**✅ Next Question (22):**

**What are LangChain Agents? How do they differ from chains? Can you give a use case?**

**You're right again!** Agents in LangChain are about **autonomous decision-making** and executing tasks using tools.

**✅ Question 24:**

**What are embeddings, and how are they used in GenAI applications like RAG or semantic search?**

**Embeddings** are dense **vector representations** of text (words, sentences, or documents) that capture their **semantic meaning** in a numerical format.

**How Embeddings Work in RAG / Semantic Search:**

1. **Text chunks** from documents (PDFs, webpages, etc.) are converted into vectors using an **embedding model** like:
   * text-embedding-ada-002 (OpenAI)
   * all-MiniLM (HuggingFace)
   * Azure OpenAI Embeddings
2. These vectors are stored in a **vector database** (like FAISS, Pinecone, Chroma).
3. At query time:
   * The user question is embedded
   * The system performs a **similarity search** (like cosine similarity)
   * Top matching chunks are **retrieved as context** for the LLM

**✅ Next Question (25):**

**In a GenAI document Q&A system, how would you chunk large documents before embedding, and why is chunking important?**

**📚 Why is Chunking Important?**

LLMs have a **token limit** (e.g., 4k, 8k, or 32k tokens), so you can’t feed an entire document at once.  
**Chunking** allows you to break large documents into **smaller, manageable pieces**, so:

* You can embed and store them efficiently
* At query time, you can retrieve only the **most relevant chunks** based on semantic similarity

**🧩 How to Chunk Documents for Embedding:**

**✅ 1. Fixed-size Chunking (most common)**

* Break the document into **fixed token or word windows**, often 200–500 tokens
* Use **overlap** (e.g., 20–50 tokens) between chunks to maintain context

**✅ Next Question (26):**

**When using an embedding-based RAG system, how do you handle hallucination by the LLM? Give at least two strategies.**

**🤯 How to Handle Hallucination in RAG Systems**

**Hallucination** = when an LLM generates information that **sounds correct** but is **factually incorrect** or **not in the source data**.

**✅ 1. Design Clear, Constrained Prompts**

* Make sure the prompt tells the model to **only answer based on the retrieved documents**.

🔹 **Example Prompt**:

"Use ONLY the provided context to answer. If the answer is not in the context, respond with 'Information not available.'"

This avoids the model “guessing.”

**✅ 2. Lower the Temperature**

* **Temperature** controls randomness.
  + 0 = deterministic (safe)
  + 1 = creative (risky)

📉 **Set temperature to 0 or 0.2** in production to **reduce hallucination**.

**✅ 3. Retrieval Quality**

* If irrelevant or low-quality chunks are retrieved, LLMs can hallucinate.
* Use:
  + Better **chunking strategy**
  + **Re-ranking** of search results
  + **Hybrid search** (dense + keyword)

**✅ 4. Response Validation / Post-processing**

* Add rule-based checks:
  + If the LLM outputs entities or numbers, validate against your source
* Optionally use another model or logic for **fact-checking**

**Bonus (Advanced):**

* Use **guardrails** like Microsoft’s **Azure Content Safety** or OpenAI's **function calling** to ensure safe and grounded responses.

**✅ Next Question (27):**

**What are function calls (a.k.a tool calling) in LLMs like OpenAI or Azure OpenAI? How are they used in GenAI apps?**

**🔧 What are Function Calls (Tool Calling) in LLMs?**

**Function calling** (also called **tool calling** in LangChain or **actions** in some platforms) allows an LLM to **trigger external functions/APIs** based on the user's input — in a structured and controllable way.

It enables the LLM to act more like an **agent** that doesn’t just reply, but can **perform actions** like:

* Fetching data
* Performing calculations
* Triggering workflows
* Reading from a database

**📦 How It Works (Conceptually):**

1. You define a function schema (JSON format)
2. The LLM decides if a function is needed and fills in the arguments
3. Your backend executes the function
4. The result is sent back to the LLM for a final answer