The AI Engineer Interview Kit

A free, portable prep kit for AI/GenAI engineering interviews — an answer framework, a 30-question bank with strong answers, a system-design template, and a mock-interview scoring rubric.

A free guide from ProductionAIEngineer.com · v1.0 · Source file — deliver as the lead-magnet PDF.

> **What this is:** a sharp, self-contained sample you can print and rehearse from. **What it isn't:** the full course. The **AI Engineer Interview & Portfolio Kit** (paid) ships 100+ questions with model answers, reference architectures, and video walkthroughs. This free kit gets you genuinely interview-ready for the fundamentals.

How to use this kit

1. Read the **answer framework** — it works for almost any AI engineering question.
2. Rehearse the **question bank** out loud until the reasoning (not the wording) is automatic.
3. Fill in the **project explanation template** for each of your projects.
4. Drill the **system-design template** on RAG and agents.
5. Record a mock and score it with the **rubric**. Repeat until you're ≥ 3.5/5 everywhere.

The role map (know which interview you're in)

Role	What they test hardest
AI Engineer	RAG, agents, system design, production judgment
GenAI Developer	Prompts, structured outputs, function calling, evals
Agentic AI Engineer	Tool use, memory, orchestration, guardrails, *when not* to use an agent
Forward Deployed Engineer	AI engineering + discovery, scoping, demos, communication

---

The universal answer framework

For any technical AI question, answer in this order. It signals senior thinking.

1. **Clarify** the requirement and constraints (scale, latency, cost, accuracy).
2. **Name the trade-off** at the heart of the question.
3. **Give your default** and *why* (the 80% answer).
4. **Show the edge** you'd handle (where the default breaks).

5. **Prove it** with a metric, a trace, or how you'd evaluate it.

> Weak candidates recite definitions. Strong candidates reason about trade-offs and say how they'd *measure* success.

Question bank (30 questions with strong answers)

A. Fundamentals

1. What is an embedding, and why does it matter? A vector representation of meaning; nearby vectors = similar meaning. It matters because it turns "find relevant text" into "nearest-neighbor search." Edge: embeddings capture semantic similarity, not correctness — which is why retrieval can be "close" but wrong.

2. Temperature: what is it and when do you change it? Controls output randomness. Default low (0–0.3) for extraction, structured output, and RAG; higher for brainstorming/creative. In production I keep it low and deterministic where I need reliability and evals.

3. What's a context window, and why is it a constraint? The max tokens a model can attend to. It caps how much retrieved context/history fits, drives cost, and creates "lost in the middle" effects — so I rank and compress context rather than stuffing it.

4. Structured outputs — why and how? Real apps need machine-readable results. I use JSON schema / function calling and then **validate** — never trust raw model JSON. Invalid output → repair or retry, don't crash.

5. When would you NOT use an LLM? When a deterministic rule, a classifier, or a SQL query is cheaper, faster, and more reliable. Using an LLM where a regex works is a red flag.

B. RAG

6. Walk me through a production RAG pipeline. Ingest → clean → structure-aware chunk (+metadata) → embed → index → retrieve (hybrid) → rerank → assemble context (budgeted) → grounded generation with citations → eval + trace. I'd emphasize retrieval quality and evals — that's where systems fail.

7. Your RAG gives confident but wrong answers. Debug it. Print the retrieved chunks first. If the right context wasn't retrieved → retrieval problem (chunking, embeddings, add hybrid + reranking, tune top-k). If it was retrieved → generation problem (grounding prompt, context order, refusal path).

8. Fixed-size vs structure-aware chunking? Fixed size is easy but splits mid-thought and destroys meaning. I chunk on structure (headings/sections), keep tables/code intact, and use parent-child (retrieve small, feed larger context).

9. What is reranking and when do you add it? A second-stage cross-encoder that reorders candidates by true relevance. Over-retrieve (20–50) then rerank to a handful. Add it whenever precision matters or vector-only returns noisy top-k.

10. Hybrid search — why? Dense vectors miss exact terms (names, IDs, acronyms). BM25/keyword catches them. Hybrid + rerank is my safe default.

11. How do you stop hallucination in RAG? Grounded prompt ("answer only from context; else say you don't know"), a low-confidence retrieval threshold that triggers refusal, required citations, and an eval that includes unanswerable questions.

12. How do you evaluate a RAG system? Retrieval: recall@k, context precision. Generation: groundedness, answer relevance, citation accuracy. On a versioned golden set, in CI, plus online feedback.

C. Agents

13. What actually makes something an "agent"? An LLM in a loop that chooses actions (tools) toward a goal, with memory and a stop condition — versus a fixed chain. The key skill is knowing when a chain or a single function call is better.

