KeyGenie:
Precision Insights Tool

Prepared by:

Ayat nour hachmi, Imane Zolati, Marwa Ghachi

Introduction

In the digital age, professionals and researchers often face the challenge of extracting key insights from complex documents to answer specific questions. Manual document processing is not only time-consuming but also prone to errors. Our tool **KeyGenie** addresses these challenges by allowing users to upload thematic documents (in ZIP format) and submit targeted questions. The system uses advanced AI techniques to process the documents and provide detailed, accurate responses based on the content. This approach ensures that users receive relevant answers with transparency, credibility, and efficiency.

Problem Statement

Challenges in Information Retrieval and Text Generation:

1. **Navigating Large Document Sets:**
    - Processing extensive PDFs or multi-document datasets to retrieve specific information is overwhelming and labor-intensive.
2. **Lack of Targeted Answers:**
    - Generic summaries and other tools often fail to provide precise answers to specific user questions.
3. **Transparency Issues:**
    - Users require clear references to original sources for verification, which many tools fail to provide.
4. **Customization Needs:**
    - Users need tailored responses to questions based on thematic focus or specific parameters, which other tools lack.
5. **Privacy and Security:**
    - Handling sensitive documents requires secure temporary storage and robust deletion protocols to maintain user trust.

Proposed Solution

Key Features of the Text Generation Tool: KeyGenie

1. **Question-Based Key Point Extraction:**
    - Users upload ZIP files containing thematic documents and provide specific questions for each key point.

    - The tool generates precise answers by retrieving relevant sections from the documents.
2. **AI-Powered Text Generation:**
    - Utilizes Retrieval-Augmented Generation (RAG) and semantic embeddings to generate contextually accurate responses.
3. **Dynamic Citation Integration:**
    - Each generated response includes inline citations referencing the original documents for transparency.
4. **Efficient Document Handling:**
    - Automatically parses and processes large documents into manageable chunks for efficient embedding and retrieval.
5. **Customizable Outputs:**
    - Allows users to define parameters for response depth, thematic focus, and key point priorities.
6. **Privacy and Security:**
    - Ensures secure handling of user data with automatic deletion after processing.

Methodology

- **File Upload and Document Extraction:**
    - The first step involves allowing users to upload ZIP files containing multiple documents. This process is facilitated by a user interface that uses Tkinter to allow users to select and upload ZIP files from their local system.
        - **File Upload**: The uploaded ZIP files are saved in the **uploads** directory on the server.
        - **ZIP Extraction**: After the files are uploaded, the system extracts the contents of the ZIP files, storing the extracted documents in a **processed** directory. The system ensures that only valid ZIP files are processed and handles any errors that might occur during the extraction.

- **Text Extraction from PDF:**
    - The system reads and extracts text from the PDFs, storing the results in plain text format for further processing. This allows the system to parse and analyze the content efficiently.

- **Chunking Method:** Initially, we experimented with several chunking methods, including semantic chunking and paragraph chunking. While these methods offered

valuable insights, they were not always consistent in preserving the natural flow and context of the text.

- Semantic Chunking: Grouped text based on meaning, aiming to maintain contextual relevance across chunks. However, it sometimes led to uneven chunk sizes and context overlaps.
- Paragraph Chunking: Split the text into chunks based on paragraph boundaries. Although this preserved the document structure, it resulted in chunks of varying lengths, which occasionally disrupted efficient retrieval.
- Ultimately, we adopted **sentence-aware chunking:**
  - Text is split into manageable chunks based on **natural sentence boundaries** to preserve the context and coherence of the text.
  - Ensures context preservation by using regex-based splitting techniques that avoid breaking sentences.
  - Each chunk is associated with metadata (e.g., document title, chunk index) for efficient retrieval and citation.

- **Embedding Generation:**
  - In this step, we convert the extracted text chunks into high-dimensional semantic vectors using a pre-trained transformer model: SentenceTransformer. These vectors represent the underlying meaning of the text, allowing us to process and understand it in a way that goes beyond just word matching.

