

🟢 Hands-on Lab: Build a Hybrid RAG Pipeline using OpenAI + FAISS + RDF Graph + Pinecone

🎯 Objective:

Combine **semantic vector retrieval (OpenAI + FAISS)** and **structured graph-based retrieval (RDF + SPARQL or Pinecone filters)** to build a **hybrid Retrieval-Augmented Generation (RAG)** pipeline that enhances accuracy and context-awareness.

🧩 Part 1: Set Up Your Environment

🔧 Requirements

- Python 3.9+
- Packages: `openai`, `faiss-cpu`, `rdflib`, `pinecone-client`, `langchain`, `datasets`, `tqdm`, `dotenv`
- Accounts:
 - [OpenAI API key](#)
 - [Pinecone account](#)
- Optional: [Jupyter Notebook](#) or VSCode for convenience

📦 Install Dependencies

```
pip install openai faiss-cpu rdflib pinecone-client langchain datasets tqdm python-dotenv
```

🧠 Part 2: Prepare Knowledge Sources

📁 2.1 Load Sample Knowledge Data

Use a sample domain like **healthcare policies** or **financial FAQs**.

```
from datasets import load_dataset

dataset = load_dataset("wiki_snippets", "en", split="train[:500]")
docs = [d["passage_text"] for d in dataset if d["passage_text"]]
```

📊 2.2 Build RDF Knowledge Graph (Lightweight)

Use `rdflib` to create triples like:

```
from rdflib import Graph, URIRef, Literal, Namespace, RDF
```

```
g = Graph()
EX = Namespace("http://example.org/")

g.add((EX["Doc1"], RDF.type, EX["Policy"]))
g.add((EX["Doc1"], EX["category"], Literal("Healthcare")))
g.add((EX["Doc1"], EX["region"], Literal("US")))
g.add((EX["Doc1"], EX["text"], Literal(docs[0])))
```

Save it to disk:

```
g.serialize(destination="knowledge.ttl", format="turtle")
```

Part 3: Vector Indexing with FAISS

3.1 Embed Documents

```
from openai import OpenAI
from openai.embeddings_utils import get_embedding
import numpy as np

openai.api_key = 'YOUR_API_KEY'

embeddings = [get_embedding(d, engine="text-embedding-3-small")
for d in docs]
```

3.2 Create FAISS Index

```
import faiss

index = faiss.IndexFlatL2(1536)
index.add(np.array(embeddings).astype('float32'))
```

Part 4: Metadata-Aware Retrieval with Pinecone

4.1 Set Up Pinecone

```
import pinecone

pinecone.init(api_key="YOUR_PINECONE_KEY", environment="us-west1-gcp")

index_name = "rag-hybrid-demo"
pinecone.create_index(index_name, dimension=1536, metric="cosine",
metadata_config={"indexed": ["category", "region"]})
pc_index = pinecone.Index(index_name)
```

4.2 Upload Chunks + Metadata

```
pc_index.upsert([
    {
        "id": f"doc{i}",
        "values": embeddings[i],
        "metadata": {"category": "Healthcare", "region": "US",
        "text": docs[i]}
    }
    for i in range(len(docs[:100]))
])
```

Part 5: Hybrid Query Pipeline

5.1 User Query

```
query = "What are the healthcare benefits in the US?"
query_embedding = get_embedding(query, engine="text-embedding-3-small")
```

5.2 Vector Search (FAISS)

```
D, I = index.search(np.array([query_embedding]).astype('float32'), k=3)
top_faiss_docs = [docs[i] for i in I[0]]
```

5.3 Pinecone Filtered Search

```
pinecone_results = pc_index.query(
    vector=query_embedding,
    top_k=3,
    include_metadata=True,
    filter={"category": {"$eq": "Healthcare"}, "region": {"$eq": "US"}}
)
top_pinecone_docs = [match["metadata"]["text"] for match in
pinecone_results["matches"]]
```

5.4 Graph Filtering via SPARQL (optional)

```
query = """
PREFIX ex: <http://example.org/>
SELECT ?text WHERE {
    ?doc ex:category "Healthcare" ;
        ex:region "US" ;
        ex:text ?text .
}
"""

results = g.query(query)
graph_texts = [str(r["text"]) for r in results]
```

🧠 5.5 Compose Final Prompt to LLM

```
context = "\n".join(top_faiss_docs + top_pinecone_docs + graph_texts[:1])
final_prompt = f"Answer based on the context below:\n\n{context}\n\nQ:
{query}\nA:"
```

✍️ Part 6: Generate Response with OpenAI

```
response = openai.ChatCompletion.create(
    model="gpt-4",
    messages=[{"role": "user", "content": final_prompt}]
)
print(response["choices"][0]["message"]["content"])
```

✅ Wrap-Up: Key Takeaways

- You built a **hybrid RAG pipeline** integrating:
 - FAISS (semantic vector search)
 - RDF Graph (structured filtering)
 - Pinecone (metadata filtering)
 - Your prompt included both **structured and semantic context**
 - This architecture is **modular, explainable, and scalable**
-