🔥 2025 EDITION

Top 30 Interview Questions

on LangChain

Master LangChain concepts in 5 minutes and ace your next AI interview

🔗 Chains

🤖 Agents

📚 RAG

🧠 Memory

⚒️ Tools

Section 1

LangChain Fundamentals

Questions 1-10

Q1 FUNDAMENTALS

What is LangChain and why do we need it?

LangChain is a Python framework that makes it easy to build applications using Large Language Models (LLMs). Think of it like Lego blocks for AI apps!

> **WHY WE NEED IT**
>
> Without LangChain, you'd have to write tons of code to connect your AI to databases, APIs, and other tools. LangChain does the heavy lifting for you.

- ☑ **Standardizes** how you work with different AI models (OpenAI, Claude, etc.)
- ☑ **Connects** AI to external data sources and tools
- ☑ **Simplifies** building chatbots, RAG apps, and AI agents

Q2 FUNDAMENTALS

What are the core components of LangChain?

LangChain has **5 main building blocks** that work together:

- ✓ **Models:** The AI brain (GPT-4, Claude, etc.) that generates text
- ✓ **Prompts:** Templates that tell the AI what to do
- ✓ **Chains:** Connect multiple steps together (like a recipe)
- ✓ **Agents:** AI that can decide which tools to use
- ✓ **Memory:** Remembers past conversations

Q3 FUNDAMENTALS

How do you initialize a Chat Model in LangChain?

Use **init_chat_model()** to create a connection to any AI model. It's the easiest way to start!

PYTHON

```python
from langchain.chat_models import init_chat_model

# Initialize OpenAI GPT-4
model = init_chat_model("gpt-4o")

# Or use Claude
model = init_chat_model("claude-sonnet-4-5-20250929")

# Send a message
response = model.invoke("Hello, how are you?")
print(response.content)
```

KEY POINT

init_chat_model() automatically detects which provider to use based on the model name!

Q4 FUNDAMENTALS

What is LCEL (LangChain Expression Language)?

LCEL is LangChain's way of connecting components using the **pipe operator (|)**. Think of it like connecting water pipes!

PYTHON

```python
from langchain_core.prompts import ChatPromptTemplate
from langchain.chat_models import init_chat_model

# Create a prompt template
prompt = ChatPromptTemplate.from_template(
    "Translate this to French: {text}"
)

# Create the model
model = init_chat_model("gpt-4o")

# Chain them with pipe!
chain = prompt | model

# Run the chain
result = chain.invoke({"text": "Hello world"})
```

Q5 FUNDAMENTALS

What are Prompt Templates in LangChain?

Prompt Templates are reusable blueprints for creating prompts. Instead of hardcoding text, you use **placeholders** that get filled in later.

PYTHON

```
from langchain_core.prompts import ChatPromptTemplate

# Create a template with variables
template = ChatPromptTemplate.from_messages([
    ("system", "You are a {role} assistant."),
    ("user", "{question}")
])

# Fill in the blanks
prompt = template.invoke({
    "role": "helpful coding",
    "question": "How do I read a file?"
})
```

WHY USE TEMPLATES?

Templates make your code cleaner and let you reuse the same

Naved Khan

Gen AI Engineer

+ Follow

Q6 FUNDAMENTALS

What are Output Parsers and why use them?

Output Parsers convert the AI's text response into structured data like JSON, lists, or Python objects. Super useful when you need clean data!

PYTHON

```python
from langchain_core.output_parsers import JsonOutputParser
from pydantic import BaseModel

# Define what you want back
class Movie(BaseModel):
    title: str
    year: int

parser = JsonOutputParser(pydantic_object=Movie)

# Chain it: prompt | model | parser
chain = prompt | model | parser

# Get structured data!
result = chain.invoke({"query": "Inception"})
# Returns: {"title": "Inception", "year": 2010}
```

Naved Khan
Gen AI Engineer

+ Follow

Q7 FUNDAMENTALS

What are Document Loaders in LangChain?

Document Loaders help you load data from different sources (PDFs, websites, databases) into LangChain. They're the "import" button for your AI!

- ✓ **TextLoader:** Load .txt files
- ✓ **PyPDFLoader:** Load PDF documents
- ✓ **WebBaseLoader:** Scrape websites
- ✓ **CSVLoader:** Load CSV files

PYTHON

```python
from langchain_community.document_loaders import PyPDFLoader

loader = PyPDFLoader("my_document.pdf")
docs = loader.load()  # Returns list of Documents
```

Naved Khan

Gen AI Engineer

+ Follow

Q8 FUNDAMENTALS

What are Text Splitters and why are they needed?

