🔹 LangChain – Interview Q&A

Q1. What is LangChain?
A: LangChain is a framework used to build GEN AI applications using Large Language Models (LLMs).
It provides components like Prompts, Chains, Agents, Tools, and Memory to integrate LLMs with external data and APIs.

Q2. Why do we need LangChain when we already have LLMs?
A:

- LLM's give 1 shot answer but we need complete application
- LLMs can't access external data, APIs, or databases by themselves.
- LangChain provides **orchestration for building appplications** → coordinates LLMs, tools, and workflows.
- Enables **real apps**: chatbots, Q&A, automation pipelines.

Q3. What are the core components of LangChain?
A:

1. **LLMs** → the "brain".
2. **Prompts** → instructions to LLM.
3. **Chains** → sequences of steps.
4. **Agents** → decision-makers.
5. **Tools** → external functions/APIs.
6. **Memory** → stores context/history.
7. **Output Parsers** → enforce structured outputs.

Chains & Orchestration

Q4. What is a Chain in LangChain?
A: A **Chain** connects multiple components together in sequence.
Types:

- **Simple Chain** → Prompt → LLM.
- **Sequential Chain** → Multi-step (output of one → input to next).
- **Custom Chain** → Mix LLMs, APIs, logic.

Q5. What is the difference between Chain and Orchestration?
A:

- **Chain** → Provides Linear sequence (one after another).
- **Orchestration** →Provides Multi step coordination (like a director managing multiple actors).

Q6. What are Agents in LangChain?
A: Agents are entities or **dynamic decision-makers** that:

- Decide which tool/step to use at runtime.
- Example: If asked "What's 123 × 456?", Agent decides → use calculator tool instead of LLM.

Prompts & LLMs

Q7. What are Prompt Templates in LangChain?
A:

- Structure for prompt which is reusable
- Templates that allow dynamic variables.
- Example: `"Translate this sentence to French: {sentence}"` → user provides sentence at runtime.

Q8. What's the difference between static and dynamic prompts?
A:

- **Static** → Fixed text instructions.
- **Dynamic** → Include variables/inputs substituted at runtime.

Memory Systems

Q9. What is Memory in LangChain? Why is it important?
A: Memory stores **past conversation/context**.
Types:

- **ConversationBufferMemory** → recent chat history.
- **SummaryMemory** → compressed summary of past chats.
- **Vector-based Memory** → semantic embeddings for long-term recall.

Q10. How does LangChain handle long conversations?
A: By using **SummaryMemory** or **Vector Memory** to avoid exceeding token limits.

Output Parsers

Q11. What are Output Parsers in LangChain?
A: They enforce structure in LLM responses.
Types:

- **StrOutputParser** → plain string.
- **StructuredOutputParser** → JSON/dict/list format.
- **PydanticOutputParser** → strict schema validation (ensures types).

Comparisons

Q12. Compare LangChain with LlamaIndex and Haystack.
A:

- **LangChain** → best for workflow orchestration, agents, custom apps.
- **LlamaIndex** → focuses on data ingestion + building indexes for RAG.
- **Haystack** → enterprise-ready QA pipelines, production use.

Advanced

Q13. What are some limitations of LangChain?
A:

- Can be **complex/over-engineered** for small apps.
- **Latency** → chaining multiple LLM/tool calls.
- Debugging is difficult with dynamic agents.
- Fast-changing → breaking API updates.

Q14. How do you optimize performance in LangChain apps?
A:

- Cache prompts & responses.
- Use smaller LLMs for trivial steps.
- Minimize unnecessary tool calls.
- Use parallel chains when possible.

Q15. What are real-world use cases of LangChain?
A:

- **RAG-based chatbots** (PDFs, company docs).
- **Customer support assistants**.
- **Data extraction & summarization pipelines**.
- **Agents for workflow automation** (e.g., fetch data from APIs, generate reports).

Q16. How does LangChain integrate with vector databases?
A:

- Provides connectors for FAISS, Pinecone, Chroma, Weaviate.
- Allows storing/retrieving embeddings inside Chains/Agents

Q17. How would you design a LangChain-based chatbot with memory and retrieval?
A:

- **Step 1:** Ingestion → Load documents into a vector DB.
- **Step 2:** Retrieval Chain → User query → embedding → retrieve top chunks.
- **Step 3:** Memory → Use `ConversationBufferMemory` or `VectorStoreRetrieverMemory` for context.
- **Step 4:** Chain → Combine query + retrieved docs + memory into a prompt.
- **Step 5:** LLM → Generate contextual answer.

Q18. How does LangChain integrate with APIs/tools?
A:

- Tools are wrapped as **functions**.
- Example: `SerpAPI`, `Calculator`, `SQLDatabaseChain`.
- Agents decide dynamically whether to call LLM directly or use a tool.

Q19. How is LangChain different from LlamaIndex?
A:

- **LangChain** → Focuses on orchestration (tools, agents, workflows).
- **LlamaIndex** → Focuses on indexing + querying custom data.
- Often used together: LlamaIndex (data access) + LangChain (workflow).

Q20. How is LangChain different from Haystack?
A:

- **LangChain** → Flexible, modular, dev-friendly.
- **Haystack** → Enterprise-ready, production-first, robust pipelines.
- LangChain better for prototyping; Haystack better for deployment at scale.

Q21. Give real-world examples where LangChain is used.

- **Financial research assistants** → RAG + live stock APIs.
- **Healthcare assistants** → Q&A over medical docs.
- **Legal AI** → Summarize + cite judgments.
- **Enterprise bots** → Search over SharePoint, Confluence.