✅ SECTION 1: AI / LLM BASICS — Questions & Answers

---

1. What is a Large Language Model (LLM) and how does it work?

Answer:
A Large Language Model (LLM) is a type of deep learning model, often based on the Transformer architecture, trained on massive datasets of text. It predicts the next word (or token) in a sequence using learned patterns in data. By doing this at scale, LLMs can generate coherent and context-aware text.

---

2. What is the difference between generative AI and traditional rule-based NLP?

Answer:
Traditional NLP uses handcrafted rules and symbolic approaches (like regex, grammar parsers), while generative AI relies on data-driven models (like LLMs) to generate and understand text. Generative AI can generalize better but lacks interpretability.

---

3. How do tokenization and context windows affect LLM responses?

Answer:
Tokenization breaks text into smaller pieces (tokens), which are the model's input units. A context window determines how many tokens the model can see at once (e.g., 2048 or 8192). Larger windows allow for more context and better understanding but require more memory and compute.

---

4. What are the differences between zero-shot, one-shot, and few-shot learning in LLMs?

Answer:

- **Zero-shot**: The model performs a task with only instructions, no examples.
- **One-shot**: The model is given one example in the prompt.

- **Few-shot**: The model is given a few examples to learn from in the prompt.
  This controls how much guidance you give the LLM before it performs a task.

---

5. Explain the importance of embeddings in LLM-based applications.

Answer:
Embeddings convert text into high-dimensional numerical vectors that capture semantic meaning. They are essential for tasks like similarity search, document clustering, and RAG (Retrieval-Augmented Generation), enabling models to "understand" relationships between words or phrases.

---

6. What are some limitations of LLMs when used in production systems?

Answer:

- **Hallucinations** (made-up facts)
- **Biases** from training data
- **High latency and compute cost**
- **Lack of explainability**
- **Difficulty in fine control over output**

---

7. What is temperature in an LLM API and how does it influence outputs?

Answer:
Temperature is a parameter that controls the randomness of output.

- **Low temperature (e.g., 0.2)** = more deterministic and focused responses.
- **High temperature (e.g., 0.9)** = more diverse and creative responses.

---

8. Describe use cases where LLMs are most effective and where they're not.

Answer:
Effective: Text summarization, code generation, creative writing, customer support bots.
Not effective: Real-time systems, high-accuracy math, multi-step reasoning requiring verified data, or highly regulated domains.

---

9. How is fine-tuning different from prompt engineering?

Answer:

- **Fine-tuning** updates the model weights using labeled data.
- **Prompt engineering** structures the input prompts to guide the model without modifying its weights.
  Prompting is faster and cheaper, while fine-tuning provides better task-specific control.

---

10. What are common reasons an LLM may return incorrect or biased outputs?

Answer:

- Poor or biased training data
- Misleading or vague prompts
- Lack of recent knowledge (for static models)
- Ambiguities in language
- Token limit causing truncation of important context

✅ SECTION 2: RAG (Retrieval-Augmented Generation) — Questions & Answers

---

1. What is RAG (Retrieval-Augmented Generation)?

Answer:
RAG is an architecture that combines a retriever (which fetches relevant documents) and a generator (LLM) that uses those documents to generate context-aware responses. It enhances the model's knowledge by pulling from external sources at runtime.

---

2. Why is RAG useful compared to using an LLM alone?

Answer:
LLMs have a fixed knowledge base from training data and can't access recent or proprietary information. RAG allows dynamic, real-time access to relevant data, improving factual accuracy and reducing hallucinations.

---

3. What components make up a typical RAG system?

Answer:

1. **Retriever** (e.g., vector database like FAISS, ChromaDB)
2. **Embedding model** (e.g., SentenceTransformers, OpenAI)
3. **LLM generator** (e.g., GPT, Mistral)
4. **Document store** or corpus

---

4. What is the role of a vector database in RAG?

Answer:
A vector database stores document embeddings and enables similarity-based retrieval. When a query is made, it's converted into an embedding, and similar vectors (documents) are fetched to provide the LLM with relevant context.

---

5. How does retrieval improve the performance of generation in RAG?

Answer:
By supplying contextually relevant information, retrieval grounds the LLM in external facts, improving response quality, reducing hallucinations, and enabling domain-specific Q&A without retraining the model.

---

6. What are potential failure points in a RAG pipeline?

Answer:

- **Poor embedding quality**
- **Irrelevant document retrieval**
- **Latency during large-scale retrieval**
- **Overload or hallucination despite retrieved context**
- **Token limits (context truncation)**

---

7. How do you evaluate the quality of retrieved documents in RAG?

Answer:
Using metrics like **Precision@k**, **Recall@k**, or semantic similarity scores between the query and retrieved results. Human validation is also critical for qualitative feedback.

