Below is the **full notebook content** (all cells) you can copy into a new Jupyter notebook (.ipynb) or into cells in an existing notebook. It matches the pipeline: **Load PDF → Split → Embed → Build FAISS → Save → Reload → RetrievalQA** using the GroqRemoteLLM wrapper.

**Cell 1 — Markdown (Introduction)**

# RetrievalQA pipeline (PDF → chunks → embeddings → FAISS → save/load → QA)

\*\*What this notebook does (end-to-end):\*\*

1. Load a local PDF (`sample.pdf`).

2. Split into chunks.

3. Create embeddings (Sentence-Transformers `all-MiniLM-L6-v2`).

4. Build FAISS vectorstore and \*\*save it locally\*\*.

5. Reload the FAISS index (demonstrates fast startup).

6. Create a RetrievalQA chain using a remote Groq LLM wrapper (if available) and run queries.

\*\*Before running:\*\*

- Put a PDF named `sample.pdf` in the same folder as this notebook, or change the `PDF\_PATH` variable below.

- Make sure your environment has the required packages installed and the environment variables set (or use the inline `os.environ` cell to set them for the session):

GROQ\_API\_URL=<https://api.groq.com/openai/v1/chat/completions>  
GROQ\_API\_KEY=sk-<your\_key>  
GROQ\_MODEL=llama-3.3-70b-versatile

- Recommended packages: `langchain`, `sentence-transformers`, `faiss-cpu`, `python-dotenv`, `requests`.

This notebook is intended for development on a CPU-only machine and uses small embedding models to keep memory modest.

**Cell 2 — Code (Optional installs)**

# Optional: install required packages (uncomment to run)

# !pip install langchain sentence-transformers faiss-cpu python-dotenv requests PyPDF2

# If using Streamlit later:

# !pip install streamlit

print('Skip installation if packages already present.')

**Cell 3 — Code (Load .env and imports)**

import os

from dotenv import load\_dotenv

load\_dotenv()

print('GROQ\_API\_URL ->', os.getenv('GROQ\_API\_URL'))

print('GROQ\_MODEL ->', os.getenv('GROQ\_MODEL'))

print('GROQ\_API\_KEY present ->', bool(os.getenv('GROQ\_API\_KEY')))

# Core imports

from langchain.document\_loaders import PyPDFLoader

from langchain.text\_splitter import CharacterTextSplitter

from langchain.embeddings import SentenceTransformerEmbeddings

from langchain.vectorstores import FAISS

from langchain.chains import RetrievalQA

print('Imports ready')

**Cell 4 — Code (GroqRemoteLLM inline fallback)**

# GroqRemoteLLM inline fallback (will prefer to import groq\_remote\_llm.py if present)

try:

from groq\_remote\_llm import GroqRemoteLLM

print('Using groq\_remote\_llm.py from working directory')

except Exception:

print('groq\_remote\_llm.py not found or failed to import — using inline fallback (works for basic calls)')

from langchain.llms.base import LLM

from typing import Optional, List, Mapping, Any

from pydantic import Field

class GroqRemoteLLM(LLM):

api\_url: str = Field(default\_factory=lambda: os.getenv('GROQ\_API\_URL'))

api\_key: str = Field(default\_factory=lambda: os.getenv('GROQ\_API\_KEY'))

model: str = Field(default\_factory=lambda: os.getenv('GROQ\_MODEL', 'llama-3.3-70b-versatile'))

timeout: int = 60

@property

def \_llm\_type(self) -> str:

return 'groq-remote-llm'

@property

def \_identifying\_params(self) -> Mapping[str, Any]:

return {'model': self.model, 'url': self.api\_url}

def \_call(self, prompt: str, stop: Optional[List[str]] = None) -> str:

import requests

headers = {'Authorization': f'Bearer {self.api\_key}', 'Content-Type': 'application/json'}

payload = {

'model': self.model,

'messages': [

{'role': 'system', 'content': 'You are a helpful assistant.'},

{'role': 'user', 'content': prompt},

],

'max\_tokens': 256,

}

resp = requests.post(self.api\_url, json=payload, headers=headers, timeout=self.timeout)

resp.raise\_for\_status()

doc = resp.json()

try:

return doc['choices'][0]['message']['content']

except Exception:

return str(doc)

print('GroqRemoteLLM available as class')