Text Splitters break large documents into smaller chunks. Why? Because AI models have **token limits** - they can only process so much text at once!

PYTHON

```python
from langchain_text_splitters import
RecursiveCharacterTextSplitter

splitter = RecursiveCharacterTextSplitter(
    chunk_size=1000,      # Max characters per chunk
    chunk_overlap=200     # Overlap between chunks
)

chunks = splitter.split_documents(docs)
```

WHY OVERLAP?

Overlap ensures context isn't lost at chunk boundaries. Imagine cutting a book into pages - overlap keeps the story connected!

Naved Khan
Gen AI Engineer

+ Follow

Q9 FUNDAMENTALS

How do you handle conversation history?

LangChain uses **message objects** to track conversation history. This helps the AI remember what you've been talking about!

PYTHON

```python
from langchain.messages import HumanMessage, AIMessage

# Build conversation history
conversation = [
    HumanMessage("My name is John"),
    AIMessage("Nice to meet you, John!"),
    HumanMessage("What's my name?")
]

# AI will remember: "Your name is John"
response = model.invoke(conversation)
```

MESSAGE TYPES

HumanMessage = user input, AIMessage = model response, SystemMessage = instructions to the AI

Naved Khan

Gen AI Engineer

+ Follow

Q10 FUNDAMENTALS

What is the difference between invoke(), stream(), and batch()?

These are 3 ways to run your LangChain code:

- ☑ **invoke():** Run once, wait for complete result
- ☑ **stream():** Get results token-by-token (like ChatGPT typing)
- ☑ **batch():** Run multiple inputs at once (faster!)

PYTHON

```python
# Single call
result = chain.invoke({"text": "Hello"})

# Streaming (for chat UX)
for chunk in chain.stream({"text": "Hello"}):
    print(chunk, end="")

# Batch processing
results = chain.batch([{"text": "Hi"}, {"text": "Bye"}])
```

Section 2

Agents, Tools & RAG

Questions 11-20

Q11 AGENTS

What are Agents in LangChain?

Agents are AI systems that can **decide which tools to use** based on the task. Unlike chains (fixed steps), agents choose their own path!

> **SIMPLE ANALOGY**
>
> A chain is like following a recipe. An agent is like a chef who decides what to cook based on available ingredients!

PYTHON

```python
from langchain.agents import create_agent

agent = create_agent(
    model="gpt-4o",
    tools=[search_tool, calculator_tool],
    system_prompt="You are helpful assistant"
)

# Agent decides which tool to use!
result = agent.invoke({
    "messages": [{"role": "user", "content": "What's 25 * 4?"}]
})
```

Q12 TOOLS

How do you create custom Tools in LangChain?

Use the **@tool decorator** to turn any Python function into a tool that agents can use!

PYTHON

```python
from langchain.tools import tool

@tool
def get_weather(city: str) -> str:
    """Get weather for a city."""
    # Your logic here
    return f"It's sunny in {city}!"

@tool
def calculate(expression: str) -> str:
    """Calculate a math expression."""
    return str(eval(expression))

# Use in agent
agent = create_agent(
    model="gpt-4o",
    tools=[get_weather, calculate]
)
```

Naved Khan
Gen AI Engineer

+ Follow

Q13 RAG

What is RAG (Retrieval Augmented Generation)?

RAG = Teaching AI with your own data! Instead of relying only on training data, RAG **retrieves relevant info** from your documents first, then generates an answer.

THE RAG PROCESS

1. User asks question → 2. Find relevant docs → 3. Send docs + question to AI → 4. AI answers using YOUR data

✓ **Without RAG:** "Who is the CEO?" → AI guesses or says outdated info

✓ **With RAG:** "Who is the CEO?" → Searches your company docs → Gives accurate answer

Naved Khan
Gen AI Engineer

+ Follow

Q14 RAG

What are Vector Stores and Embeddings?

Embeddings convert text into numbers (vectors). **Vector Stores** save and search these vectors to find similar content.

> **SIMPLE ANALOGY**
>
> Embeddings = Converting books to GPS coordinates. Vector Store = Finding the nearest books to your location!

PYTHON

```python
from langchain_openai import OpenAIEmbeddings
from langchain_qdrant import QdrantVectorStore

# Create embeddings
embeddings = OpenAIEmbeddings()

# Store documents
vector_store = QdrantVectorStore.from_documents(
    documents=docs,
    embedding=embeddings
)

# Search for similar content
```

Naved Khan
Gen AI Engineer

+ Follow

Q15 RAG

How do you implement a basic RAG pipeline?

