Slack-Integrated Retrieval-Augmented Generation (RAG) System

Objective

Build an intelligent Slack assistant powered by:

- **LangGraph** (for orchestrating retrieval & generation)
- **OpenAI (GPT-3.5)**
- **Qdrant** (Vector Database)
- **Flask** (Webhook receiver)
- **Ngrok** (Public tunnel for Flask server)

Overview Flow

1. User sends a message in Slack using `/askrag`
2. Slack sends the message to Flask via events endpoint
3. Flask receives and processes the Slack event
4. LangGraph queries Qdrant for contextual documents
5. OpenAI GPT-3.5 generates a relevant answer
6. Answer is posted back into Slack

Libraries & Their Purpose

Library	Purpose
`os`, `dotenv`	Load environment variables
`flask`	Handle Slack events as webhooks

Library	Purpose
`slack_sdk`	Interact with Slack (send messages, etc.)
`langchain_core`, `Document`	Define document structure
`RecursiveCharacterTextSplitter`	Chunk large text into smaller segments
`OpenAIEmbeddings`	Convert text to vector embeddings
`ChatOpenAI`	Access GPT-3.5 API for answers
`qdrant_client`	Interface with Qdrant Vector DB
`VectorParams`, `Distance`	Set Qdrant vector settings
`StateGraph`, `RunnableLambda`	Build LangGraph DAG

Step-by-Step Code Breakdown

Step 1: Load Environment

```
from dotenv import load_dotenv
load_dotenv()
```

Loads API keys like `SLACK_BOT_TOKEN`, `OPENAI_API_KEY`, etc.

Step 2: Load & Chunk Text

```
text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overla
p=200)
texts = text_splitter.split_text(raw_text)
documents = [Document(page_content=chunk) for chunk in texts]
```

Splits your source text into overlapping chunks to preserve context.

Step 3: Generate Embeddings

```
embedding_function = OpenAIEmbeddings()
```

Converts chunks into 1536-dimensional embeddings via OpenAI.

Step 4: Initialize Qdrant

```
qdrant_client = QdrantClient(host="localhost", port=6333)
qdrant_client.recreate_collection(
    collection_name="rag_txt_collection",
    vectors_config=VectorParams(size=1536, distance=Distance.COSINE),
)
db = Qdrant(client=qdrant_client, collection_name="rag_txt_collection", embe
ddings=embedding_function)
db.add_documents(documents)
```

Creates and populates the Qdrant collection.

Step 5: Define LangGraph State

```
class GraphState(TypedDict):
    question: str
    context: str
    answer: str
```

Shared state passed between LangGraph nodes.

Step 6: Retrieval Node

```
def retrieve(state: GraphState):
    query = state["question"]
    retriever = db.as_retriever()
    docs = retriever.invoke(query)
    context = "\n\n".join([doc.page_content for doc in docs])
    return {"question": query, "context": context}
```

Fetches relevant context from Qdrant.

Step 7: Generation Node

```
llm = ChatOpenAI(model="gpt-3.5-turbo")

def generate(state: GraphState):
    prompt = f"""Answer the question using this context:\n\n{state['context']}\n
\nQuestion: {state['question']}"""
    response = llm.invoke(prompt)
    return {**state, "answer": response.content}
```

Uses GPT-3.5 to craft a natural language response.

Step 8: LangGraph Pipeline

```
graph = StateGraph(GraphState)
graph.add_node("retrieve", RunnableLambda(retrieve))
graph.add_node("generate", RunnableLambda(generate))
graph.set_entry_point("retrieve")
graph.add_edge("retrieve", "generate")
graph.add_edge("generate", END)
rag_app = graph.compile()
```

Defines a simple two-node LangGraph pipeline.

Step 9: Slack Event Handler (Flask)

```
@app.route("/slack/events", methods=["POST"])
def slack_events():
    event = request.json.get("event", {})
    if "bot_id" in event:
        return Response("Ignored", status=200)

    user_question = event.get("text")
    channel_id = event.get("channel")

    if user_question:
        state = {"question": user_question, "context": "", "answer": ""}
        final_state = rag_app.invoke(state)
        answer = final_state["answer"]
        client.chat_postMessage(channel=channel_id, text=f":brain: Answer: {an
swer}")

    return Response("OK", status=200)
```

Handles Slack events, invokes LangGraph, and replies in-channel.

Step 10: Launch Flask Server

```
if __name__ == "__main__":
    app.run(port=5000)
```

Server is tunneled via **Ngrok** to Slack.

Summary

Step	Action
1	Load environment variables
2	Chunk and embed documents
3	Store vectors in Qdrant
4	Query relevant context
5	Generate answer using GPT-3.5
6	Reply to user in Slack
