Mowka
Grow with Confidence

TOP 30

RAG

Interview Questions

These RAG Questions Decide Your AI Interview — With Answers

Shubham Kansal
Founder | Mouka
https://mowka.in/

Question 1

Explain Retrieval Augmented Generation (RAG) and its importance for Large Language Models.

ANSWER

Retrieval Augmented Generation (RAG) is an approach that enhances Large Language Models (LLMs) by linking them to external, proprietary data sources.

Traditional LLMs face limitations because their knowledge remains fixed at their training cutoff date and they lack access to private organizational data. RAG addresses these limitations through:

Retrieving current or proprietary information from external sources such as databases.
Incorporating that information directly into the prompt.
Generating responses using the provided context.

Question 2

What are the two primary phases of a RAG application architecture?

ANSWER

RAG applications consist of two main phases:

- Indexing (Preparation Phase): This occurs offline. Data is collected, processed, and structured. The process includes loading documents, dividing them into smaller segments, and saving them in a vector database for future retrieval.
- Retrieval and Generation (Runtime Phase): This occurs when a user submits a query. The system fetches relevant segments from the index and provides them to the LLM to produce the final response.

Presented By:

Question 3

Explain how the indexing process works and its significance.

ANSWER

Indexing organizes data to enable efficient processing by LLMs. It consists of three main steps:

Load: Bringing in data through Document Loaders (for formats like PDFs or web pages).
Split: Dividing large documents into smaller segments using Text Splitters.
Store: Transforming segments into numerical representations (embeddings) and saving them in a VectorStore.

Proper indexing is essential because without it, the system cannot search through data quickly enough to provide real-time responses.

Question 2

What are the two primary phases of a RAG application architecture?

ANSWER

RAG applications consist of two main phases:

- Indexing (Preparation Phase): This occurs offline. Data is collected, processed, and structured. The process includes loading documents, dividing them into smaller segments, and saving them in a vector database for future retrieval.
- Retrieval and Generation (Runtime Phase): This occurs when a user submits a query. The system fetches relevant segments from the index and provides them to the LLM to produce the final response.

Presented By:

Question 4

What function does the Retrieval component serve?

ANSWER

The Retrieval component serves as a connector between user queries and the database. Its primary responsibility is identifying the most relevant information (context) based on the user's question.

When retrieval fails and returns irrelevant data, the LLM may produce hallucinations or incorrect answers. Effective retrieval is crucial for producing accurate RAG results.

Question 5

Why do we need to break documents into smaller chunks?

ANSWER

You cannot process an entire 100-page document with an LLM in a single operation. Document splitting is necessary for several reasons:

- Context Window Constraints: LLMs have a maximum limit on the amount of text they can process (known as the context window).
- Accuracy: Locating a specific section is more precise when searching through focused segments rather than entire documents.
- Performance: Smaller chunks process faster and cost less to embed.

Question 6

Why is LangSmith configuration important for complex applications?

ANSWER

LangSmith provides observability capabilities. When developing complex chains, identifying where errors occur or understanding why an agent produced a particular response can be challenging.

LangSmith enables developers to trace application execution, offering a detailed log showing exactly what inputs were sent to the model and what outputs were generated. This capability is crucial for debugging and performance optimization.

Presented By:

Question 7

What dependencies are required for setting up a LangChain RAG system?

ANSWER

Building a standard RAG pipeline requires three essential packages, which can be installed using pip:

- langchain: The core framework for constructing chains.
- langchaincommunity: Provides third-party integrations including loaders and specialized tools.
- langchainchroma: Enables connection to the Chroma vector database (alternatively, you can use langchainopenai for models).

pip install langchain langchaincommunity langchain_chroma

Question 8

What functions do `WebBaseLoader` and `RecursiveCharacterTextSplitter` perform?

ANSWER

These two components collaborate to process web content:

- WebBaseLoader: Extracts HTML from a given URL and transforms it into plain text.
- RecursiveCharacterTextSplitter: Processes that text and intelligently divides it into smaller segments. It's called recursive because it attempts to preserve paragraph and sentence boundaries instead of splitting sentences mid-way.

Presented By:

Question 9

What purpose do the environment variables `LANGCHAIN_TRACING_V2` and `LANGCHAIN_API_KEY` serve?

ANSWER

These variables enable connection between your local application and the LangSmith cloud service:

• LANGCHAINTRACINGV2: A toggle (set to true) that activates the logging and tracing functionality.