Here's a complete RAG pipeline in 4 steps:

PYTHON

```python
# 1. Load documents
from langchain_community.document_loaders import PyPDFLoader
docs = PyPDFLoader("data.pdf").load()

# 2. Split into chunks
from langchain_text_splitters import
RecursiveCharacterTextSplitter
chunks =
RecursiveCharacterTextSplitter(chunk_size=1000).split_documents

# 3. Store in vector store
vector_store.add_documents(chunks)

# 4. Create retrieval tool
@tool
def retrieve(query: str):
    """Search knowledge base."""
    return vector_store.similarity_search(query)
```

Q16 MEMORY

What is Memory in LangChain?

Memory allows AI to remember past conversations. Without memory, each message is like talking to someone with amnesia!

- ☑ **Short-term:** Current conversation (stored in messages)
- ☑ **Long-term:** Stored in databases (vector stores)

PYTHON

```python
from langgraph.checkpoint.memory import InMemorySaver

# Create memory
checkpointer = InMemorySaver()

# Agent with memory
agent = create_agent(
    model="gpt-4o",
    tools=[...],
    checkpointer=checkpointer  # Adds memory!
)
```

Naved Khan

Gen AI Engineer

+ Follow

Q17 AGENTS

What is the create_agent() function?

create_agent() is the new standard way to build agents in LangChain v1. It has built-in streaming, memory, and middleware support!

PYTHON

```python
from langchain.agents import create_agent

def send_email(to: str, body: str):
    """Send an email"""
    return f"Email sent to {to}"

agent = create_agent(
    model="gpt-4o",
    tools=[send_email],
    system_prompt="You are an email assistant."
)

result = agent.invoke({
    "messages": [{"role": "user", "content": "Send hi"}]
})
```

Naved Khan
Gen AI Engineer

+ Follow

Q18 TOOLS

How does tool calling work in LangChain?

When you ask a question, the AI **decides if it needs a tool**. If yes, it generates a "tool call" with the function name and arguments.

THE FLOW

1. User asks → 2. AI picks tool → 3. Tool executes → 4. Result sent back → 5. AI generates final answer

PYTHON

```python
# User: "What's the weather in Tokyo?"

# AI generates tool call:
tool_call = {
    "name": "get_weather",
    "args": {"city": "Tokyo"}
}

# Tool executes and returns result
# AI uses result to answer: "It's 22°C in Tokyo"
```

Q19 AGENTS

What is the ReAct pattern?

ReAct = Reasoning + Acting. The AI thinks out loud before taking action. This makes it smarter and easier to debug!

```python
# Question: "Population of France × 2?"

Thought: I need to find France's population first
Action: search("France population")
Observation: 67 million

Thought: Now multiply by 2
Action: calculate("67 * 2")
Observation: 134

Thought: I have the answer
Final Answer: 134 million
```

WHY IT MATTERS

You can see HOW the AI reached its answer, making it easier to fix mistakes!

Naved Khan
Gen AI Engineer

+ Follow

Q20 AGENTS

How do you use bind_tools() with models?

bind_tools() tells a model which tools it can use. The model then knows how to call them!

PYTHON

```python
from pydantic import BaseModel, Field

# Define tool as Pydantic model
class GetWeather(BaseModel):
    """Get weather for a location"""
    location: str = Field(description="City name")

# Bind tools to model
model = init_chat_model("gpt-4o")
model_with_tools = model.bind_tools([GetWeather])

# Model can now use the tool!
response = model_with_tools.invoke("Weather in Paris?")
print(response.tool_calls)  # Shows tool call
```

Naved Khan
Gen AI Engineer

+ Follow

Section 3

Advanced Topics

Questions 21-30

Q21 ADVANCED

What is LangGraph and when should you use it?

LangGraph is LangChain's framework for building complex, stateful AI workflows. Use it when simple chains aren't enough!

☑ **Use LangChain:** Simple tasks, basic chains, quick prototypes

☑ **Use LangGraph:** Complex workflows, multi-agent systems, human-in-loop

KEY FEATURES

Graphs with nodes and edges, state persistence, streaming, cycles and loops, human approval steps

Q22 ADVANCED

How do you implement multi-agent systems?

Multi-agent systems = Multiple specialized agents working together. One agent can call another agent as a tool!

PYTHON

```python
# Create specialized agents
researcher = create_agent(model, tools=[search])
writer = create_agent(model, tools=[write_doc])

# Wrap as tool for main agent
@tool
def do_research(query: str):
    """Research a topic"""
    result = researcher.invoke({...})
    return result["messages"][-1].content

# Main agent coordinates others
supervisor = create_agent(
    model, tools=[do_research, call_writer]
)
```

Naved Khan
Gen AI Engineer

+ Follow

Q23 ADVANCED

What is Middleware in LangChain agents?