---

8. How does chunking strategy affect RAG performance?

Answer:
Text is split into chunks before being embedded and stored. If chunks are too small, context is lost. If too large, important parts may be missed or truncated in generation. Balanced chunk size ensures better retrieval relevance.

---

9. What are common methods for embedding documents in RAG systems?

Answer:

- **OpenAI Embeddings (text-embedding-ada-002)**
- **HuggingFace models (e.g., `all-MiniLM-L6-v2`)**
- **Google's Universal Sentence Encoder**
  Embeddings convert text into high-dimensional vectors for similarity comparison.

---

10. What kind of testing is required for a RAG system?

Answer:

- **Unit testing** of retrieval and embedding logic
- **Integration testing** of LLM + retriever
- **Latency & load testing** for scaling
- **Evaluation metrics** (precision, recall, faithfulness)
- **User-based testing** for relevance and satisfaction

✅ SECTION 3: LLM Evaluation / RAG Evaluation & Testing — Questions & Answers

---

1. What is the difference between testing and evaluation in LLM-based systems?

Answer:

- **Testing** focuses on verifying system behavior (e.g., correctness, latency, errors).
- **Evaluation** assesses model quality (e.g., relevance, coherence, factual accuracy) using qualitative and quantitative metrics.

---

2. What are some key metrics used to evaluate LLM output?

Answer:

- **BLEU**, **ROUGE** (for summarization/translation)
- **Exact Match (EM)**, **F1 Score** (for QA)
- **Faithfulness**, **Helpfulness**, **Relevance**, **Toxicity**
- **Embedding-based scores** like cosine similarity
- **Human evaluations** (manual scoring)

---

3. What is hallucination in LLMs, and how can it be tested?

Answer:
Hallucination refers to confident but factually incorrect outputs.
To test:

- Use factuality benchmarks or gold standards
- Check generated content against a trusted knowledge base

- Run automated truthfulness checks with tools like RAGAS or DeepEval

---

4. What tools are used to evaluate RAG pipelines?

Answer:

- **RAGAS** (RAG-specific evaluation: Faithfulness, Answer Relevancy)
- **LangChain's eval modules**
- **DeepEval** (Python-based open-source testing framework)
- **TruLens**, **Phoenix**
- Manual spot checks

---

5. How can you automate evaluation of LLM output?

Answer:

- Compare generated outputs with expected results using eval scripts
- Use embedding similarity for semantic match
- Use rule-based or LLM-as-a-judge techniques
- Integrate evaluation in CI/CD using tools like DeepEval or RAGAS

---

6. What is ‘Ground Truth’ and how does it relate to LLM testing?

Answer:
Ground truth refers to the correct or expected answer used as a benchmark. LLM outputs are compared against it to determine correctness, especially in QA or summarization tasks.

---

7. How do you perform regression testing on LLM systems?

Answer:

- Store expected outputs for common prompts
- Run them on every model update
- Compare current vs previous outputs
- Flag differences using string or semantic similarity checks
- Use snapshot testing with version control

---

8. How do you evaluate relevance in retrieved documents in a RAG setup?

Answer:

- Use **Recall@k**, **Precision@k**
- Compute embedding similarity between query and document
- Human ratings (Good/Fair/Poor relevance)
- Relevance scoring tools in RAGAS or custom logic

---

9. How do you test latency and performance of a RAG system?

Answer:

- Benchmark response time for queries under load
- Measure latency for each component: embedding, retrieval, generation
- Use load testing tools (e.g., JMeter, Locust)
- Monitor timeouts and fallbacks

---

10. How can you test prompt robustness in LLMs?

Answer:

- Use **fuzz testing** (alter prompt structure or phrasing)
- Validate consistency of answers across paraphrased prompts
- Test edge cases and adversarial inputs
- Apply structured prompt evaluation

✅ SECTION 4: SDET-Specific Testing of AI/LLM Applications — Questions & Answers

---

1. As an SDET, how would you test the integration between a retriever and a generator in a RAG pipeline?

Answer:

- Validate retrieved documents using `Precision@k`, `Recall@k`
- Check if context is passed correctly to the LLM
- Test different query types (edge cases, rephrased inputs)
- Ensure generated outputs are grounded in retrieved facts

---

2. How do you apply test automation principles in LLM-based applications?

Answer:

- Create test cases with fixed prompts and expected responses
- Automate evaluations using frameworks like **DeepEval** or **RAGAS**
- Use assertions for accuracy, response length, latency, etc.
- Integrate tests into CI/CD pipelines for model version tracking

---

3. What are common challenges in testing LLM applications?

Answer:

- Non-deterministic outputs
- Evaluating correctness without exact matches

- Prompt sensitivity
- Lack of traditional pass/fail criteria
- Inconsistent responses for the same input