• LANGCHAINAPIKEY: Your authentication credential that permits your code to send logs to your designated LangSmith project.

Question 10

What is the main function of DocumentLoaders?

ANSWER

DocumentLoaders normalize incoming data from various sources. Whether the source is a text file, website, or CSV, the loader transforms it into a collection of Document objects.

Each Document object contains two essential elements:

page_content: The actual text or information.

metadata: Additional context such as source URL or page number, which helps with source attribution later.

Presented By:

Question 11

How does `WebBaseLoader` operate in the LangChain framework?

ANSWER

WebBaseLoader connects your application to web content. It enables LangChain to retrieve live web pages through:

Fetching: Using urllib to retrieve raw HTML from a URL.
Parsing: Using BeautifulSoup to remove HTML tags and extract readable text.
Formatting: Converting that text into a collection of Document objects (with content and metadata) ready for processing.

Question 12

How can you modify the HTML parsing behavior in `WebBaseLoader`?

ANSWER

You can control what the loader extracts by providing arguments to the underlying parser through the bs_kwargs (BeautifulSoup keyword arguments) parameter.

This is frequently needed because many websites include unnecessary elements such as navigation menus, footer links, and advertisements. By setting specific parameters (such as using a SoupStrainer), you can direct the loader to skip these elements and extract only text from designated HTML classes like post-content or article-body.

Question 2

What are the two primary phases of a RAG application architecture?

ANSWER

RAG applications consist of two main phases:

- Indexing (Preparation Phase): This occurs offline. Data is collected, processed, and structured. The process includes loading documents, dividing them into smaller segments, and saving them in a vector database for future retrieval.
- Retrieval and Generation (Runtime Phase): This occurs when a user submits a query. The system fetches relevant segments from the index and provides them to the LLM to produce the final response.

Presented By:

Question 13

What role does BeautifulSoup play in this workflow?

ANSWER

BeautifulSoup handles data cleaning. While WebBaseLoader establishes the connection, BeautifulSoup processes the complex HTML structure (the DOM). It removes code elements (tags like div, script, style) and extracts human-readable text, ensuring the data sent to the LLM is clean and usable.

Question 14

What is `SoupStrainer` and how does it enhance performance?

ANSWER

SoupStrainer is a filtering mechanism used during parsing. Rather than loading the entire webpage and then removing unwanted content, SoupStrainer instructs the system to parse only specific tags from the start.

For instance, you can configure it to target only div tags with the class main-content. This approach dramatically improves processing speed and ensures unnecessary content (such as sidebars) never enters memory.

Presented By:

Question 15

Why is custom HTML parsing important for RAG systems?

ANSWER

Output quality depends directly on input quality (the principle of Garbage In, Garbage Out).

If your RAG system receives headers, copyright notices, and advertisements, the retrieval mechanism may mistake this noise for actual content. Custom parsing ensures only valuable content gets indexed, resulting in more accurate and focused AI responses.

Question 16

Why should long documents be segmented before embedding?

ANSWER

Document segmentation (chunking) is required for two key reasons:

Context Window Limitations: LLMs have a fixed memory capacity. While you cannot fit a 300-page manual into one prompt, you can include relevant paragraphs.

Search Accuracy: Finding specific answers is much simpler when the database contains small, focused concepts rather than large, unstructured documents.

Presented By:

Question 17

What is chunk overlap and what purpose does it serve?

ANSWER

Chunk overlap creates a sliding window effect where the end of one chunk repeats at the start of the next (for example, a 200-character overlap).

This is important for maintaining context. Without overlap, a sentence or concept might be split at the boundary, causing the model to lose the relationship between related statements. Overlap connects the segments so meaning is preserved.

Question 18

Why is `RecursiveCharacterTextSplitter` preferred for general text?

ANSWER

This splitter intelligently handles text division. Instead of arbitrarily cutting text every 1,000 characters, it attempts to split hierarchically by:

Paragraphs (\n\n)
Sentences (\n)
Spaces

This approach ensures semantically related text remains together. It prevents splitting sentences mid-way, which helps the LLM better understand the complete meaning of retrieved segments.

Question 19

How does the `add_start_index=True` parameter benefit the process?

ANSWER

This feature assigns a coordinate to each text segment. It records the character position (start_index) where that segment originally appeared in the source document.