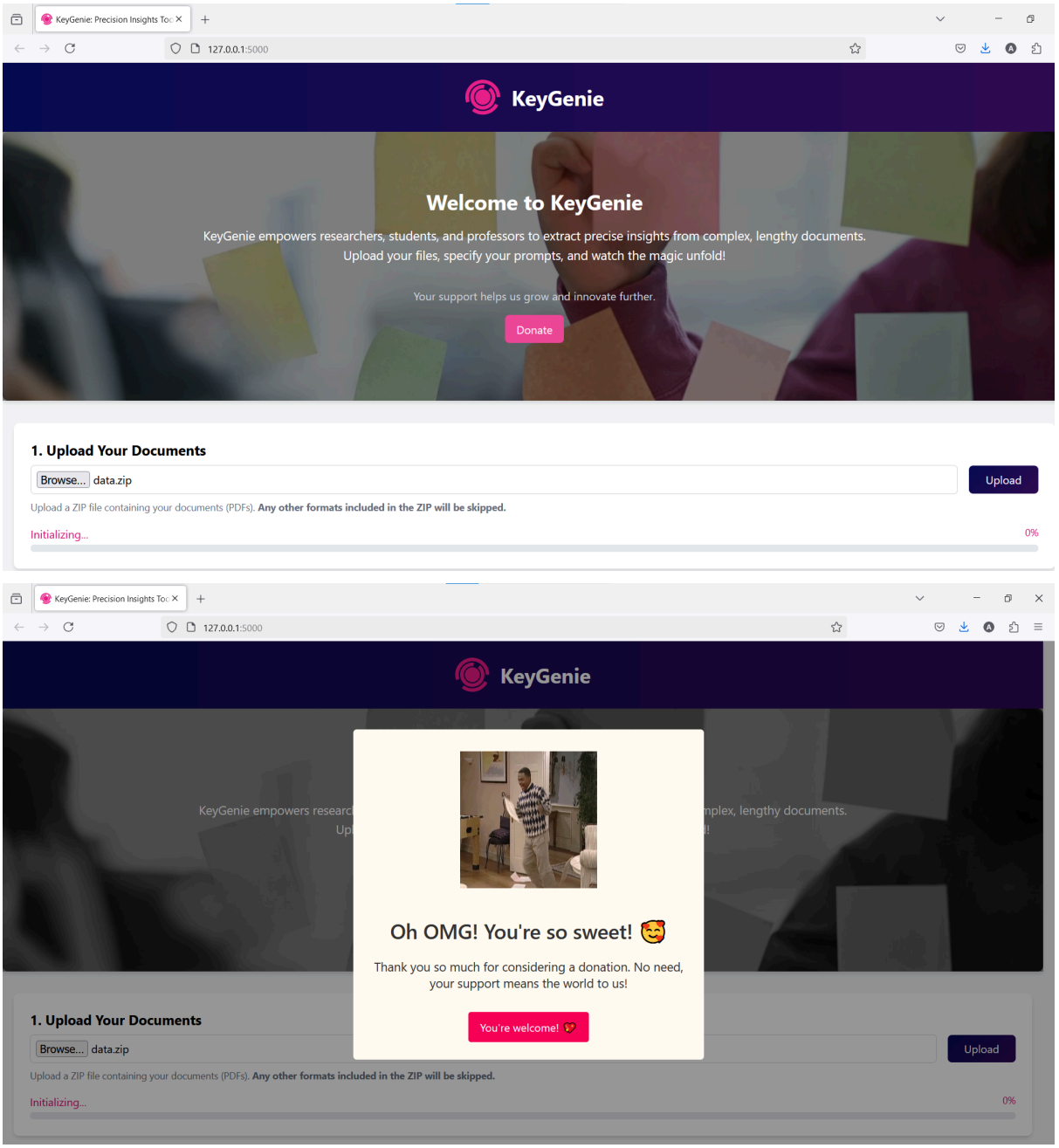
- **RAG Pipeline:**
  - The embeddings of the text chunks are indexed using **Pinecone**, a vector database that enables fast and scalable similarity search. Pinecone allows the system to query the indexed embeddings and retrieve the most relevant chunks based on the semantic similarity to the user's query.
  - When the user submits a query, it is converted into an embedding using the same Sentence Transformer model. This query embedding is then compared against the indexed document embeddings in Pinecone, and the system retrieves the most relevant chunks.
  - Once relevant chunks are retrieved, the system uses **Google's Generative AI model** to generate a detailed response to the user's query. The generative model combines the context from the retrieved chunks and generates a coherent and informative answer.