---

4. How can you test LLM prompt engineering strategies?

Answer:

- Test various versions of the same prompt (e.g., few-shot, zero-shot)
- Evaluate consistency, helpfulness, and completeness
- Use LLM-as-a-judge to compare multiple prompt outputs
- Automate comparison using similarity metrics

---

5. What logging or observability would you implement for AI pipelines?

Answer:

- Log inputs, retrieved documents, and generated outputs
- Include token usage, latency, and response score
- Use monitoring dashboards for tracking metrics (e.g., Prometheus + Grafana)
- Capture error cases for retraining or debugging

---

6. How do you test fallbacks in AI systems (e.g., when retrieval fails)?

Answer:

- Simulate failures by mocking the retriever or DB layer

- Ensure fallback logic kicks in (e.g., default response or LLM-only generation)
- Verify logs and user feedback are correctly updated
- Add tests for fallback accuracy and completeness

---

7. What's your approach to data validation in LLM-based systems?

Answer:

- Validate structure and completeness of input data
- Check if documents are properly chunked and embedded
- Ensure metadata (e.g., document ID, source) is correct
- Confirm no duplicate or empty embeddings exist

---

8. How do you test multilingual support in LLM or RAG apps?

Answer:

- Provide queries in different languages
- Validate retrieved documents are language-matched
- Use translation APIs to check consistency
- Ensure token limits are respected for verbose languages

---

9. How do you ensure scalability and performance in a RAG system?

Answer:

- Load test retrieval and generation independently

- Use caching to reduce redundant generation
- Monitor vector search latency
- Benchmark under high concurrency using tools like Locust or Artillery

---

10. How do you version control and test model updates as an SDET?

Answer:

- Store past outputs and evaluate diffs
- Track performance of each model version using key metrics
- Run regression tests on a representative test suite
- Maintain baseline expected results for prompt+input pairs

✅ SECTION 5: Evaluation Tools & Frameworks (DeepEval, RAGAS, TruLens, etc.) — Questions & Answers

---

1. What is RAGAS, and how is it used in LLM evaluation?

Answer:
RAGAS (Retrieval-Augmented Generation Assessment Score) is an open-source evaluation framework for RAG pipelines. It assesses:

- **Faithfulness** (truthfulness of the answer to the retrieved context)
- **Answer Relevancy**
- **Context Precision and Recall**
  It can automate scoring of RAG pipelines using embeddings and LLMs.

---

2. What is DeepEval, and how is it useful for QA?

Answer:
DeepEval is a Python-based test framework for evaluating LLMs. It allows:

- Assertion-based testing for LLM outputs
- Evaluation of **semantic similarity**, **toxicity**, **grammar**, etc.
- Test case creation with ground truth and metadata
  Useful in CI pipelines to test LLM behaviors.

---

3. How is TruLens used in LLM observability?

Answer:
TruLens provides **evaluation and traceability** for LLM apps by:

- Logging prompts, responses, intermediate steps

- Evaluating **faithfulness**, **relevance**, **toxicity**

- Visualizing evaluation scores
  Helps understand and debug LLM pipelines.

---

4. What does LangSmith offer for LLM testing?

Answer:
LangSmith is a monitoring and evaluation platform by LangChain. It enables:

- Tracing of chains and agents

- Capturing inputs, outputs, metadata

- Manual or automatic grading

- Run comparisons between LLM versions

---

5. Can these tools (RAGAS, DeepEval, TruLens) be integrated in CI/CD pipelines?

Answer:
Yes. All support Python APIs or CLI tools, allowing you to:

- Add test runs in GitHub Actions, Jenkins, etc.

- Generate and store reports

- Fail builds on low scores or regressions

---

6. How does DeepEval differ from unit testing frameworks like Pytest?

Answer:
Pytest tests logic with exact matches and hard-coded asserts.
DeepEval supports:

- Fuzzy, semantic, or embedding-based matching
- Grading with LLMs
- Task-specific test cases (QA, summarization, classification)

---

7. What metrics does RAGAS provide that traditional tests don't?

Answer:

- **Faithfulness Score** – ensures answers come from retrieved context
- **Context Recall/Precision** – checks if right documents were retrieved
- **Answer Relevancy** – evaluates response usefulness
  These go beyond pass/fail checks and assess semantic quality.

---

8. Which framework is best for evaluating factual correctness in LLM output?

Answer:

- **RAGAS** (if retrieval context is used)
- **TruLens** or **DeepEval** with "faithfulness" metric
- LLM-as-a-judge methods (e.g., GPT-4 scoring output for factuality)

---

9. What tools can evaluate hallucination in generated content?

Answer:

- **TruLens** and **RAGAS** for factual grounding
- **DeepEval** with "faithfulness" metric

- Manual checking or use of external APIs for truth validation (e.g., Wikipedia or search)

---

10. How do these tools support continuous evaluation as the model evolves?

Answer:
They support:

- Versioning outputs
- Logging prompt-response pairs
- Comparison dashboards (old vs new scores)
- Regression test suites
  This ensures changes don’t degrade quality over time.