This is particularly valuable for citations. It enables the application to trace answers back to their exact location in the original file, helping users verify the source of information.

Question 20

What does the `split_documents()` method return?

ANSWER

The splitdocuments() method returns a Python list containing Document objects.

Each object represents one segment and includes:

- pagecontent: The text portion.
- metadata: Source information, including the new start_index if configured.

This list represents the final format needed to start the embedding and indexing workflow.

Presented By:

Question 21

What is the goal of embedding text and storing it in a vector store?

ANSWER

The embedding process converts human language into a format that computers can efficiently understand and search.

- Embedding: An embedding model (such as OpenAI Embeddings) transforms a text segment into a numerical vector that captures the meaning of that text.
- Storing: These vectors are stored in a Vector Store (a specialized database).

Question 22

What is Cosine Similarity and why is it used for vector searches?

ANSWER

Cosine similarity measures how closely related two pieces of text are in meaning.

When a user asks a question, it gets converted into a vector. The system then computes the angle (cosine) between the question vector and all document vectors in the store.

- Small Angle: Documents are very similar (relevant).
- Large Angle: Documents are unrelated.

It's preferred over other methods because it focuses on vector direction (semantic meaning) rather than magnitude (length), making it highly effective for matching questions to answers.

Question 23

What components are needed to build and query a vector store?

ANSWER

To set up a vector store in LangChain, you need four interconnected components:

Documents: The source material (segmented text).
Embedding Model: The converter that transforms text into vectors.
Vector Database: The storage system (such as Chroma or Pinecone) that holds the vectors.
Similarity Search: The function that queries the database to locate the nearest neighbors to your question.

Question 24

What challenges do developers commonly encounter with the Chroma vector store?

ANSWER

Developers typically encounter three main issues:

• Data Formatting: Chroma requires properly formatted Document objects. Providing raw strings or incorrectly structured lists will trigger errors.
• API Key Management: Since embedding often relies on external APIs (like OpenAI), missing or incorrectly configured environment keys will halt the ingestion process.
• Parameter Understanding: Distinguishing between search parameters (such as k for result count versus specific distance thresholds) can be confusing but is essential for optimizing results.

Question 25

How can you examine the contents of a vector store?

ANSWER

Unlike traditional SQL databases, you cannot simply open a vector store to view rows directly. To confirm data exists, you typically use:

- collection.count(): A method that returns the total number of stored segments.
- Test Queries: Running a similaritysearch with a general term to see what results are returned.

These verification steps are needed to confirm your ingestion pipeline worked correctly and that the store contains data.

Question 26

What is the workflow for embedding and storing documents?

ANSWER

The typical workflow consists of four main stages:

Prepare: Transform your raw text segments into the proper Document object format.
Configure: Set up your embedding model (make sure API keys are configured).
Ingest: Use the from_documents() method. This embeds the text and inserts the vectors into the Chroma store in a single operation.
Validate: Run a test query right away to confirm the system returns relevant results.

Question 27

What role does a Retriever play in LangChain?

ANSWER

A Retriever is an abstraction layer that provides a standardized way to fetch data.

While a Vector Store is the database, a Retriever is the tool that communicates with that database. It accepts a string (the user's question) as input and returns a list of Document objects. It hides the complex mathematics of vector search, allowing the rest of the application to request relevant documents without understanding how they are located.

Question 28

How does `VectorStoreRetriever` work?

ANSWER

This is a specialized retriever that interfaces with a vector database.

- Input: It accepts a user's query.
- Process: It executes a vector similarity search within the associated store.
- Output: It returns the top k most similar document segments.

Its main purpose is to narrow down the large database to just the specific paragraphs needed to answer the current question.

Question 29

Why should you review the documents returned by the retriever?

ANSWER

Reviewing retrieved documents is a quality assurance step. Before sending data to the LLM, you should confirm that:

The documents are actually relevant to the question.
The text is readable and not corrupted code.

If the retriever returns irrelevant context, the LLM will produce poor results. Examining this output helps developers adjust chunk size or change the embedding model to enhance accuracy.

Question 30

What is the objective of combining retrieval with generation?

ANSWER

Integration merges the Researcher (Retrieval) with the Writer (Generation).

- Without integration: You have a search engine that returns links, or a chatbot that invents facts.
- With integration: The system locates facts first, then uses the LLM to combine those facts into a coherent, natural response.

This combination creates a system that is both factually accurate (due to retrieval) and conversational (due to generation).