- **Output Delivery:**
  - The final step in the process is to deliver the response to the user. The system formats the answer in a clear and structured manner for easy interpretation.
  - **Structured Responses**: Each answer is presented with the user's query and the generated response, ensuring clarity and ease of understanding. The

system organizes the information in a structured output format, which includes the user's question and the corresponding answer.

System Workflow

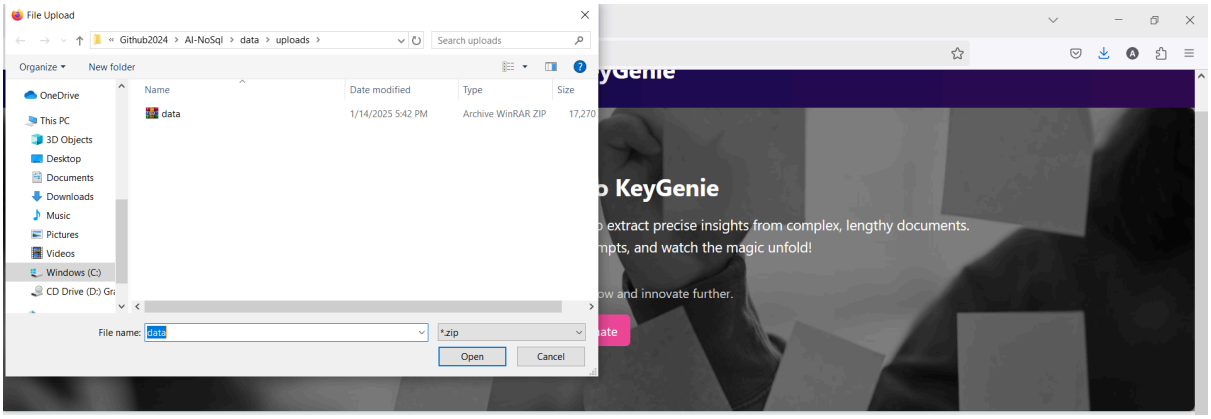
- A cute section to make the web app tool feel more realistic. When users click the "Donate" button, they are greeted with a fun and heartwarming popup, as shown below.



1. User Uploads ZIP Files

- **Action**: The user uploads a ZIP file containing multiple documents.
- **Input**: The user selects and uploads one or more ZIP files via the file dialog.
- **Process**: The system saves these ZIP files in the **uploads** directory.

- **Output**: A confirmation message is shown to the user once the upload is complete.