**Cell 5 — Code (Config: load PDF, split, embed, vectorstore)**

# Config

PDF\_PATH = 'sample.pdf' # change if needed

CHUNK\_SIZE = 800

CHUNK\_OVERLAP = 120

EMBED\_MODEL = 'all-MiniLM-L6-v2'

SAVE\_DIR = 'faiss\_index'

# 1) Load PDF

if not os.path.exists(PDF\_PATH):

raise FileNotFoundError(f"Place a PDF named '{PDF\_PATH}' in the notebook folder or update PDF\_PATH")

loader = PyPDFLoader(PDF\_PATH)

docs = loader.load()

print('Loaded', len(docs), 'pages')

# 2) Split into chunks

splitter = CharacterTextSplitter(chunk\_size=CHUNK\_SIZE, chunk\_overlap=CHUNK\_OVERLAP)

chunks = splitter.split\_documents(docs)

print('Created', len(chunks), 'chunks')

# 3) embeddings

embeddings = SentenceTransformerEmbeddings(model\_name=EMBED\_MODEL)

# 4) build FAISS vectorstore

vectorstore = FAISS.from\_documents(chunks, embeddings)

print('Vectorstore created with', len(chunks), 'vectors')

**Cell 6 — Code (Persist FAISS)**

# Persist the FAISS index to disk

vectorstore.save\_local(SAVE\_DIR)

print('Saved FAISS index to', SAVE\_DIR)

**Cell 7 — Code (Reload FAISS + retriever)**

# reload\_faiss\_allow\_pickle.py

from langchain.embeddings import SentenceTransformerEmbeddings

from langchain.vectorstores import FAISS

EMBED\_MODEL = "all-MiniLM-L6-v2"

SAVE\_DIR = "faiss\_index"

emb = SentenceTransformerEmbeddings(model\_name=EMBED\_MODEL)

# WARNING: allow\_dangerous\_deserialization=True will load pickled Python objects.

# Only use this if you TRUST the files in SAVE\_DIR (you created them locally).

vectorstore = FAISS.load\_local(SAVE\_DIR, emb, allow\_dangerous\_deserialization=True)

retriever = vectorstore.as\_retriever(search\_kwargs={"k": 3})

print("Loaded FAISS index (pickle deserialization enabled). Retriever ready.")

**Cell 8 — Code (Create QA & run sample queries)**

# Run a few queries and print answers

queries = [

"Summarize the main conclusion of the document in one sentence.",

"What methods were used in the study?",

"Who are the authors?",

]

for q in queries:

print("\nQUESTION:", q)

try:

ans = qa.run(q) # qa was created earlier as RetrievalQA

print("ANSWER:", ans.strip())

except Exception as e:

print("Error while answering:", repr(e))

**Cell 9 — Code (Run a few QA queries (sanity checks))**

# Helper: get retrieved chunks for an arbitrary question

def show\_chunks(question, k=3):

docs = retriever.get\_relevant\_documents(question)

print(f'Retrieved {len(docs)} chunks for: {question}')

for i, d in enumerate(docs, 1):

print('\n--- CHUNK', i, '---')

print(d.page\_content[:800])

# Example usage (uncomment and edit):

# show\_chunks('What is the main conclusion of the paper?')

**Cell 10 — Markdown (Footer notes)**

---

\*\*Notes & troubleshooting\*\*

- If the Groq LLM call fails, check your `.env` and run the connectivity `requests.post` test shown earlier.

- If `SentenceTransformer` import errors occur, set `os.environ['TRANSFORMERS\_NO\_TF']='1'` before importing or use a clean conda env.

- To reuse the saved FAISS index in a Streamlit app, use `FAISS.load\_local` at app startup.

Enjoy — run the cells sequentially from top to bottom. If you want, I can also provide this notebook as a downloadable file.

If you want, I can now:

* convert the above into an .ipynb file and provide a downloadable link (I can recreate it and paste it here), or
* paste the same content in a single .py script version, or
* produce a step-by-step run order and suggested cells to execute (with expected outputs and checks).

Which would you prefer next?