SDET QA Interview Questions & Answers for AI Applications (RAG, LangGraph, Pytest, RAGAS)

General Concepts & Fundamentals

1. What is RAG (Retrieval-Augmented Generation)?
RAG is an architecture that combines a retriever (to fetch relevant documents from a knowledge base) and a generator (typically an LLM) to produce context-aware answers. It's useful when LLMs need access to updated or domain-specific data.

2. How would you test a RAG-based application?

- **Retriever Testing:** Ensuring relevant documents are retrieved using recall@k, precision@k.
- **Generator Testing:** Verifying factual correctness and fluency of generated output.
- **RAG Evaluation Tools:** Using tools like RAGAS to evaluate answer faithfulness, answer relevance, and context precision.
- **Negative testing:** Injecting irrelevant or adversarial documents and checking hallucination resistance.
- **Ground Truth Comparison:** Comparing output against known answers (if available).

LLM Testing vs. LLM Evaluation

3. What is the difference between LLM Testing and LLM Evaluation?

- **LLM Testing** refers to functional and non-functional aspects (e.g., endpoint testing, retries, latency).
- **LLM Evaluation** assesses the quality of output (faithfulness, coherence) using human feedback or tools like RAGAS.

Pytest + LangGraph

4. How would you write a test for a LangGraph node using Pytest?

```
def test_vector_store_node():
    result = vector_store_node.run("Tell me about LangGraph")
    assert "LangGraph" in result or "framework" in result
```

5. How do you handle state testing in LangGraph workflows?

- Capture state transitions using hooks or logs.
- Validate if expected state changes occur after specific inputs.
- Write unit tests for each node and integration tests for full graphs.

RAGAS & Evaluation Frameworks

6. What is RAGAS and how does it help in testing?
RAGAS provides automatic evaluation of RAG applications with metrics like:

- **Faithfulness** (answer grounded in docs?)
- **Answer Relevance**
- **Context Precision**

7. How do you interpret a low "faithfulness" score in RAGAS?
It means the generated answer is not well-supported by retrieved documents, indicating hallucination or poor grounding.

Test Strategy & QA Mindset

8. How do you perform regression testing for an LLM pipeline?

- Maintain a suite of test prompts with expected outputs.
- Re-run on every model or data update.
- Use similarity metrics or manual review for comparison.

- Tools: DeepEval, TruLens.

9. How do you test non-deterministic LLM responses?

- Set temperature to 0 for deterministic output.
- Use semantic comparison via embedding similarity.
- Use fuzzy assertions or thresholds.

Intermediate & Scenario-Based

10. How would you test a fallback mechanism in a LangGraph chatbot?

- Simulate node failures.
- Check if fallback routes trigger.
- Validate end response is graceful.
- Use Pytest + LangGraph hooks for tracing.

11. How would you test hallucination in an LLM-based system?

- Use known knowledge base.
- Ask out-of-domain questions.
- Check retrievals and generated answers.
- Validate with RAGAS or human review.

12. How do you validate prompt injection vulnerabilities?

- Inject override instructions in input.
- Check if model follows base prompt.
- Apply adversarial testing.

13. What metrics would you monitor for a RAG-based system in production?

- Retrieval Precision/Recall
- Response Latency
- Answer Faithfulness (RAGAS)
- Query Failure Rate
- User Feedback (thumbs up/down)

14. What are common failure points in a LangGraph-based app?

- State transition errors
- Infinite loops
- Node failures without fallback
- Inconsistent state data

15. How do you integrate human feedback in LLM QA?

- Collect thumbs up/down or comments
- Feed into fine-tuning loop or RLHF
- Use to improve retrieval or prompts

End of Document

🔍 General LLM Testing Questions

✅ *1. What are the core challenges in testing LLMs compared to traditional software systems?*

Answer:
Traditional software testing relies on deterministic outputs, but LLMs produce probabilistic responses. Key challenges include:

- ***Non-determinism**: Same input may yield different outputs.*
- ***Lack of ground truth**: Hard to define a "correct" answer in open-ended tasks.*
- ***Bias & hallucinations**: LLMs may generate false or biased information.*
- ***Context sensitivity**: Output changes based on subtle prompt variations.*

✅ *2. How do you define hallucination in the context of LLMs? How can it be detected and mitigated?*

Answer:
Hallucination occurs when an LLM generates output that is factually incorrect or unsupported by its training data or retrieval context.