Middleware lets you intercept and modify agent behavior. Use cases: logging, rate limiting, safety filters, dynamic prompts.

PYTHON

```python
from langchain.agents.middleware import dynamic_prompt

@dynamic_prompt
def custom_prompt(request):
    """Change prompt based on context"""
    role = request.runtime.context.get("role")
    if role == "admin":
        return "You have full access."
    return "You are a helpful assistant."

agent = create_agent(model, middleware=[custom_prompt])
```

Naved Khan
Gen AI Engineer

+ Follow

Q24 ADVANCED

How do you handle streaming in LangChain?

Streaming shows output token-by-token (like ChatGPT typing). Great for user experience!

PYTHON

```python
# Stream from a chain
for chunk in chain.stream({"input": "Hello"}):
    print(chunk, end="", flush=True)

# Stream from an agent with modes
for event in agent.stream(
    {"messages": [...]},
    stream_mode=["updates", "custom"]
):
    print(event)
```

STREAM MODES

"values" = full state, "updates" = just changes, "custom" = custom events from tools

Naved Khan
Gen AI Engineer

+ Follow

Q25 ADVANCED

What is LangSmith and how is it used?

LangSmith is LangChain's debugging and monitoring platform. It shows you exactly what's happening inside your AI app!

- ✓ **Tracing:** See every step of your chain/agent
- ✓ **Debugging:** Find why something went wrong
- ✓ **Evaluation:** Test your app's quality

PYTHON

```python
import os
os.environ["LANGSMITH_TRACING"] = "true"
os.environ["LANGSMITH_API_KEY"] = "your-key"
# That's it! All runs are now traced
```

Naved Khan
Gen AI Engineer

+ Follow

Q26 ADVANCED

How do you handle errors in LangChain agents?

Agents can fail! Use **error handling** to retry or recover. Always return helpful error messages so AI can try alternatives.

PYTHON

```python
# Handle errors in the tool itself
@tool
def safe_search(query: str):
    try:
        return do_search(query)
    except Exception as e:
        return f"Search failed: {e}"

# Or use ToolStrategy for auto-retry
from langchain.tools import ToolStrategy
strategy = ToolStrategy(
    schema=MyTool,
    handle_errors=(ValueError, TypeError)
)
```

Naved Khan
Gen AI Engineer

+ Follow

Q27 ADVANCED

What are best practices for RAG optimization?

Better RAG = Better answers! Here's how to optimize:

- ☑ **Chunk size:** Experiment with different sizes (500-2000 chars)
- ☑ **Overlap:** 10-20% overlap prevents lost context
- ☑ **Hybrid search:** Combine vector + keyword search
- ☑ **Reranking:** Re-order results by relevance
- ☑ **Metadata:** Filter by date, source, category

Q28 ADVANCED

How do you implement dynamic prompts?

Dynamic prompts change based on context, user, or conversation state. Great for personalization!

PYTHON

```python
from langchain.agents.middleware import dynamic_prompt,
ModelRequest

@dynamic_prompt
def smart_prompt(request: ModelRequest) -> str:
    # Check conversation length
    msg_count = len(request.messages)

    base = "You are a helpful assistant."

    if msg_count > 10:
        base += "\nBe extra concise."

    return base

agent = create_agent(model, middleware=[smart_prompt])
```

Naved Khan
Gen AI Engineer

+ Follow

Q29 ADVANCED

What is human-in-the-loop in LangChain?

Human-in-the-loop = Pause the AI and ask a human for approval before taking critical actions.

> **USE CASES**
>
> Sending emails, making purchases, deleting data, or any action that can't be undone!

- ✓ **interrupt_before:** Pause before specific tools run
- ✓ **interrupt_after:** Pause after tools, review results
- ✓ **Edit state:** Human can modify before resuming

Naved Khan
Gen AI Engineer

+ Follow

Q30 ADVANCED

How do you deploy LangChain apps to production?

Production-ready LangChain apps need these considerations:

- ☑ **API Keys:** Use environment variables, never hardcode
- ☑ **Caching:** Cache embeddings and responses to save costs
- ☑ **Rate Limiting:** Protect against abuse
- ☑ **Monitoring:** Use LangSmith for observability
- ☑ **LangServe:** Deploy chains as REST APIs easily

Naved Khan
Gen AI Engineer

+ Follow

🎉 THAT'S ALL!

Thank You for Reading!

Naved Khan

Gen AI Engineer

Save this for your interview prep. Share with someone who needs it!

💾 Save 🔄 Repost 💬 Comment

+ Follow for More