**Doc-Scanner**

**🧩 Key Components Involved**

Let’s first remember what’s involved:

* ✅ **Flask App** – User interface & file upload
* 📄 **Uploaded File** – Document to review
* ✂️ **LangChain** – Splits text into small parts
* 🧮 **Google Embedding-001** – Turns each part into numbers
* 🗃️ **ChromaDB** – Stores those numbers + chunks
* 🔍 **Retriever** – Finds similar past content
* 🤖 **Gemini LLM** – Gives suggestions based on that
* 🧠 **GeminiAISuggestionEngine** – Your custom logic for all this

**🔁 Sequence: What Happens When You Upload a File**

**1️⃣ You Upload a Document**

* You drag and drop or choose a document (PDF, DOCX, etc.)
* Flask receives the file

**2️⃣ Text is Extracted**

* The app reads the document and **extracts plain text**
  + python-docx, PyPDF2, or BeautifulSoup might be used here depending on file type

**3️⃣ Text is Split into Chunks**

* LangChain **splits** the text into small parts (sentences or paragraphs)
  + This is so the AI can analyze and compare small meaningful chunks
  + It may use RecursiveCharacterTextSplitter or SentenceSplitter

**4️⃣ Embeddings Are Generated**

* Each chunk is passed to **Google's Embedding-001 model**
* It converts each chunk into a list of numbers (vector) that **represents its meaning**

**5️⃣ Embeddings + Chunks Are Stored in ChromaDB**

* ChromaDB saves:
  + The chunk (text)
  + The embedding (vector)
  + Metadata (like filename or section)

This makes your document searchable for future comparisons.

**6️⃣ RAG Starts: Review + Suggestion Begins**

* When reviewing a sentence:
  + A **retriever** checks ChromaDB for similar content
  + These "similar chunks" are passed to Gemini 1.5 Flash

**7️⃣ Gemini LLM Gives Suggestions**

* Gemini sees:
  + The sentence to review
  + Similar examples from your document (retrieved from ChromaDB)
* It gives a **smart, context-aware suggestion** or comment

**8️⃣ Custom Logic Wraps It Up**

* Your GeminiAISuggestionEngine class combines all the parts:
  + Formatting
  + Style rule checks
  + Suggestion formatting
  + HTML rendering for the UI

**9️⃣ Flask App Displays the Feedback**

* The user sees:
  + Original sentence
  + Suggestion from Gemini
  + Explanation (if any)
  + Option to accept, ignore, or revise

**✅ Bonus: If You Upload More Files**

* More files → More data chunks + embeddings → ChromaDB grows
* Now Gemini can pull smarter suggestions using a **larger memory of your writing style**
* Your app keeps getting "contextually smarter" over time!

**🔁 Visual Summary (Simple Flow)**

pgsql

CopyEdit

User uploads file

↓

Text is extracted

↓

Text is chunked

↓

Embeddings are generated

↓

Chunks + embeddings stored in ChromaDB

↓

Retriever finds similar content

↓

Gemini generates review/suggestions

↓

Flask shows them on the UI