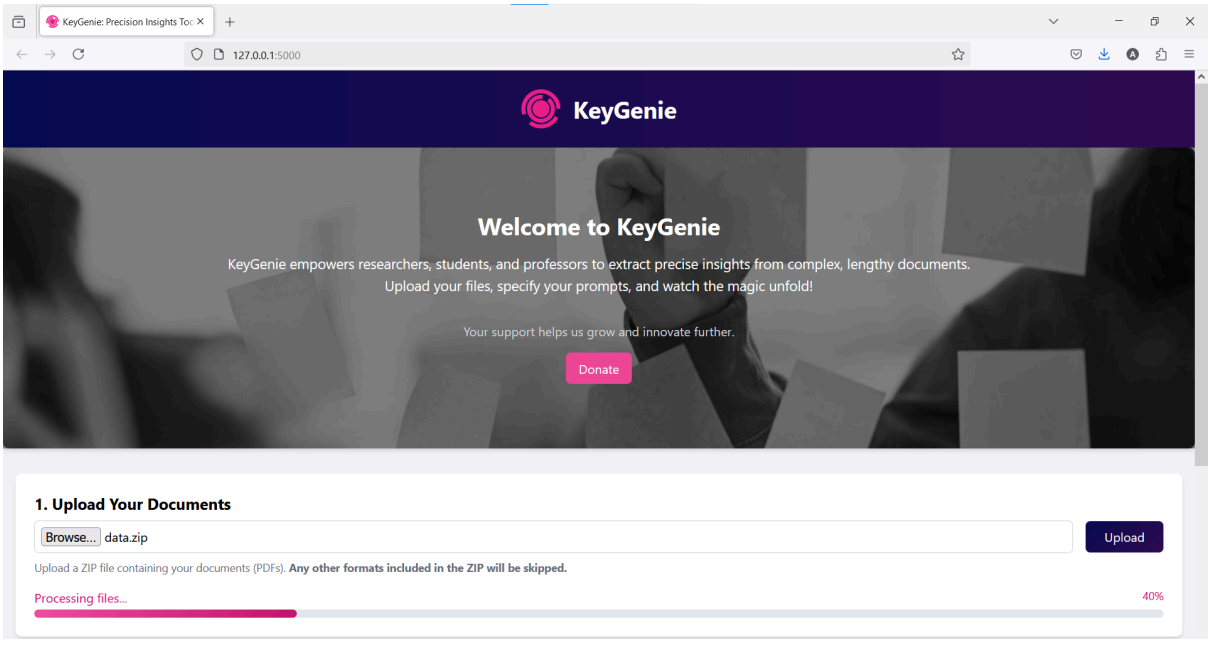


2. Extract Files from ZIP

- **Action**: The system extracts the contents of the uploaded ZIP file.
- **Input**: The ZIP files uploaded by the user.
- **Process**:
    - The system checks if the uploaded files are in ZIP format.
    - The contents of the ZIP file are extracted into the **processed** directory, creating a separate folder for each extracted document.
- **Output**: Extracted documents are stored in the **processed** directory.

3. Text Extraction from PDF Documents

- **Action**: The system extracts text from each PDF document in the processed directory.
- **Input**: The extracted PDF files.
- **Process**: The system reads the PDFs and extracts the text content, storing it in plain text format for further processing.
- **Output**: A collection of text files containing the extracted text from the PDFs.