✅ Scenario-Based Interview Questions & Answers (AI/LLM/RAG Testing)

🔹 1. Scenario:

You're testing a customer support chatbot. You notice changing the prompt from *"How can I reset my password?"* to *"I forgot my password, what do I do?"* gives very different results.
Q: How do you handle this prompt sensitivity issue?

Answer:

- Group similar intents using paraphrases.
- Test multiple rewordings of each prompt.
- Use semantic similarity to evaluate consistency.
- Report and refine prompt templates to reduce variability.
- Suggest prompt engineering strategies like few-shot examples or system instructions.

🔹 2. Scenario:

Your RAG app gives a correct answer but references a source that doesn't contain that information.
Q: How do you identify and address hallucination?

Answer:

- Use **faithfulness scoring** from tools like **TruLens** or **RAGAS**.
- Check if answer text is found in the retrieved documents.
- Add logging to trace context vs generated answer.
- Update retrieval logic or fine-tune the model if hallucination is frequent.

🔹 3. Scenario:

Your generative AI model gives different answers for the same input during regression testing.
Q: How do you manage non-determinism in LLM testing?

Answer:

- Set a fixed `temperature=0` for deterministic output.
- Compare using **semantic similarity** instead of exact match.
- Maintain golden test cases with expected behavior.
- Use snapshot testing with tolerances for acceptable variation.

🔹 4. Scenario:

You added a new model version to production and users report degraded answers.
Q: How would you conduct a model regression test as an SDET?

Answer:

- Run both old and new versions on a benchmark dataset.
- Compare outputs using evaluation metrics (BLEU, ROUGE, faithfulness, relevancy).
- Use **LangSmith** or **TruLens** to trace and compare outputs.
- Generate a diff report and flag degraded cases.

🔹 5. Scenario:

A product search LLM is returning outdated or irrelevant results.
Q: What would your test strategy be?

Answer:

- Validate vector embeddings and recency of indexed data.

- Test retrieval precision and recall for new vs old products.
- Include edge-case queries and category-specific prompts.
- Trigger full re-indexing and compare results.

🔹 6. Scenario:

The chatbot gives polite but vague answers for critical issues like “account locked.”
Q: How would you improve this via testing and feedback?

Answer:

- Write test cases for high-priority intents.
- Use `expected_answer_quality` criteria: specificity, actionability.
- Evaluate using **LLM-as-a-judge** or human scoring.
- Refine prompts or provide structured retrieval context.

🔹 7. Scenario:

Your multilingual LLM gives poor results in non-English inputs.
Q: How would you test and improve it?

Answer:

- Provide language-specific prompts and validate context coverage.
- Test token counts and encoding across languages.
- Use translation to back-translate and compare answer quality.
- Report metrics per language: accuracy, coverage, fluency.

🔹 8. Scenario:

Users report the chatbot gives different results for capitalized vs lowercase queries.
Q: How do you handle such format-related inconsistencies?

Answer:

- Add test cases with casing variations (e.g., `reset PASSWORD` vs `Reset password`).
- Normalize input (lowercase or sentence case) before processing.
- Apply prompt preprocessing step in the pipeline.
- Track response quality across different formatting styles.

🔹 9. Scenario:

You want to verify that a generated summary captures key facts from a long input.
Q: What's your approach as an SDET?

Answer:

- Compare summary against key fact checklist.
- Use tools like **DeepEval** to measure semantic coverage.
- Add tests for hallucination and missing entities.
- Evaluate with human-in-the-loop or LLM-based QA extraction.

🔹 10. Scenario:

Your RAG system fails when internet connectivity is cut (retrieval goes down).
Q: How do you test fallback and resilience in this case?

Answer:

- Simulate retrieval failure (mock vector store API).

- Assert fallback response or LLM-only generation is triggered.
- Log fallback event and return a user-friendly message.
- Add tests to ensure graceful degradation and retry mechanisms.

11. Scenario:

Users often enter prompts with spelling mistakes or typos, e.g., “How do I resset my pasword?” instead of “reset my password.”

Q: How would you test the AI/LLM system’s handling of misspelled prompts?

Answer:

- Prepare test cases with common and random misspellings and typos for key intents.
- Check if the model or retrieval layer can still understand and respond accurately.
- Test if autocorrect or fuzzy matching modules are integrated and effective.
- Use semantic similarity or intent classification to verify robustness.
- Add negative tests with ambiguous misspellings to check fallback behavior.
- Log misinterpretations and feed back results to improve spell correction or prompt engineering.