Topics learnt in LangChain

1. What are LLMs?
2. What is langchain?
3. Components of langchain?
    a. Models
    b. Prompts
    c. Chains
    d. Agents
    e. Memory
    f. Retrievel
4. What are **Models**?
    a. Language Models (Text → Text)
        i. LLMs
        ii. Chat Models
    b. Embedding Models (Text → Vector)
5. What are **Prompts**?
    a. Types of prompts
        i. Static Prompts
        ii. Dynamic Prompts
    b. **PromptTemplate** (For creating dynamic prompts)
        i. Why Prompt template over f string?
            1. Validations available
            2. Reusable by exporting in JSON
            3. Fit well with LangChain ecosystems
6. What are **messages**?
    a. Single Message
        i. Static Message
        ii. Dynamic Message (**PromptTemplate**)
    b. List of Messages (Multi turn conversations) (Chat History)
        i. Static Message
            1. AIMessage
            2. HumanMessage
            3. SystemMessage
        ii. Dynamic Message (**ChatPromptTemplate**)
    c. **MessagePlaceholder** (special placeholder used inside ChatPromptTemplate to dynamically insert chat history or list list of messages at runtime)
7. What and how of **Structured output**?
    a. Ways to define structured output
        i. TypeDict
            1. Annotated
            2. Optional

            3. Literals
        ii. **Pydantic**
            1. Data validation and data parsing library
            2. It makes sure that data you work with is correct, structured and type safe.
        iii. **Json_schema**
    b. Output parsers (For open source models, how don't have access to with_structured_output and can't provide you structured output)
        i. StrOutputParser
        ii. JSONOutputParser
        iii. StructuredOutputParser
        iv. PydanticOutputParser
8. **Chains** in LangChain?
    a. What and why of chains?
    b. Kinds of chains?
        i. Simple Chain
        ii. Structured Chain
        iii. Parallel Chain
        iv. Conditional Chain
9. **Runnables** in LangChain?
    a. Runnables are different components inside LangChain ecosystem for some particular tasks
        i. Documents Loader
        ii. Retrievers
        iii. Vector Stores
        iv. Parsers
    b. Types of Runnable
        i. Runnable Sequence
        ii. Runnable Parallel
        iii. Runnable Passthrough
        iv. Runnable Lambda
10. What is **RAG**?
    a. Components of RAG?
        i. Document Loaders
        ii. Text splitters
        iii. Vector Databases
        iv. Retrievers
11. DocumentsLoaders
    a. Concept?
    b. Most popular types of loaders
        i. TextLoaders
        ii. PyPDFLoaders
        iii. DirectoryLoader
        iv. WebBaseLoader

v. CSVLoader

c. Load vs Lazy Load

12. Text Splitters

a. What?

i. Process of splitting larger chunks of text (i.e. articles, PDF files, HTML pages, or books) into smaller manageable chunks that an LLM can handle efficiently

b. Reasons of using Text Splitter

i. Content length limitation of LLMs

ii. Text splitting improves nearly every LLM tasks

1. **Embeddings**: Short chunks yield more accurate vectors
2. **Semantic search** improves
3. **Summarization** improves

iii. Optimize computational resources

c. Text Splitters types

i. Length Based (Chunk size can be in characters or tokens)

ii. Text Structure Based (Based on text structure and add overlay to avoid breaking words or sentences)

iii. Document Structure Based

iv. Semantic Meaning Based

13. Vector Stores

a. Why a vector store?

i. Embedding vector generation?

ii. Vector databases

iii. Semantic search

b. What are vector stores?

i. System design to store and retrieve data represented as numerical vectors.

c. Key features of Vector Stores

i. Ensures that vectors and their associated metadata and retained, whether **in-memory** for faster look ups or on **disk** for durability and large case use

ii. Similarity search

iii. Indexing

d. Vector Store VS Vector databases

i. Vector Store

1. Typically focus on storing vectors(embeddings) and performing similarity search
2. May not include traditional databases features like transactions, RBAC, ACID properties
3. Example(FAISS, when you can store vectors and can query them but you handle persistence and scaling separately)

ii. Vector Database

1. A full fledge database system designed to store and query vectors
2. Example (Pinecone)

e. Vector Stores in LangChain

14. **Retrievers**
    a. What?
        i. Take user query as input and give that query to data source and get documents as output
    b. Multiple types of retrievers
        i. Wikipedia Retriever
            1. It queries the wikipedia API to fetch relevant content
            2. How it works?
                a. You give it a query
                b. It sends the query to wikipedia API
                c. It retrieves the most relevant articles
                d. It returns them as LangChain **Document** Objects
        ii. **Vector Store Retriever**
            1. It search and fetches the documents from a vector store based on semantic similarity using vector embeddings
            2. How it works?
                a. You store your document in a vector store
                b. Each document is converted into a dense vector using an embedding models
                c. When the user enters a query
                    i. It's also turned into vector
                    ii. The retriever compares the query vector with the stored vector
                    iii. It retrieves the top-k most similar ones
        iii. Maximum marginal relevance (MMR)
            1. It is designed to reduce redundancy in the retrieved results while maintaining high relevance to the query
            2. Why MMR Retriever?
                a. In regular similarity search, you may get documents that are:
                    i. All very similar to each other
                    ii. Repeating the same info
                    iii. Lacking diverse perspectives
                b. MMR Retriever avoids that by:
                    i. Picking the most relevant document first
                    ii. Then picking the next most relevant and least similar to already selected doc
                    iii. And so on…
                c. This helps especially in RAG pipelines where:
                    i. You want your context window to contain diverse but still relevant informatic
                    ii. Especially useful when documents are semantically overlapping

15. **RAG** (Retrievel augumented generation)