**CHAPTER 6:**

**config.yml**

*RETURN\_SOURCE\_DOCUMENTS: True*

*VECTOR\_COUNT: 3*

*CHUNK\_SIZE: 1000*

*CHUNK\_OVERLAP: 100*

*DATA\_PATH: 'source\_data/'*

*DB\_FAISS\_PATH: 'vector\_storage/db\_faiss'*

*MODEL\_TYPE: 'llama'*

*MODEL\_BIN\_PATH: 'models/llama-2-7b-chat.ggmlv3.q8\_0.bin'*

*MAX\_NEW\_TOKENS: 256*

*TEMPERATURE: 0.2*

**prompt\_template**

*qa\_template = """Use the following pieces of information to answer the user's question.*

*If you don't know the answer, just say that you don't know, don't try to make up an answer.*

*Context: {context}*

*Question: {question}*

*Only return the helpful answer below and nothing else.*

*answer:*

*“””*

**llm.py**

*from langchain.llms import CTransformers*

*from dotenv import find\_dotenv, load\_dotenv*

*import box*

*import yaml*

*# Load environment variables from .env file*

*load\_dotenv(find\_dotenv())*

*# Import variables from config files*

*with open('config/config.yml', 'r', encoding='utf8') as ymlfile:*

*cfg = box.Box(yaml.safe\_load(ymlfile))*

*def build\_llm():*

*# Local CTransformers model*

*llm = CTransformers(model=cfg.MODEL\_BIN\_PATH,*

*model\_type=cfg.MODEL\_TYPE,*

*config={'max\_new\_tokens': cfg.MAX\_NEW\_TOKENS,*

*'temperature': cfg.TEMPERATURE}*

*)*

*return llm*

**utils.py**

*import box*

*import yaml*

*from langchain import PromptTemplate*

*from langchain.chains import RetrievalQA*

*from langchain.embeddings import HuggingFaceEmbeddings*

*from langchain.vectorstores import FAISS*

*from src.prompt\_template import qa\_template*

*from src.llm import build\_llm*

*# Import config vars*

*with open('config/config.yml', 'r', encoding='utf8') as ymlfile:*

*cfg = box.Box(yaml.safe\_load(ymlfile))*

*def set\_qa\_prompt():*

*"""*

*Prompt template for QA retrieval for each vectorstore*

*"""*

*prompt = PromptTemplate(template=qa\_template,*

*input\_variables=['context', 'question'])*

*return prompt*

*def build\_retrieval\_qa(llm, prompt, vectordb):*

*dbqa = RetrievalQA.from\_chain\_type(llm=llm,*

*chain\_type='stuff',*

*retriever=vectordb.as\_retriever(search\_kwargs={'k': cfg.VECTOR\_COUNT}),*

*return\_source\_documents=cfg.RETURN\_SOURCE\_DOCUMENTS,*

*chain\_type\_kwargs={'prompt': prompt}*

*)*

*return dbqa*

*def setup\_dbqa():*

*embeddings = HuggingFaceEmbeddings(model\_name="sentence-transformers/all-MiniLM-L6-v2",*

*model\_kwargs={'device': 'cpu'})*

*vectordb = FAISS.load\_local(cfg.DB\_FAISS\_PATH, embeddings)*

*llm = build\_llm()*

*qa\_prompt = set\_qa\_prompt()*

*dbqa = build\_retrieval\_qa(llm, qa\_prompt, vectordb)*

*return dbqa*

**vectordb\_build.py**

*import box*

*import yaml*

*from langchain.vectorstores import FAISS*

*from langchain.text\_splitter import RecursiveCharacterTextSplitter*

*from langchain.document\_loaders import PyPDFLoader, DirectoryLoader*

*from langchain.embeddings import HuggingFaceEmbeddings*

*# Import variables from config files*

*with open('config/config.yml', 'r', encoding='utf8') as ymlfile:*

*cfg = box.Box(yaml.safe\_load(ymlfile))*

*# create indexes in vector storage*

*def run\_db\_build():*

*loader = DirectoryLoader(cfg.DATA\_PATH, glob='\*.pdf', loader\_cls=PyPDFLoader)*

*documents = loader.load()*

*text\_splitter = RecursiveCharacterTextSplitter(chunk\_size=cfg.CHUNK\_SIZE, chunk\_overlap=cfg.CHUNK\_OVERLAP)*

*texts = text\_splitter.split\_documents(documents)*

*embeddings = HuggingFaceEmbeddings(model\_name='sentence-transformers/all-MiniLM-L6-v2',model\_kwargs={'device': 'cpu'})*

*vectorstore = FAISS.from\_documents(texts, embeddings)*

*vectorstore.save\_local(cfg.DB\_FAISS\_PATH)*

*if \_\_name\_\_ == "\_\_main\_\_":*

*run\_db\_build()*

**main.py**

*import box*

*import timeit*

*import yaml*

*import argparse*

*from dotenv import find\_dotenv, load\_dotenv*

*from src.utils import setup\_dbqa*

*# Load environment variables from .env file*

*load\_dotenv(find\_dotenv())*

*# Import variables from config files*

*with open('config/config.yml', 'r', encoding='utf8') as ymlfile:*

*cfg = box.Box(yaml.safe\_load(ymlfile))*

*if \_\_name\_\_ == "\_\_main\_\_":*

*parser = argparse.ArgumentParser()*

*parser.add\_argument('--input',*

*type=str,*

*default='How does the transformer architecture work?',*

*help='Enter your question?')*

*args = parser.parse\_args()*

*# Setup vector storage QA*

*start = timeit.default\_timer()*

*dbqa = setup\_dbqa()*

*response = dbqa({'query': args.input})*

*end = timeit.default\_timer()*

*print(f'\nAnswer: {response["result"]}')*

*print('='\*50)*

*# Process source documents*

*source\_docs = response['source\_documents']*

*for i, doc in enumerate(source\_docs):*

*print(f'\nSource Document {i+1}\n')*

*print(f'Source Text: {doc.page\_content}')*

*print(f'Document Name: {doc.metadata["source"]}')*

*print(f'Page Number: {doc.metadata["page"]}\n')*

*print('='\* 60)*

*print(f"Time to retrieve response: {end - start}")*