

AI Engineer Interview Kit

Free sample · 2026 edition · stackchief.com

Questions, worked answers and a 2-week prep plan for AI/LLM engineering interviews

This sample has 12 of the kit's 120 questions, one from each chapter, each with the worked answer, what the interviewer is testing and the follow-ups they are likely to ask.

The full kit adds the other 108 questions and a 14-day prep plan:

stackchief.com/products/ai-engineer-interview-kit

What the full kit covers

1. LLM fundamentals for application engineers (10 questions)
2. RAG and retrieval (10 questions)
3. Evaluation and testing (10 questions)
4. Inference, latency and cost (10 questions)
5. Agents and tool use (10 questions)
6. Prompting and context engineering (10 questions)
7. LLM system design (10 questions)
8. Behavioral questions for AI roles (10 questions)
9. Fine-tuning and training data (10 questions)
10. Safety, security and guardrails (10 questions)
11. Multimodal: documents, images and voice (10 questions)
12. Coding rounds, take-homes and AI-assisted engineering (10 questions)

1. LLM fundamentals for application engineers

1.1 What is a token, and why does tokenization matter in practice?

Junior · AI lab, AI product startup, Big tech, Enterprise

What they're testing: Connecting tokenization to cost, limits and odd model behaviour.

Models read and write tokens, not characters or words. Tokenizers (commonly byte-pair encoding variants) split text into frequent chunks: common English words are often one token, while rare words, code identifiers, numbers and many non-English scripts take several.

Practical consequences: pricing and context limits are counted in tokens, so the same content can cost more in some languages or formats; JSON with long keys and whitespace wastes tokens; and tasks at the character level (counting letters, reversing strings, some arithmetic) are harder because the model doesn't see individual characters.

Use the provider's tokenizer or token-counting endpoint to measure real inputs instead of estimating from word counts.

Likely follow-ups:

- Why might a model struggle to count the letters in a word?

2. RAG and retrieval

2.1 Your RAG assistant gives fluent, confident answers that are wrong. How do you find out whether retrieval or generation is at fault?

Mid · AI product startup, Big tech, Enterprise

What they're testing: Whether you separate the pipeline into measurable stages instead of tweaking the prompt.

Split the failure in two with a small labelled set: 30 to 100 real questions where you know which document (or passage) holds the answer.

First, measure retrieval alone. For each question, check whether a correct passage appears in the top-k results (recall@k). If it doesn't, the generator never had a chance and prompt changes won't help.

Second, measure generation given perfect context. Feed the known-correct passage directly and grade the answer. If answers are still wrong, the problem is the prompt, the model, or the context format (passages too long, conflicting, missing metadata such as dates).

Then read the failures by hand and bucket them: the right chunk wasn't retrieved, it was retrieved but ranked below k, it was retrieved but cut off by chunking, or it was there and the model ignored or contradicted it. Each bucket has a different fix: embeddings or hybrid search, reranking, chunking, or grounding instructions and citations.

Finish by saying you'd keep that labelled set as a regression suite, so every later change is measured against it.

Likely follow-ups:

- How would you build the labelled set if you have no users yet?
- What k would you evaluate at, and why does it depend on the context budget?

3. Evaluation and testing

3.1 Why can't you test an LLM feature the way you unit-test ordinary code?

Junior · AI lab, AI product startup, Big tech, Enterprise

What they're testing: Understanding non-determinism and open-ended outputs, and what replaces exact-match assertions.

Outputs are open-ended and can vary between runs (sampling, model updates), so there's rarely one correct string to assert against. Two different answers can both be right, and a fluent answer can be wrong.

You still test, but you change what you assert. Deterministic parts (parsing, tool-call schemas, routing, retrieval filters) get normal unit tests. Model outputs get property checks (valid JSON, required fields present, no banned content, cites a source) and graded evaluations over a dataset rather than a single example.

Results become rates, not pass/fail: "92% of the golden set graded correct". Compare rates between versions, and look at the individual cases that changed, because an average can hide a new class of failure.

Likely follow-ups:

- Would you set temperature to 0 for tests? What does that fix and what doesn't it fix?

4. Inference, latency and cost

4.1 Walk me through estimating the monthly model cost of a support chatbot.

What goes into the calculation?

