Detailed Explanation of Code Files: RAG Agent, Indexing Script And Embedding Function

File 1: RAG Voice Agent (main.py)

This file defines and runs a LiveKit-based voice assistant for the Orion Store, which uses Retrieval-Augmented Generation (RAG) to answer questions by searching a knowledge base stored in Qdrant.

Key Functionalities:

1. **Environment & Logging Setup:**
    - Loads environment variables from `.env` (e.g., Qdrant URL, API Key).
    - Configures logging for debugging and monitoring.
2. **RAGEnrichedAgent Class:**
    - Inherits from `Agent` provided by `livekit.agents`.
    - Defines the assistant's behavior and integrates Qdrant for knowledge retrieval.
    - Loads the embedding function and initializes a Qdrant client.
    - Keeps track of previously seen search results to avoid repetition.
3. **Function Tool - `info_search`:**
    - Exposes a tool the LLM can use to search the Orion Store's knowledge base.
    - Performs a similarity search in Qdrant for relevant information.
    - Filters out previously seen chunks.
    - Logs query processing latency.
    - Returns relevant information if found, otherwise suggests rephrasing.

4. **Session Event Handlers:**
    - `on_enter`: Sends a greeting message when the session starts.
    - `on_transcription`: Handles user speech, ignores filler words, and generates replies.
5. **Entrypoint Function:**
    - Connects to the LiveKit server.
    - Initializes a `AgentSession` with Google Gemini LLM and TTS.
    - Starts the agent in the specified room.
6. **Main Execution:**
    - Runs the agent using `cli.run_app()` with the provided entrypoint.

File 2: Indexing Script (rag_builder.py)

This script prepares and indexes documents (PDFs) into Qdrant so that the RAG system can retrieve relevant information.

Key Functionalities:

1. **Environment & Setup:**
    - Loads environment variables (Qdrant URL, API Key).
    - Defines constants for collection name and data path.
2. **Main Function:**
    - Parses the `--reset` argument to optionally clear and recreate the Qdrant collection.
    - Loads PDF documents from the `data/` directory.
    - Splits documents into smaller chunks for better search performance.

- Adds the chunks to Qdrant with unique UUIDs.

3. **Helper Functions:**
   - `load_documents`: Loads all PDFs using `PyPDFDirectoryLoader` from LangChain.
   - `split_documents`: Splits documents into 300-character chunks with 30-character overlap.
   - `calculate_chunk_ids`: Assigns unique UUIDs to each chunk's metadata.
   - `add_to_qdrant`: Adds all chunks to Qdrant using LangChain's Qdrant wrapper.
4. **Qdrant Collection Management:**
   - If `--reset` is provided, deletes the existing collection and recreates it with cosine distance and 1536-dimension vectors.

Summary of Workflow:

1. You run the **Indexing Script** to load, split, and store your knowledge (PDFs) into Qdrant.
2. You run the **RAG Voice Agent**, which uses LiveKit for voice interaction and Qdrant for RAG-powered answers.
3. The agent listens to the user's questions, searches Qdrant, and responds conversationally using Google Gemini LLM and TTS.

Requirements for Both Files:

- `livekit-agents`, `livekit-plugins`
- `langchain`, `langchain-community`, `langchain-qdrant`, `langchain-text-splitters`

- `qdrant-client`, `python-dotenv`
- `google-cloud-speech`, `google-generativeai`, `openai`
- `poppler-utils` (system dependency for PDF extraction)

This setup provides a complete voice assistant with RAG, ideal for answering store-related questions using structured knowledge from PDFs.

File 3: Embedding Function(get_embedding_function.py)

This file defines a function that provides the embedding model used to generate vector representations of text. These embeddings are essential for tasks like semantic search, Retrieval-Augmented Generation (RAG), and storing documents in vector databases like Qdrant.

What Happens in This File:

1. **Environment Variables Loaded**
    - The `dotenv` library loads values from a `.env` file.
    - This is commonly used to securely store API keys and configuration for embedding providers like OpenAI or AWS Bedrock.
2. **Import Options for Embedding Models**
    - The file shows multiple options for generating embeddings:
        - `OpenAIEmbeddings`: Uses OpenAI's hosted models.
        - `OllamaEmbeddings` (commented out): Allows using a local Ollama model.
        - `BedrockEmbeddings` (commented out): Allows using Amazon Bedrock's embedding service.

- Only `OpenAIEmbeddings` is active in the current version.

3. **`get_embedding_function()` Definition**
    - This function returns an initialized embedding model.
    - By default, it uses OpenAI's `"text-embedding-3-small"` model.
    - You can switch to Ollama or Bedrock by simply uncommenting the relevant lines and commenting out the OpenAI line.