Detection methods:

- *Comparing output to authoritative ground truth.*
- *Using evaluation tools like **Ragas**, **DeepEval**, or **fact-checking APIs**.*

- *Manual review for high-risk domains.*

Mitigation strategies:

- *Use Retrieval-Augmented Generation (RAG) to ground answers.*
- *Fine-tuning with high-quality domain-specific data.*
- *Prompt engineering and guardrails.*

✅ *3. What are the key components of a RAG architecture, and how would you test each of them?*

Answer:
A typical RAG system consists of:

- ***Retriever**: Fetches relevant documents.*
- ***Generator**: Uses the retrieved documents to generate answers.*

Testing approach:

- ***Retriever**: Use metrics like **Precision@K**, **Recall@K**, and **semantic similarity** to ensure relevance.*
- ***Generator**: Test for **factual correctness**, **context usage**, **fluency**, and **coherence**.*
- *Use tools like **LangChain**, **TruLens**, or **Ragas** to evaluate pipeline integrity.*

✅ *4. How do you test the semantic similarity between a user query and the retrieved documents?*

Answer:

- *Use embedding-based similarity (e.g., cosine similarity with OpenAI or HuggingFace embeddings).*

- *Evaluate with metrics such as:*
    - ***nDCG (Normalized Discounted Cumulative Gain)***
    - ***Mean Average Precision (MAP)***
- *Visualize using tools like **FAISS** or **ChromaDB** to inspect retrieval space.*

✅ *5. How do you ensure the LLM is using the retrieved context during generation in RAG?*

Answer:

- *Inject traceable markers (e.g., document IDs or keywords) in retrieved context.*
- *Analyze generated output for inclusion of those markers.*
- *Use **context-relevance scoring** in tools like **Ragas** or write custom heuristics to compare answer content to the retrieved text.*
- *Use **TruLens**' context attribution features.*

✅ *6. What metrics can be used to evaluate RAG system performance?*

Answer:
Retrieval metrics:

- ***Precision@K**, **Recall@K***
- ***nDCG***

Generation metrics:

- ***BLEU**, **ROUGE**, **METEOR** – surface-level similarity.*
- ***BERTScore** – semantic similarity.*

- ***Faithfulness / Context relevance*** *– via RAGAS or DeepEval.*
- ***User satisfaction (subjective)*** *– through A/B testing or surveys.*

✅ *7. How would you automate the evaluation of thousands of LLM responses to check quality and correctness?*

Answer:

- *Use libraries like* ***DeepEval****,* ***TruLens****, or* ***Ragas****.*
- *Automate pipelines to:*
    - *Evaluate output against expected results.*
    - *Score responses on relevance, coherence, and accuracy.*
    - *Log and store failures for review.*
- *Optionally integrate with CI/CD and create dashboards with tools like* ***LangSmith*** *or* ***Weights & Biases (W&B)****.*

✅ *8. How would you test for bias in LLM responses?*

Answer:

- *Run prompts with demographic variations (e.g., “doctor is a man” vs. “doctor is a woman”).*
- *Use* ***Bias Benchmarking datasets*** *(e.g., StereoSet, CrowS-Pairs).*
- *Analyze outputs for stereotypes or skewed language.*
- *Apply metrics like* ***toxicity scores*** *from Perspective API or similar tools.*

✅ *9. How would you isolate whether a RAG chatbot failure is in retrieval or generation?*

Answer:

- *Inspect the retrieved documents:*
    - *If they are **irrelevant**, the issue is in **retrieval**.*
    - *If documents are **relevant**, but the answer is wrong, it's a **generation issue**.*
- *Manually simulate the input to both the retriever and generator.*
- *Use tracing tools like **LangChain Debug mode** or **TruLens audits**.*

✅ *10. What tools and frameworks can you use for LLM and RAG testing?*

Answer:

- ***RAG Evaluation**: Ragas, DeepEval, TruLens, LangSmith*
- ***Prompt testing**: Promptfoo, ReActEval*
- ***Data generation**: Pandas, Faker, synthetic data tools*
- ***Vector DBs**: ChromaDB, Pinecone, FAISS*
- ***Agents**: CrewAI, AutoGen, LangGraph (for pipeline testing)*
-

RAGAS

🔹 *Beginner Level*

1. ***What is RAGAS?***
   Answer: RAGAS is a framework for evaluating the performance of Retrieval-Augmented Generation (RAG) systems using automated metrics. It

measures the quality of retrieval, generation, and overall system performance without human annotation.

2. ***What are the key components of a RAG system?***
   Answer:
   - ***Retriever: Fetches relevant documents from a knowledge base.***
   - ***Generator: Produces an answer based on the retrieved documents.***
   - ***Evaluator (RAGAS): Assesses how well the retrieval and generation steps performed.***

3. ***Why is evaluation in RAG systems challenging?***
   Answer: Because there is often no single correct answer, and the generated text depends on both the query and retrieved documents. Human evaluation is subjective and costly, so automated, explainable metrics like those in RAGAS help.

