# **Semantic Spotter Assignment : Automated System for Extracting, Comparing, and Summarizing Insights from Government Documents**

Badari Prasad H

Table of Contents

[Introduction 2](#_Toc185700356)

[Problem Statement 2](#_Toc185700357)

[Solution 3](#_Toc185700358)

[Approach to Problem Resolution 3](#_Toc185700359)

[Architecture 4](#_Toc185700360)

[System Flow 4](#_Toc185700361)

[Challenges Faced 4](#_Toc185700362)

[Learnings 4](#_Toc185700363)

[Why use QdrantVectorStore 5](#_Toc185700364)

[Future Implementations 6](#_Toc185700365)

[Code Url 6](#_Toc185700366)

[Appendix 6](#_Toc185700367)

[References 7](#_Toc185700368)

## Introduction

The project's goal is to create an automated system that extracts, compares, and summarizes information from Indian government documents, such as Prime Minister speeches, President speeches, and budget releases. By utilizing advanced technology, this system aims to deliver valuable insights to stakeholders, including researchers, policymakers, and the general public.

## Problem Statement

The challenge involves efficiently retrieving data from various sources like eparlib.nic.in and sansad.in, analyzing the documents to identify common themes, trends, and differences, and summarizing the information into clear, concise summaries. This requires developing a robust system capable of handling diverse document types and extracting meaningful insights from them.

Create an automated system to extract, compare, and summarize information from Indian government documents, such as Prime Minister speeches, President speeches, and budget releases. Ensure the system retrieves data from the specified URLs:

* <https://eparlib.nic.in/>
* <https://sansad.in/ls/knowledge-centre/speeches>

The system should be able to:

1. **Information Handling:** Automatically extract text content from Prime Minister speeches, President speeches, and budget releases.
2. **Comparative Analysis:** Examine the retrieved documents to uncover common themes, trends, and distinctions among various types of speeches and reports. This analysis may include sentiment analysis, topic modeling, or other relevant techniques to compare the content.
3. **Summarizing Text:** Condense the extracted information into clear, concise summaries. These summaries should highlight the key points and main ideas from the documents, allowing users to quickly understand the essential content without reading through lengthy texts.

The system should be efficient, accurate, and user-friendly, offering valuable insights into the content of Indian government documents for stakeholders like researchers, policymakers, and the general public.

## Solution

The proposed solution strategy involves building a proof of concept (POC) to address the requirements. By utilizing tools like LlamaIndex for efficient data processing and Hugging Face embeddings for text analysis, the system aims to extract key information, identify trends and comparisons, and generate concise summaries. This approach ensures accuracy and lays the foundation for further improvements and customization.

**LLM Model Selection**

* **LLM Model:** gpt-3.5-turbo by OpenAI.
* **Embedding model:** HuggingFaceEmbedding "BAAI/bge-small-en-v1.5"

### Approach to Problem Resolution

Develop a proof of concept (POC) that meets the following requirements:

* Extract key information, identify trends and comparisons, and generate concise summaries.
* Offer valuable insights for researchers, policymakers, and the public to understand government communication.

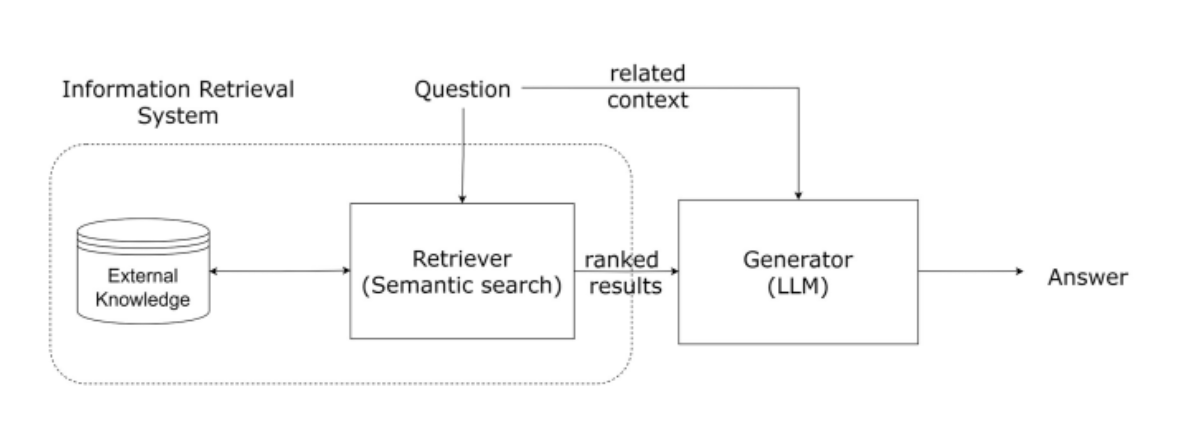
**Goal** - Successfully addressing these two requirements in the POC will ensure the overall model's accuracy is high, making further improvements and customizations worthwhile..

