelligently.
- Local inference using *Ollama* kept everything privacy-focused.
- *SummaryIndex* provided concise outputs, while *VectorStoreIndex* enabled deep semantic lookups.

7. Limitations

- Local LLMs (like llama2) can be **slower** and **less accurate** than GPT-4.
- Summarization may fail on **extremely large PDFs** (>100 pages) unless chunked well.
- No sentiment scoring model was used; can be added with fine-tuned classifiers.

8. Future Work

- Integrate a **feedback loop** to improve summaries over time.
- Use **RAG with fine-tuned local models** for even deeper policy analysis.
- Add **user submission analysis** module (compare user ideas to past policies).

9. Conclusion

- Successfully built a **retrieval + summarization pipeline** using **LlamaIndex** and **LLaMA2**.
- Enables **offline policy document analysis** with explainable output.
- Can be scaled for **government think tanks**, **academia**, and **citizen engagement portals**.