**Document Summarization using Retrieval-Augmented Generation (RAG)**

This project uses a Retrieval-Augmented Generation (RAG) approach, which enhances a language model’s summarization capabilities by first retrieving only the most relevant content using semantic similarity.

The pipeline consists of the following core components:

* Text Chunking (Spacy-based sentence splitting)
* Chunk Embedding (Sentence-BERT model)
* Similarity Indexing (FAISS)
* Top-k Semantic Retrieval
* Summarization using a transformer model (facebook/bart-large-cnn)

**1.** **Chunking Text for Retrieval**

**chunk\_article(article, chunk\_size=5, overlap=2)**

* **Purpose:** Breaks a single article (text) into overlapping chunks of sentences to preserve contextual flow.
* **How it works:**
  + Uses **spaCy** to split the article into **sentences**.
  + Each chunk contains chunk\_size sentences.
  + An overlap of overlap ensures continuity between adjacent chunks (important for preserving meaning).
  + Example:
    - Sentences: [S1, S2, S3, S4, S5, S6, S7]
    - Chunks with chunk\_size=3, overlap=1:  
      [S1 S2 S3], [S3 S4 S5], [S5 S6 S7]
* **Purpose:** Applies chunk\_article() to a **list of articles**, aggregating all their chunks into one list.

**2. Generating Semantic Embeddings**

**get\_embedding(text)**

* **Purpose:** Converts a single text into its vector embedding using SentenceTransformer.
* **Note:** This is a helper function not used in the main code but useful for debugging or expansion.

**embed\_chunks(chunks)**

* **Purpose:** Converts all chunks into embeddings using **all-MiniLM-L6-v2.**
* **Details:**
  + convert\_to\_numpy=True returns NumPy arrays for compatibility with FAISS.
  + Returned as float32 to match FAISS format requirements.
* **Model Used:** all-MiniLM-L6-v2 from Sentence Transformers  
  A small, fast, and reasonably accurate model for semantic similarity.

**3. Indexing for Fast Similarity Search**

**build\_faiss\_index(embeddings)**

* **Purpose:** Constructs a FAISS index for **efficient similarity search**.
* **Details:**
  + Uses **L2 distance** (Euclidean distance).
  + Index type: IndexFlatL2 (no compression, good for small to medium datasets).
  + Embeddings are added once and can be searched in milliseconds.
* **Why FAISS?**  
  Facebook AI Similarity Search (FAISS) enables ultra-fast nearest neighbor lookups, ideal for semantic retrieval.

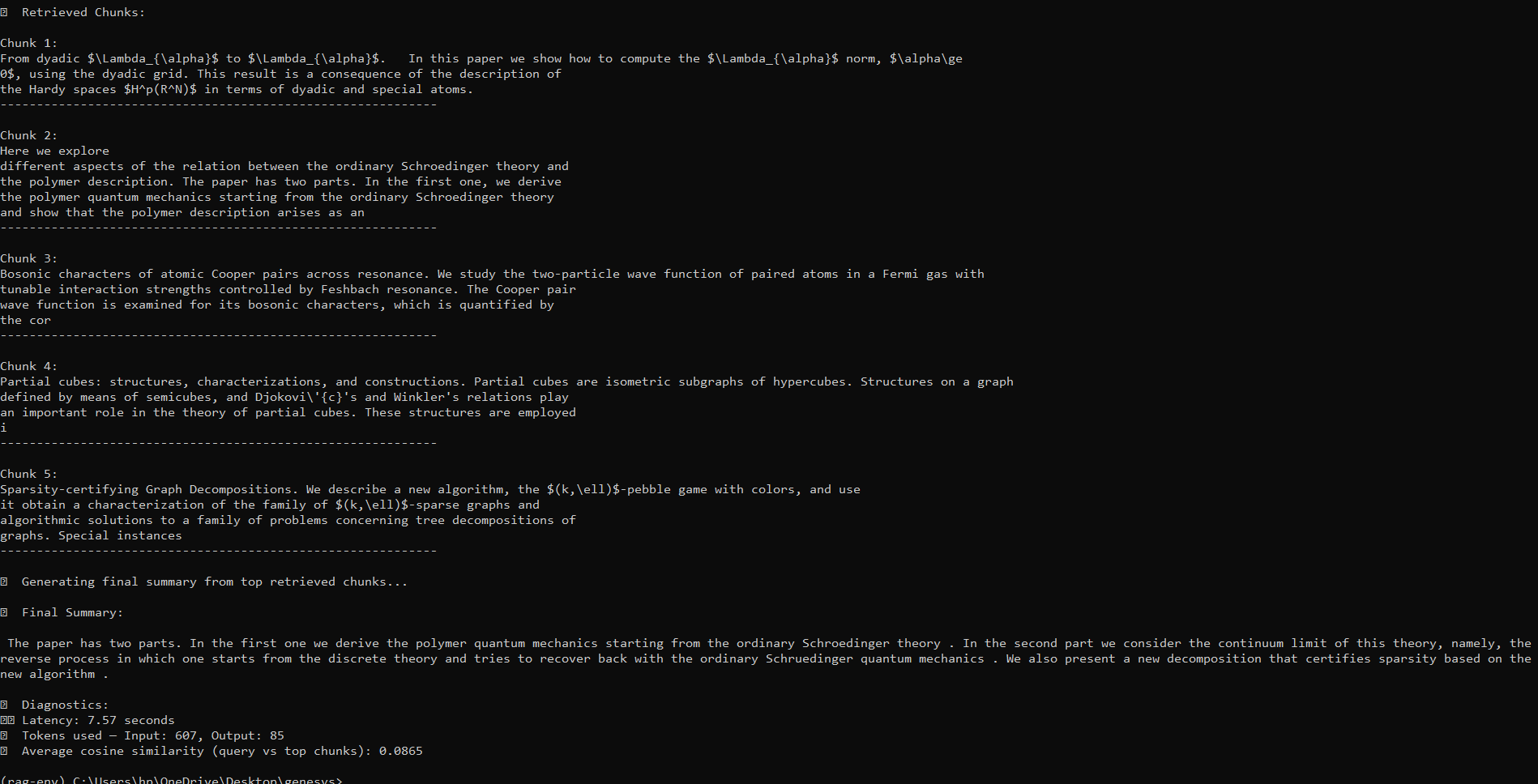
**4. Retrieving Most Relevant Chunks**

**retrieve\_top\_k\_chunks(query, k, index, chunks)**

* **Purpose:** Finds the top k chunks most similar to a given query.
* **How it works:**
  + Embeds the query using the same sentence transformer model.
  + Searches the FAISS index to retrieve the closest k embeddings.
  + Uses returned indices to select the corresponding chunks.
* **Result:** These top k chunks are the **context** fed to the language model for summarization.

**HOW IT FITS TOGETHER (In main\_pdf.py)**

1. **PDF → Raw Text:** Loaded and extracted using PyMuPDF.
2. **Text → Chunks:** Sentences grouped with overlap for better context preservation.
3. **Chunks → Embeddings:** Each chunk is embedded semantically.
4. **Embeddings → FAISS Index:** Indexed for rapid similarity search.
5. **Query → Top-k Chunks:** Retrieves most relevant chunks to the prompt: "Summarize this document".
6. **Chunks → Summary:** Passed to BART transformer for final summarization.

**OUTPUTS:**

A computer screen with text

AI-generated content may be incorrect.

A computer screen with text on it

AI-generated content may be incorrect.