Junior · AI product startup, Big tech, Enterprise

What they're testing: Setting up a cost model from first principles with input and output tokens priced separately.

Write the formula first: requests per month $\times$ (input tokens per request $\times$ input price + output tokens per request $\times$ output price). Providers price input and output tokens separately, and output tokens usually cost several times more per token.

Fill in assumptions out loud. For example: 10,000 conversations a day, 4 turns each, so 1.2 million requests a month. Input per request includes the system prompt, retrieved passages and the conversation so far, so it grows with each turn; say 3,000 tokens on average. Output, say 300 tokens.

That's 3.6 billion input tokens and 360 million output tokens a month. Multiply by the per-million-token prices of the model you're considering, and note that input dominates here because of the long context.

Then name the levers: shorter system prompts, fewer or shorter retrieved passages, summarising old turns, prompt caching for the repeated prefix, and routing easy questions to a smaller model.

Likely follow-ups:

- How does prompt caching change this estimate?
- Which assumption would you validate first, and how?

5. Agents and tool use

5.1 How does tool calling work, from the model's point of view and from your application's?

Junior · AI lab, AI product startup, Big tech, Enterprise

What they're testing: Knowing that the model only proposes calls and the application executes them.

You describe each tool to the model with a name, a description and a JSON schema for its arguments. When the model decides a tool would help, it returns a structured tool call (the name plus arguments) instead of, or alongside, text.

The model doesn't execute anything. Your application validates the arguments, runs the tool with its own credentials, and sends the result back as a new message. The model then continues, possibly calling more tools, until it produces a final answer.

That split matters for safety and reliability: your code decides what's actually allowed, handles errors and timeouts, and can require confirmation before any action with side effects.

Likely follow-ups:

- What makes a tool description good or bad?
- How would you handle a tool call with invalid arguments?

6. Prompting and context engineering

6.1 What goes into a good system prompt for a production feature?

Junior · AI product startup, Big tech, Enterprise

What they're testing: Structure and specificity rather than clever wording.

State the role and the task concretely: who the users are, what the assistant should help with, and what it must not do. Vague instructions such as "be helpful and accurate" add little.

Give the rules the product depends on: output format, tone, length, when to refuse or escalate, how to handle missing information (say so rather than guess), and how to use any provided context or tools.

Include a few short examples for behaviour that is hard to describe, and separate instructions from data with clear delimiters or sections so the model doesn't confuse a document's content with your instructions.

Then treat it like code: keep it in version control, change one thing at a time, and run the eval set on every change.

Likely follow-ups:

- What belongs in the system prompt versus the user message?

7. LLM system design

7.1 How do you structure your answer in an LLM system design interview?

Mid · AI product startup, Big tech, Enterprise

What they're testing: A repeatable framework that covers the AI-specific concerns.

Clarify first: users and use cases, traffic and latency targets, quality bar and the cost of a wrong answer, data sources and their sensitivity, and whether the system only answers or also takes actions.

Sketch the simplest end-to-end design: client, API, orchestration, model calls, retrieval or tools, storage, and logging. Say which model tier you'd start with and why.

Then go deep on what makes LLM systems different: how you'll evaluate quality before and after launch, the cost per request and how it scales, latency budget and streaming, guardrails and safety, failure handling when the model or a tool is wrong or slow, and how prompts and models are versioned and rolled out.

Close with trade-offs you made and what you'd do with more time. Interviewers reward explicit trade-offs more than an exhaustive component list.

Likely follow-ups:

- Which clarifying question changes the design the most?

8. Behavioral questions for AI roles

8.1 Tell me about an AI project you built. What would you do differently?

Junior · AI lab, AI product startup, Big tech, Enterprise

What they're testing: Depth of ownership and honest reflection, not the project's novelty.

Pick a project you can go deep on, even a small one. Cover the problem and who it was for, the design you chose and one alternative you rejected, how you knew it worked (an eval, a metric, user feedback), and one concrete number.

The "differently" part carries most of the signal. Name a real decision you'd change and why: you'd have built the eval set before iterating on prompts, started with a simpler workflow instead of an agent, or logged traces from day one.

Avoid listing technologies. Interviewers follow up on the decisions, so expect "why that chunk size?" or "how did you measure that?" and have the answers ready.

