Absolutely! Below is a **complete, step-by-step blueprint** to build your full-fledged project that:

* Loads multiple assignments
* Tokenizes or chunks them
* Converts them into **TF-IDF** and **semantic embeddings**
* Stores them in a **vector store**
* Compares similarities across documents
* Clusters similar assignments
* Outputs grouped results like: “These X assignments are very similar (possibly copied).”
* Uses **LangChain**, but without heavy dependency on paid APIs

**🎓 PROJECT GOAL:**

**Build an intelligent system** that takes multiple assignment documents as input and **clusters or flags similar ones**, using both **semantic meaning (embeddings)** and **word-based statistics (TF-IDF)**.

**🔧 STEP-BY-STEP IMPLEMENTATION**

**✅ 1. Document Input (Assignment Collection)**

* Use **LangChain's DirectoryLoader** or UnstructuredFileLoader to load multiple assignment files (PDF, Word, TXT, etc.).
* Extract the **filename or document ID** as metadata so you can later refer to specific assignments.

🎯 Output: List of Document objects with text + metadata.

**✅ 2. Splitting the Document Text (Chunking)**

* Use **LangChain's RecursiveCharacterTextSplitter** or TokenTextSplitter to break long assignment documents into **chunks** (e.g., 500–1000 characters).
* This is important because long documents must be broken down for vector processing.

🎯 Output: Smaller chunks with original metadata preserved.

**✅ 3. Optional Preprocessing (Tokenization & Cleaning)**

You can apply **SpaCy** or **NLTK** for:

* Lowercasing
* Removing stopwords
* Lemmatization or stemming
* Removing punctuation or special characters

🎯 This step is helpful for improving TF-IDF quality. You can skip this for embedding models which already understand raw text context.

**✅ 4. Create TF-IDF Vectors (Bag-of-Words Representation)**

* Use **Scikit-learn’s TfidfVectorizer** to transform chunks (or whole documents) into **TF-IDF vectors**.
* You can choose:
  + Either chunk-level TF-IDF
  + Or whole-document TF-IDF (after merging the chunks back)

🎯 Output: TF-IDF matrix (documents × terms)

**✅ 5. Create Embeddings (Semantic Representation)**

* Use a **free, local embedding model** (to avoid OpenAI API usage). Examples:
  + sentence-transformers → like all-MiniLM-L6-v2
  + Hugging Face transformers like BAAI/bge-small-en
* Generate an embedding vector **per chunk** or **per document**.

🎯 Output: Dense embedding matrix (chunks × 384/512 dimensions)

**✅ 6. Store Vectors in Vector Store (FAISS or Chroma)**

* You now have **two sets of vectors**:
  1. TF-IDF (sparse)
  2. Embeddings (dense)
* You can create:
  1. One **FAISS store** for embeddings
  2. One **manual structure** (like NumPy arrays or Pandas DataFrame) for TF-IDF since FAISS doesn’t support sparse vectors natively.

🎯 Output: Stored vectors for fast similarity search

**✅ 7. Similarity Calculation**

**🔸 A. For Embeddings:**

* Use **Cosine Similarity** across the dense embedding vectors (can be per chunk or averaged per document).
* If chunk-level, average the vectors to get a **single vector per document**.

**🔸 B. For TF-IDF:**

* Also use **Cosine Similarity** across the TF-IDF vectors (Scikit-learn’s cosine\_similarity does this easily).

**✅ 8. Hybrid Similarity Scoring (TF-IDF + Embedding)**

* For every document pair:
  + Compute **TF-IDF similarity score**
  + Compute **Embedding similarity score**
  + Combine them (e.g., weighted average)

Hybrid Score=0.5×TF-IDF Score+0.5×Embedding Score\text{Hybrid Score} = 0.5 \times \text{TF-IDF Score} + 0.5 \times \text{Embedding Score}

🎯 Output: A similarity matrix between all documents

**✅ 9. Clustering Similar Documents**

* Use **unsupervised clustering** on hybrid similarity scores:
  + **Agglomerative Clustering** (good for text)
  + **DBSCAN** (good for finding groups with outliers)
  + **KMeans** (if you want fixed number of groups)

🎯 Output: Cluster IDs assigned to each document (e.g., these 5 are in cluster 1, next 3 in cluster 2)

**✅ 10. Group and Report Similar Assignments**

* Group documents by cluster ID
* Report which assignments are **strongly similar (same cluster)**
* Optional:
  + Provide **similarity score**
  + Show **percentage overlap**
  + Show **top matching chunks**

🎯 Final output: Clear grouping of similar/copy-like assignments by their file names or IDs.

**✅ 11. Bonus: LangChain Chat Integration (Optional)**

* If you still want a chat interface:
  + Use ChatOpenAI or any LLMChain to let user ask:  
    “Show me which assignments are similar to Assignment\_7.docx”
  + Use the vector store to find nearest documents (RAG style)
  + Generate a textual summary of similarity using a PromptTemplate

🎯 This is optional, but great for interactive use.

**✅ 12. Evaluation (Check Results Are Good)**

* Compare clusters manually (read a few assignments in same cluster)
* Tune:
  + Chunk size
  + TF-IDF parameters (n-grams, stopwords)
  + Embedding model
  + Weights in hybrid score
  + Clustering method

**🧠 Summary of the Full Project Workflow**

| **Stage** | **Tool / Method** |
| --- | --- |
| Load documents | LangChain DirectoryLoader |
| Split documents | LangChain RecursiveTextSplitter |
| Clean (optional) | SpaCy / NLTK |
| TF-IDF Vectors | Scikit-learn |
| Embedding Vectors | SentenceTransformers (Free) |
| Store Vectors | FAISS or Chroma |
| Compare Similarity | Cosine similarity |
| Combine Similarities | Hybrid Scoring |
| Cluster Similar Docs | KMeans / DBSCAN / Agglomerative |
| Output Similar Groups | Group by cluster + show names |
| (Optional) Chat interface | LangChain + PromptTemplate |

**✅ Why this is powerful:**

* Doesn’t rely heavily on paid APIs
* Uses both **shallow (TF-IDF)** and **deep (embeddings)** representations
* Clustering gives clean groupings
* Expandable to chat, RAG, dashboards

Would you like:

* A visual flowchart?
* Or a folder structure suggestion for organizing this project?  
  Let me know and I’ll prepare that too!