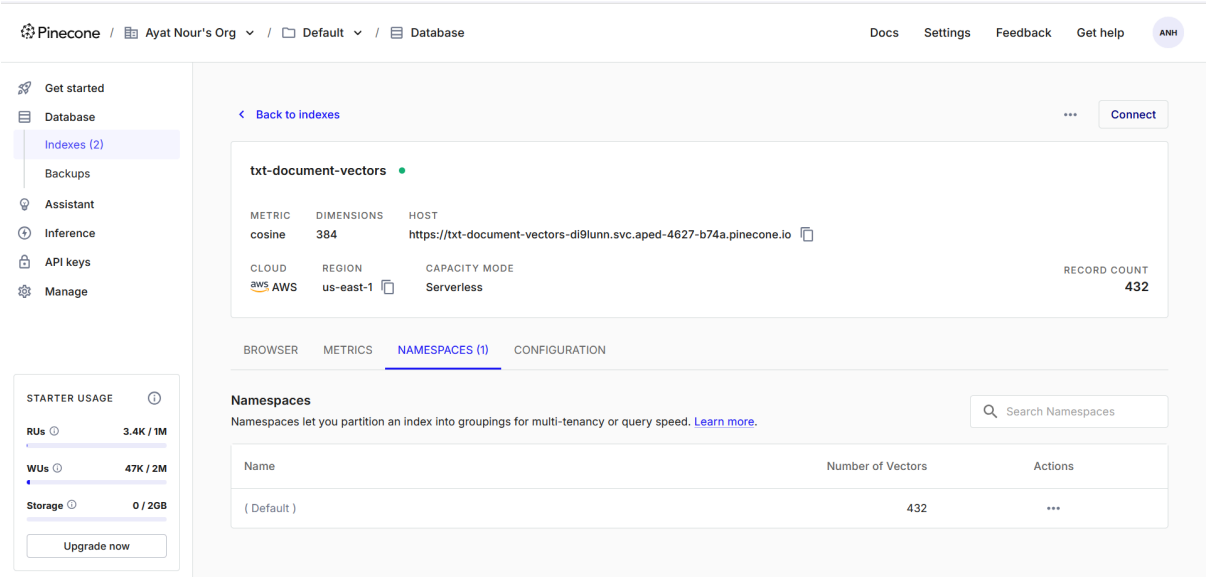


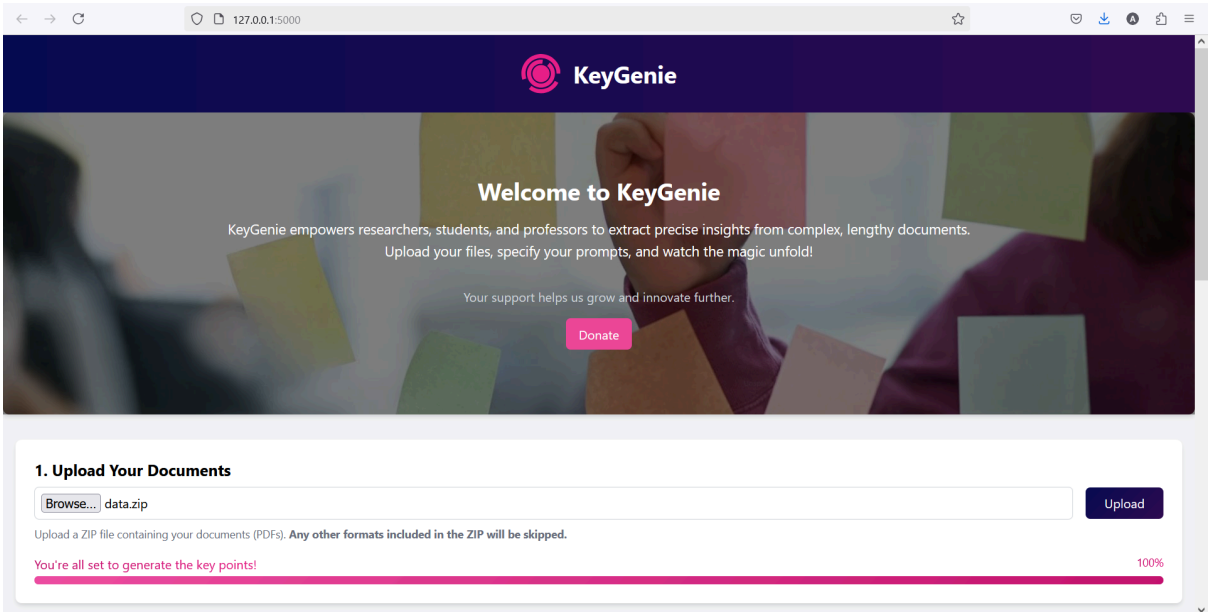
4. Sentence-Aware Chunking

- **Action**: The extracted text is divided into smaller, manageable chunks.
- **Input**: The text content extracted from each PDF document.
- **Process**:
    - The text is split using **sentence-aware chunking**, ensuring that chunks are created based on natural sentence boundaries.
    - Each chunk is treated as a distinct piece of content, preserving its meaning and context.
- **Output**: A set of text chunks, each associated with metadata such as document title and chunk index.