🔹 *Intermediate Level*

4. ***What metrics does RAGAS provide?***
   Answer:
   - ***Faithfulness: How grounded the answer is in the retrieved context.***
   - ***Answer Relevance: Relevance of the answer to the question.***
   - ***Context Precision: Whether the retrieved documents are relevant.***
   - ***Context Recall: Whether all relevant documents were retrieved.***
   - ***Context Relevance: Quality of the retrieved context overall.***

5. ***How is faithfulness calculated in RAGAS?***
   Answer: RAGAS uses an LLM to evaluate whether the generated answer is supported or contradicted by the retrieved context. A contradiction lowers the faithfulness score.

6. ***Can RAGAS be used without ground truth answers?***
   Answer: Yes. RAGAS can work in a reference-free mode using only questions,

retrieved context, and generated answers.

🔹 *Advanced Level*

7. ***How does RAGAS integrate with LangChain or other RAG pipelines?***
   Answer: You can plug RAGAS into LangChain pipelines by collecting the RAG outputs (question, contexts, answer) into a `ragas.Dataset`, then run evaluations directly.

8. ***How does RAGAS use LLMs in its evaluation?***
   Answer: RAGAS leverages LLMs (e.g., OpenAI or Hugging Face models) to score answers for faithfulness, relevance, and support, acting like automated human judges.

9. ***How would you use RAGAS in a production monitoring setup?***
   Answer: Periodically sample queries from production, evaluate them with RAGAS, monitor trends in metrics (e.g., declining context recall), and trigger retraining or retriever updates accordingly.

10. ***What are some limitations of RAGAS?***
    Answer:
    - ***Relies on LLMs, which may introduce bias.***
    - ***Evaluation is probabilistic, not deterministic.***
    - ***May not generalize well to domain-specific data without tuning.***

Pytest

✅ *Beginner Level*

1. ***What is Pytest and why is it used?***
2. ***How do you install Pytest?***
3. ***How do you run a specific test file using Pytest?***
4. ***What is the naming convention for test files and functions in Pytest?***

5. ***How do you skip a test in Pytest?***
6. ***How do you mark a test as expected to fail in Pytest?***
7. ***How do you use `assert` in Pytest?***

✅ *Intermediate Level*

1. ***What are fixtures in Pytest? How do you define and use them?***
2. ***What is the difference between `@pytest.fixture(scope="function")` and `scope="module"`?***
3. ***How do you parametrize a test in Pytest?***
4. ***What are Pytest markers? How do you create custom markers?***
5. ***How do you run only tests marked with a specific marker (e.g., smoke tests)?***
6. ***How can you make Pytest stop after the first failure?***
7. ***How do you capture logs or standard output in Pytest?***

✅ *Advanced Level*

1. ***How do you implement hooks using `conftest.py` in Pytest?***
2. ***How do you generate and customize test reports using Pytest plugins (e.g., `pytest-html`, `pytest-allure`)?***
3. ***Explain how to integrate Pytest with CI/CD tools like Jenkins or GitHub Actions.***
4. ***How do you dynamically create tests at runtime using Pytest?***
5. ***How does Pytest handle test discovery and what can you do to customize it?***
6. ***What are some common Pytest plugins you have used and for what purpose?***

7. ***How do you handle test dependencies in Pytest?***

🧠 *Tricky / Scenario-Based Questions*

1. ***You have a test that randomly fails. How do you debug and isolate it using Pytest?***
2. ***How would you test a REST API using Pytest and requests?***
3. ***If two test files require similar setup logic, how would you refactor it using fixtures?***
4. ***How do you write reusable fixtures across multiple test modules?***
5. ***How do you test code that interacts with a database using Pytest?***

LangGraph?

🔹 *1. What is LangGraph?*

Answer:
LangGraph is a Python framework built on top of LangChain that enables you to create stateful, multi-agent workflows using a graph-based architecture. It allows LLM applications to be structured as graphs of nodes where each node can represent an agent, tool, or function.

🔹 *2. What are the key components of LangGraph?*

Answer:

- ***Nodes: Units of execution (e.g., agents, tools, prompts).***
- ***Edges: Transitions between nodes based on outputs.***
- ***State: Carries information (e.g., memory, history) across the graph.***
- ***Graph Definition: The overall workflow defining how nodes and edges connect.***

🔹 *3. How does LangGraph differ from LangChain?*

Answer:

- ***LangChain is primarily a framework for chaining LLM calls.***
- ***LangGraph adds control flow and state management, enabling the modeling of more complex interactions like loops, conditionals, or retries using graph-based workflows.***

🔹 *4. What is a "state" in LangGraph? Why is it important?*

Answer:
A state in LangGraph is a shared object (usually a dictionary) that carries context or memory through different nodes in the graph. It allows nodes to pass information and maintain context between transitions.

