**🧱 Obiettivo del progetto**

Realizzare un sistema **Retrieval-Augmented Generation** (RAG) che:

* Usa un **LLM** per generare risposte intelligenti,
* Si basa su documenti **scientifici** distribuiti tramite **HDFS**,
* Applica **preprocessing distribuito** (con **Ray** o **Spark**),
* Esegue **embedding** del contenuto,
* Indicizza i vettori in un **vector database**,
* Recupera contenuti rilevanti e li integra nella risposta del LLM.

**🔧 Tecnologie consigliate (modulari e moderne)**

| **Fase** | **Tecnologia/i consigliate** |
| --- | --- |
| Storage distribuito | **HDFS** (Hadoop Distributed File System) |
| Preprocessing distribuito | **Ray** (alternativa: Spark se già installato) |
| Estrazione e cleaning | **Python**, **pandas**, **pdfminer / BeautifulSoup** |
| Embedding testuale | **SentenceTransformers** (es. all-MiniLM-L6-v2) |
| Vector Database | **FAISS** (locale) o **Qdrant** (via Docker o cloud) |
| LLM (Generazione) | **OpenAI GPT-3.5**, **LLaMA**, **Mistral**, via API o locale |
| RAG pipeline | **LangChain** o **Haystack** (per orchestrare RAG) |
| API / demo | **FastAPI** o **Flask** |
| Optional: orchestrazione | **Docker** (per FAISS/Qdrant + backend) |

**📅 Fasi di sviluppo passo-passo**

**🟩 FASE 0: Setup ambienti e dataset**

1. **Scarica il dataset** arXiv da Kaggle.
2. **Caricalo in HDFS**:
   * Se usi VM: metti i file in /user/tuo\_utente/ su HDFS.
   * Se sei in locale: simula HDFS o usa un file system strutturato.

**🟨 FASE 1: Preprocessing distribuito**

1. **Scegli Ray (consigliato)** o Spark.
2. Estrai campi utili (id, title, abstract, categories).
3. Pulisci gli abstract:
   * Rimuovi caratteri speciali, simboli, link.
   * Fai chunking: spezza il testo in blocchi di 2-3 frasi.
4. **Salva i chunk** preprocessati in JSON o Parquet, pronti per l’embedding.

🔧 *Tecnologie*: Python + Ray (per parallelizzare extract\_and\_clean())

python

CopiaModifica

@ray.remote

def extract\_and\_clean(doc):

# Pulisci e restituisci lista di chunk testuali

**🟧 FASE 2: Embedding e indicizzazione**

1. Carica i chunk di testo pulito.
2. Usa sentence-transformers per calcolare embedding:

python

CopiaModifica

from sentence\_transformers import SentenceTransformer

model = SentenceTransformer('all-MiniLM-L6-v2')

embedding = model.encode(text\_chunk)

1. Inserisci gli embedding nel vector DB:
   * **FAISS**: veloce, in-memory, perfetto per demo accademica.
   * **Qdrant** o **Weaviate**: offrono REST API, metadati, indexing persistente.
2. Aggiungi metadati (id, titolo, categoria).

**🟦 FASE 3: Retrieval e sistema RAG**

1. Implementa una funzione di retrieval:
   * Dato un input utente → calcola embedding → cerca top-k simili.
2. Passa i documenti recuperati come **contesto** al LLM.
3. Usa **LangChain** o **Haystack** per orchestrare:
   * Query → Retrieve → Prompt → Generate.

Esempio LangChain:

python

CopiaModifica

retriever = FAISS.load\_local("vector\_db").as\_retriever()

rag\_chain = RetrievalQA.from\_chain\_type(llm=OpenAI(), retriever=retriever)

**🟪 FASE 4: Interfaccia API o demo**

1. Crea un’interfaccia **FastAPI** per interagire con il sistema:
   * Input: una domanda.
   * Output: risposta generata + documenti di supporto.
2. Esempio endpoint:

python

CopiaModifica

@app.post("/ask")

def ask\_question(query: str):

result = rag\_chain.run(query)

return {"risposta": result}

**🟥 FASE 5: Valutazione finale**

1. Crea un set di 5–10 domande accademiche.
2. Fai rispondere il **LLM puro** e il **LLM con RAG**.
3. Valuta con griglia qualitativa:
   * Accuratezza, rilevanza, citazioni corrette, completezza.
4. (Facoltativo) Usa anche BERTScore o ROUGE se hai risposte attese.