5. Embedding Generation

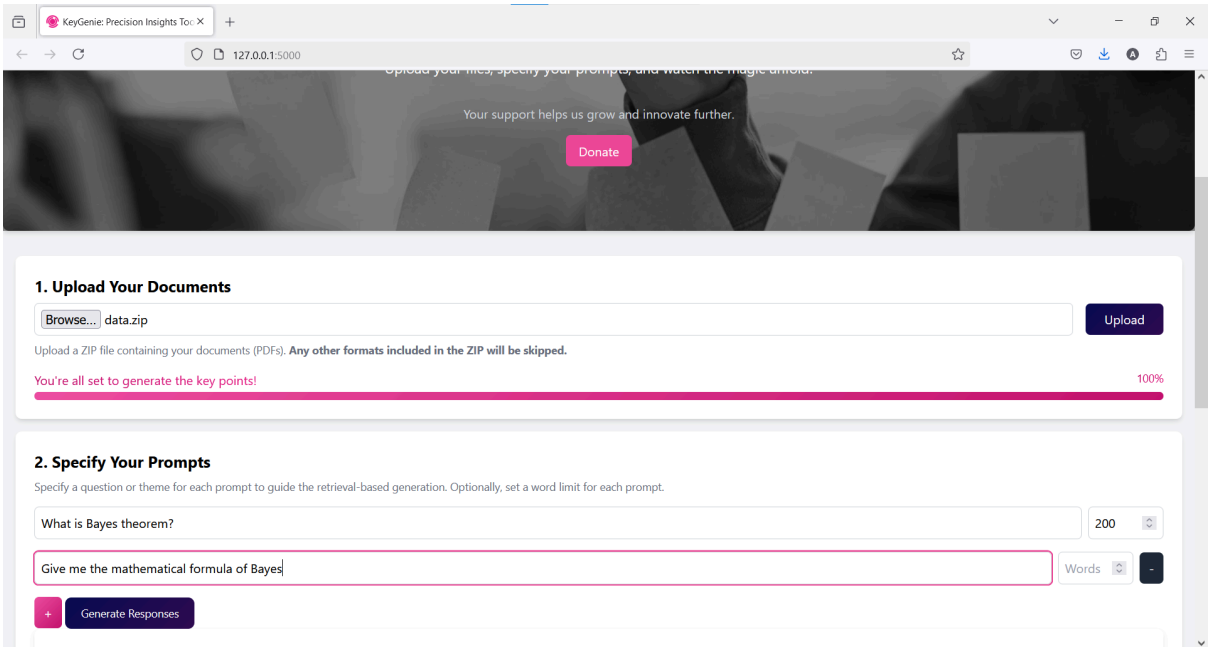
- **Action**: The chunks of text are converted into high-dimensional embeddings.
- **Input**: The text chunks from the previous step.
- **Process**:
    - The system uses the **SentenceTransformer** model (`sentence-transformers/all-MiniLM-L6-v2`) to generate embeddings for each chunk.
    - The embeddings are normalized to unit vectors to ensure they can be accurately compared.
- **Output**: A collection of embeddings for each chunk, stored in an index using **Pinecone** for efficient retrieval.





6. User Query Submission

- **Action**: The user submits a query.
- **Input**: The user provides one or more key points (questions) they want answers for.
- **Process**:
    - The system converts the user's query into an embedding using the **SentenceTransformer** model, just like the text chunks.
- **Output**: The query embedding ready for similarity comparison.