**Data Used** - Indian government documents are available as PDFs and are organized into four folders: "Prime Minister Speech", "President Speech", and "Interim Budget".

**Tools used** - LlamaIndex has been utilized for now because of its powerful query engine, rapid data processing with data loaders and directory readers, and its ease of implementation with fewer lines of code.

## Architecture

The system architecture consists of several components, including retrieving data from specified URLs, parsing and processing documents using tools like Simple Directory Reader and Sentence Splitter, generating embeddings with Hugging Face models, and indexing with Qdrant Vector Store. Query engines are then set up to facilitate efficient querying and response generation..



## System Flow

Once initialized, the system enables users to input questions or queries about government documents. The query engine processes these inputs, using both retrieval and summarization tools to produce relevant responses. These responses are then displayed to the users, allowing them to quickly access the necessary information.

## Challenges Faced

Throughout the development process, challenges emerged, including compatibility issues with dependencies and the integration of various components. Ensuring the accuracy and relevance of the extracted information also proved to be a significant hurdle. However, these challenges were overcome through rigorous testing and optimization efforts.

## Learnings

1. Efficient Document Retrieval: Utilized llamaIndex for rapid data processing and extraction from various government documents.
2. Comparative Analysis Insights: Employed multiple indices to examine and contrast speeches by the Prime Minister and President, committee reports, and budget announcements.
3. Concise Summarization Techniques: Used HuggingFaceEmbeddings and QdrantVectorStore to distill key information into clear summaries.
4. Effective Tool Selection: Selected appropriate tools like SummaryIndex for summarization and VectorStoreIndex for context retrieval.
5. Query Engine Configuration: Set up query engines for summarization and context retrieval to facilitate efficient user interactions.
6. Integration of Multiple Technologies: Combined llamaIndex, HuggingFaceEmbeddings, and QdrantVectorStore for thorough document analysis.
7. Collaborative Development: Worked with various tools and libraries to effectively tackle different aspects of the problem statement.
8. Scalability Considerations: Addressed scalability by considering factors like chunk size and context window for efficient processing of large document volumes.

## Why use QdrantVectorStore

Qdrant is an open-source and fully managed high-performance, massive-scale Vector Database for the next AI generation. The vector search engine provides a production-ready service with a convenient API to store, search, and manage vectors with an additional payload.

Qdrant provides a robust solution for utilizing vector stores, particularly when integrated with Langchain. Qdrant provides a robust framework for implementing retrieval methods that leverage both dense and sparse embeddings. This dual approach allows for enhanced search capabilities, making it suitable for various applications, including those utilizing the langchain qdrantvectorstore. The integration of these methods can significantly improve the efficiency and accuracy of information retrieval tasks.

**Chroma DB vs Qdrant: Key Differences**

The table below lays out the key differentiating features of Chroma DB and Qdrant:

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Chroma DB** | **Qdrant** |
| Scalability | Scales as per user needs. | Supports scaling via sharding. |
| Indexing | Handles indexing automatically. | Offers different indexing techniques, including payload index and full-text index. |
| Hybrid Search | You cannot implement any direct hybrid search within Chroma DB. | Supports hybrid search for sparse and dense vectors through Query API. |
| Data Security | Authentication is done through Static API, at-rest encryption, and SSL/TLS certificates for data in transit encryption. | Authentication through APIs, granular control with JWT, TLS for encryption, and RBAC for authorization and privacy. |
| Cost | Chroma DB is free and open source under the Apache 2.0 License. | Pricing depends on the deployment option. |

More at:

* <https://www.restack.io/p/vector-database-knowledge-langchain-qdrant-cat-ai>
* <https://airbyte.com/data-engineering-resources/chroma-db-vs-qdrant>
* <https://monotonic.ai/llm-answer-retrieval-with-qdrant-vector-database-8cc08d83ded7>

## Future Implementations

For future improvements, the system can be enhanced by adding features like sentiment analysis, topic modelling, and user feedback mechanisms. Additionally, expanding the system's capabilities to process a wider variety of government documents and languages could increase its utility and accessibility for a broader audience.

## Code Url

<https://github.com/badarihp/Semantic-Spotter-LlammaIndex.git>

## Appendix

The bge-small-en model is a small-scale English text embedding model developed by BAAI (Beijing Academy of Artificial Intelligence) as part of their FlagEmbedding project. Check this website for more detailed evaluation on : <https://huggingface.co/BAAI/bge-small-en-v1.5>

## References

<https://eparlib.nic.in/>

<https://sansad.in/ls/knowledge-centre/speeches>

<https://monotonic.ai/llm-answer-retrieval-with-qdrant-vector-database-8cc08d83ded7>

<https://huggingface.co/BAAI/bge-small-en-v1.5>

<https://www.restack.io/p/vector-database-knowledge-langchain-qdrant-cat-ai>

<https://airbyte.com/data-engineering-resources/chroma-db-vs-qdrant>

<https://monotonic.ai/llm-answer-retrieval-with-qdrant-vector-database-8cc08d83ded7>