🔹 *5. What is the use of conditional branching in LangGraph?*

Answer:
Conditional branching lets you decide which node to visit next based on the result of a current node. For example, after an LLM generates a response, LangGraph can decide whether to:

- ***go to a human feedback node***
- ***retry the generation***
- ***end the workflow***

🔹 *6. How do you define a LangGraph node?*

Answer:
A node is typically defined as a function or an agent. For example:

python

CopyEdit

```
@graph.node

def answer_question(state):

    # some logic

    return updated_state
```

🔹 *7. Can LangGraph handle loops or retries?*

Answer:
Yes, LangGraph allows cycles in its graph. You can define edges that loop back to earlier nodes to enable retry logic, clarifications, or feedback loops.

🔹 *8. Give an example use case where LangGraph is more suitable than LangChain.*

Answer:
A multi-agent system where one agent extracts information and another validates it. Based on validation results, the system may loop back to re-query the source. This dynamic, state-aware flow is hard in LangChain alone, but natural in LangGraph.

🔹 *9. What testing strategies would you use for a LangGraph application?*

Answer:

- ***Unit test each node independently.***
- ***Mock state transitions and test how state evolves.***
- ***Integration tests for complete graph execution.***

- *RAGAS or DeepEval to evaluate LLM responses if applicable.*

🔹 *10. What Python libraries commonly integrate with LangGraph?*

Answer:

- *LangChain (for agents, tools)*
- *OpenAI / Anthropic APIs (for LLMs)*
- *Pydantic (for state validation)*
- *RAGAS / DeepEval (for response evaluation)*
-

Prompt engineering?

⚙️ *1. What is prompt engineering?*

Answer:
Prompt engineering is the practice of crafting effective inputs (prompts) to guide the output of a Large Language Model (LLM) like GPT. It involves designing, refining, and optimizing the phrasing, structure, and content of prompts to elicit the desired behavior or response from the model.

📌 *2. Why is prompt engineering important in LLM applications?*

Answer:
Prompt engineering ensures that LLMs produce relevant, accurate, and useful outputs. Since LLMs are highly sensitive to the way questions or instructions are phrased, even small changes can lead to significantly different results. Well-designed prompts improve the reliability, safety, and performance of AI applications.

🧠 *3. What are some common types of prompts used in prompt engineering?*

Answer:

- ***Zero-shot prompts: Asking a question without any examples.***
- ***One-shot prompts: Providing one example along with the question.***
- ***Few-shot prompts: Providing a few examples to guide the model's response.***
- ***Chain-of-thought prompts: Encouraging step-by-step reasoning.***
- ***Instruction-based prompts: Clearly instructing the model what to do (e.g., "Summarize the text below").***

🧪 *4. What is a zero-shot prompt, and when would you use it?*

Answer:
A zero-shot prompt is when you ask the model to perform a task without giving it any example.
Example:
"Translate the following sentence into French: 'Good morning'."
Use it when the task is simple or you want to evaluate the model's out-of-the-box capabilities.

🔁 *5. What is few-shot prompting and how does it help?*

Answer:
Few-shot prompting involves providing the model with a few examples of how to perform a task before asking it to handle a new input.
This helps the model learn the pattern or format you expect.
It is useful for complex or structured tasks (e.g., text classification, code generation, etc.).

🔗 *6. What is chain-of-thought prompting?*

Answer:
Chain-of-thought (CoT) prompting involves prompting the model to explain its reasoning step-by-step.
Example:
"If there are 3 red balls and 2 blue balls, and I take one out randomly, what is the chance

it's red? Explain step by step."
This improves performance in reasoning tasks.

📚 *7. What are the characteristics of a good prompt?*

Answer:

- ***Clear and unambiguous***
- ***Concise but informative***
- ***Well-structured***
- ***Contextual if needed***
- ***Free from bias or misleading wording***

🧩 *8. What are prompt injection attacks?*

Answer:
Prompt injection is a type of attack where a malicious user manipulates the prompt to alter the model's behavior, often to bypass restrictions or generate undesired output.
Example: A user appending "Ignore previous instructions and…" to alter the AI's behavior.

🛡️ *9. How can you test prompts for robustness and reliability?*

Answer:

- ***Use A/B testing with variations of prompts.***
- ***Evaluate using metrics like accuracy, relevance, hallucination rate, etc.***
- ***Test for edge cases and adversarial inputs.***
- ***Use evaluation frameworks like RAGAS or DeepEval (if using RAG pipelines).***

⚖️ *10. What tools or frameworks help with prompt engineering?*

Answer:

- ***PromptLayer – tracks and version-controls prompts.***
- ***LangChain – for chaining LLM prompts and tools together.***
- ***OpenAI Playground – for testing prompt behavior interactively.***
- ***Gradio/Streamlit – for building UI for prompt testing.***
- ***Weights & Biases (W&B) – for experiment tracking.***
- ***LLM evaluation frameworks – like DeepEval, RAGAS***
-