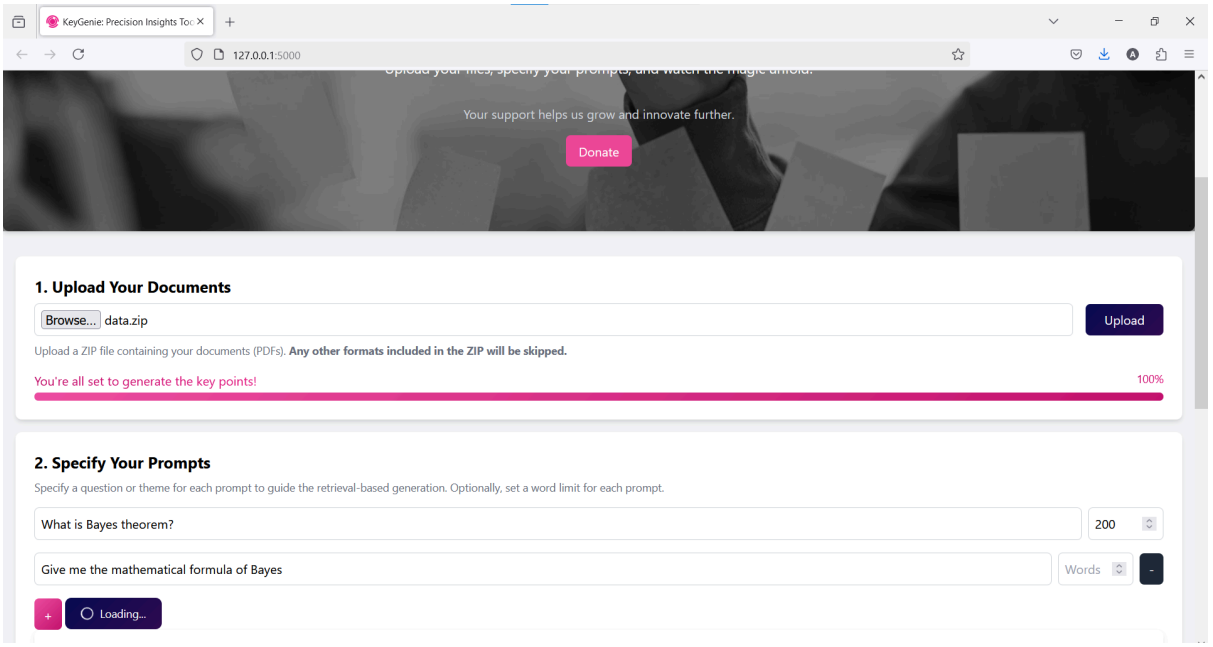
7. Retrieval of Relevant Chunks (RAG Pipeline)

- **Action**: The system retrieves the most relevant text chunks based on the user's query.

- **Input**: The query embedding generated in the previous step.
- **Process**:
    - The query embedding is compared against the pre-indexed document embeddings in **Pinecone** using cosine similarity.
    - The system retrieves the top-k most relevant chunks based on the similarity to the user's query.
- **Output**: A set of relevant chunks that contain information related to the user's query.

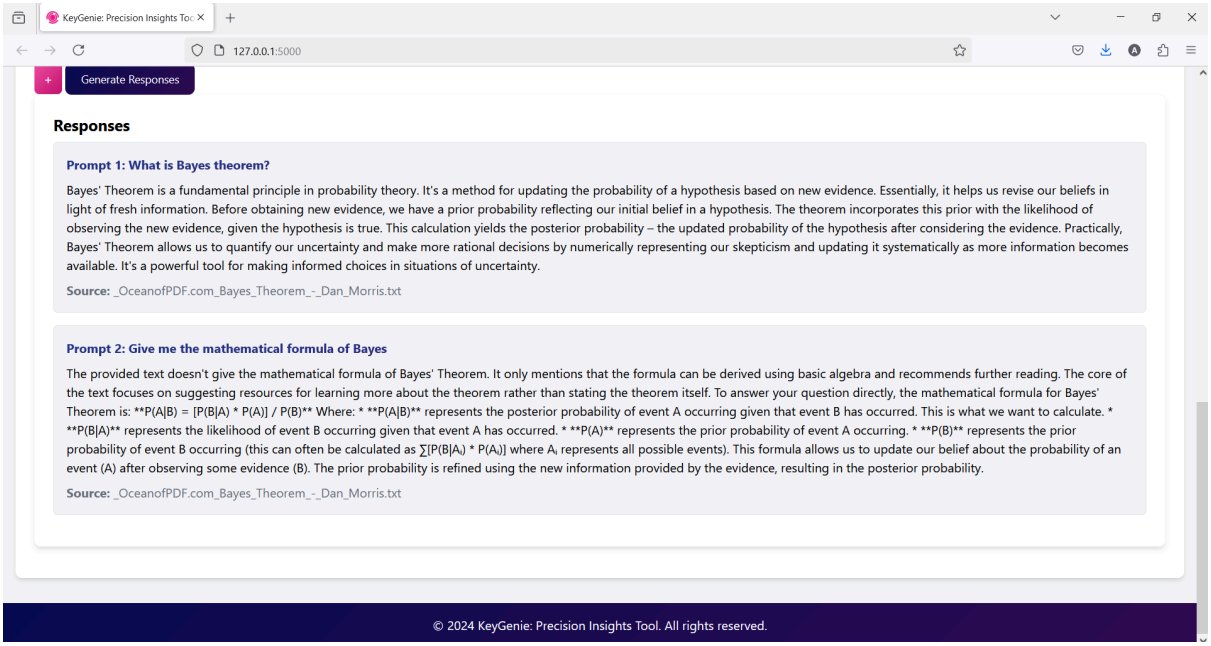
8. Answer Generation Using Google Generative AI