Likely follow-ups:

- What was the hardest bug?
- How did you decide it was good enough to ship?

9. Fine-tuning and training data

9.1 How do you decide between prompting, retrieval and fine-tuning for a new feature?

Junior · AI product startup, Big tech, Enterprise

What they're testing: Knowing what each technique changes and picking the cheapest one that solves the actual problem.

Start from the failure you're trying to fix. If the model lacks facts, especially facts that change or are private, that's a retrieval problem: fine-tuning is a poor way to inject knowledge, because it's expensive to update and the model can still confabulate around what it learned.

If the model has the knowledge but produces the wrong format, tone or behaviour, try better instructions and a few examples first. That's fast to iterate and easy to roll back.

Fine-tune when prompting has plateaued on a measured eval, when the behaviour is consistent enough to learn from examples (a strict output schema, a house style, a narrow classification), or when you need a smaller, cheaper model to match a larger one on one task.

Say what you'd need before starting: an eval set that shows the gap, a few hundred to a few thousand high-quality examples, and a plan to retrain when the base model or the task changes.

Likely follow-ups:

- What would convince you to stop fine-tuning and go back to prompting?
- Can you combine fine-tuning and retrieval? When would you?

10. Safety, security and guardrails

10.1 What is prompt injection, and how is it different from a jailbreak?

Junior · AI product startup, Big tech, Enterprise

What they're testing: Clear definitions and why indirect injection matters for real applications.

A jailbreak is a user trying to get the model to break its own rules, for example to produce content the provider or the application forbids. The attacker is the person typing.

Prompt injection is when text the model processes contains instructions that override the developer's intent. Direct injection comes from the user's message. Indirect injection hides in content the application feeds the model: a web page, an email, a retrieved document, a tool result.

Indirect injection is the bigger risk in agents and RAG systems, because the attacker never talks to your app. They plant text where your app will read it, and the model can't reliably tell data from instructions.

The OWASP Top 10 for LLM Applications lists prompt injection first. The practical takeaway is to assume it will sometimes succeed and limit what a successful injection can do.

Likely follow-ups:

- Give an example of indirect injection against an email assistant.
- Why can't you fix this with a better system prompt alone?

11. Multimodal: documents, images and voice

11.1 How does a vision-language model 'see' an image, and what does that imply for cost and accuracy?

Junior · AI product startup, Big tech, Enterprise

What they're testing: A working mental model of image inputs as tokens.

The image is split into patches, each patch is encoded into vectors, and those vectors are projected into the language model's input space, where they're processed like tokens alongside the text.

So images cost tokens, and more pixels usually means more tokens, more latency and more money. Providers typically resize large images or tile them, and their docs explain how image size maps to token count.

Accuracy depends on resolution: small text, dense tables and fine details can be lost when an image is downsampled. Cropping to the relevant region often beats sending a huge image.

Models can also hallucinate plausible details that aren't in the image, so for anything that matters, such as numbers on a receipt, validate against rules or ask the model to quote exactly what it read.

Likely follow-ups:

- How would you handle a 40-page scanned PDF?
- When would you crop versus tile an image?

12. Coding rounds, take-homes and AI-assisted engineering

12.1 In a live coding round, implement a function that calls an LLM API and returns parsed JSON. What would you make sure to handle?

Junior · AI product startup, Big tech

What they're testing: Production instincts around a simple model call: errors, retries, validation.

Say the plan before typing: build the request, call the API with a timeout, parse and validate the response, and handle failures explicitly.

Use the provider's structured output or JSON mode if available, then still validate against a schema, because valid JSON isn't the same as the right fields and types.

Handle the failures that actually happen: timeouts, rate-limit and server errors (retry with exponential backoff and jitter, with a cap), and malformed output (one retry with the validation error in the prompt, then a clear error).

Keep the API key in configuration, not code, and write one quick test with a stubbed client. Interviewers notice when you test the unhappy path without being asked.

Likely follow-ups:

- Which errors should not be retried?
- How would you make this function testable?

Get the full kit

120 questions with worked answers across 12 chapters, tagged by level and company type, plus the 14-day prep plan.

stackchief.com/products/ai-engineer-interview-kit