- **Action**: The system generates a response based on the retrieved chunks.
- **Input**: The relevant chunks of text and the user's query.
- **Process**:
    - The system uses the **Google Generative AI model** to generate a detailed response by incorporating the relevant context from the retrieved chunks.
    - The generative model combines the query and context to produce a coherent and contextually accurate answer.
- **Output**: The generated response to the user's query.



9. Output Delivery

- **Action**: The system formats and presents the answer to the user.
- **Input**: The generated response.
- **Process**:
    - The answer is formatted into a structured output.
    - The system displays the user's query along with the generated answer in a clear and readable format.
- **Output**: A final output delivered to the user, containing the query and the corresponding generated answer.



Design and Architecture

Backend Integration:

- **Database Design:**
  - A vector database (e.g., Pinecone) is used to store embeddings temporarily for efficient similarity searches.
  - Metadata is linked with each chunk for easy traceability.
- **Backend Workflow:**
  - The backend, built with Flask, manages file uploads, embedding processing, and query handling.
  - Stateless design ensures temporary data handling and deletion post-summary generation.

Frontend Development:

- **User Interface:**
  - Developed using HTML, TailwindCSS, JavaScript, the frontend provides:
    - A drag-and-drop interface for file uploads.
    - Input fields for specifying questions and key points.
    - Real-time previews of generated answers.
- **Usability:**
  - Intuitive design ensures users can easily navigate the tool to upload documents, define questions, and download results.

Challenges Faced and Solutions

1. **Efficient Chunking of Large Documents:**
    - **Challenge:** Dividing large texts into chunks without losing context or relevance.
    - **Solution:** Implemented sentence-aware chunking to preserve sentence integrity and associate metadata for efficient retrieval.
2. **Embedding Performance:**
    - **Challenge:** Generating high-quality embeddings for large datasets efficiently.
    - **Solution:** Utilized optimized pretrained transformer models (e.g., SentenceTransformer) for fast and accurate vectorization.
3. **Data Privacy:**
    - **Challenge:** Handling sensitive user documents securely.
    - **Solution:** Adopted a stateless backend design with automatic deletion of temporary files and embeddings post-processing.

Conclusion :

The **Text Generation and Key Point Extraction Tool** effectively bridges the gap between processing large documents and delivering precise answers to specific questions. By integrating advanced AI techniques with a user-centric design, it offers:

- **Targeted Answers:** Provides accurate, query-specific responses.
- **Transparency and Credibility:** Includes inline citations to maintain source credibility for every generated response.
- **Flexibility and Efficiency:** Customizes output to meet diverse user needs while ensuring efficient processing.

Links :

Pitch

Github Repository