14. When should you NOT build an agent? When the task is deterministic or a single tool call solves it. Agents add latency, cost, and failure surface. I reach for the simplest thing that works.

15. How do you keep an agent from looping forever? A hard step/iteration cap, a cost ceiling, a clear stop condition, and progress checks. Unbounded loops are an outage and a bill.

16. Agent memory — what kinds? Short-term (conversation/scratchpad) and long-term (persistent store, often vector-backed). I keep only what's needed, summarize to control tokens, and never persist secrets.

17. Tool/function calling — how do you make it reliable? Typed schemas, argument validation, idempotent tools, error handling that lets the agent recover, and evals on tool-call accuracy.

18. Human-in-the-loop — where? Before irreversible or high-cost actions (payments, deletes, external sends). Checkpoints turn a risky agent into a shippable one.

D. Evals, observability, production

19. How do you measure LLM quality without ground truth? LLM-as-judge calibrated against human labels, rubric-based scoring, and online signals (thumbs, edits). Report judge-human agreement so the metric is trustworthy.

20. What do you log for every LLM call? request/session id, input (redacted), retrieved chunks+scores, model+prompt version, tokens, per-stage latency, cost, output, citations, feedback, status. Redact PII before logging.

21. How do you control LLM cost? Right-size models, cache (semantic + embedding), enforce per-request/day budgets, cap tokens/context, and trace where tokens go.

22. How do you prevent regressions when you change a prompt? Prompts are versioned; a regression suite on the golden set runs in CI and blocks merges that drop key metrics.

23. Prompt injection — what is it and how do you defend? Malicious instructions in user input or **retrieved content** that hijack the model. Defenses: treat retrieved data as untrusted, separate instructions from data, constrain outputs, and never give the model unchecked power over irreversible actions.

24. How do you deploy an LLM app safely? Docker, secrets in a manager (never in code/client), health checks, CI with an eval smoke test, timeouts/retries, and monitoring + alerts before you scale traffic.

E. Coding / ML / LLM applied

25. Given a slow RAG endpoint at p95, how do you speed it up? Trace per stage. Usual wins: reduce top-k, compress/dedupe context, cache, parallelize retrieval+rerank, use a smaller generation model where quality holds.

26. Design a rate-limited, retried client for an LLM API. Exponential backoff + jitter on 429/5xx, a token-bucket limiter, per-request timeout, circuit breaker on sustained failure, and idempotency for retried calls.

27. How would you A/B test a new prompt in production? Split traffic, hold the eval harness constant, compare groundedness/relevance + business metric + cost/latency, and require significance before rollout.

28. How do you choose an embedding model? Benchmark candidates on *your* corpus (recall@k), weigh dimensions/cost/latency, and confirm the same model is used for index and query.

F. Behavioral

29. Tell me about a production AI system you built. *(Use the template below.)* Problem → approach → the hard trade-off → result **with a metric**. Keep it to 2 minutes; invite follow-ups.

30. Tell me about a time your AI system failed in production. Pick a real failure, show the trace/diagnosis, the fix, and the guardrail/eval you added so it can't recur. Owning failure + adding a safeguard is exactly the signal they want.

Project explanation template

Fill one per project; rehearse to ~2 minutes.

```
Project: <name>
In one sentence: <what it does, for whom>
Problem: <why it mattered>
Approach: <architecture in 3 bullets — data flow>
Hardest problem: <the real challenge>  →  How I solved it: <…>
Key trade-off: <chose X over Y because …>
Result: <metric — eval score, latency, cost, adoption>
If I had more time: <the honest next step>
```

System-design template (use in the round)

1. **Clarify** — users, scale (QPS, corpus size), latency & cost targets, accuracy bar.
2. **High-level** — draw ingestion, retrieval, generation, evaluation, and data flow.
3. **Deep-dive the hard part** — retrieval quality (RAG) or the agent loop (agents).
4. **Evals** — how you'd measure success (metrics + golden set).
5. **Production** — observability, cost control, security (injection), failure modes.
6. **Trade-offs** — call out what you'd change at 10× scale.

Two designs to have ready: a production RAG system, and a tool-using agent.

Mock-interview scoring rubric

Score each 1–5. Target **≥ 3.5 across all** before interviewing for real.

Dimension	1	3	5
Problem clarification	Jumps in	Asks some	Nails constraints first
Technical depth	Definitions only	Solid mechanics	Trade-offs + edges
System design	Logos, no flow	Reasonable design	Evals + cost + security included
Production judgment	Ignores ops	Mentions it	Observability/cost/safety by default
Communication	Rambles	Clear	Crisp, structured, invites follow-ups
Evidence/metrics	None	Some	Every claim has a number

Total /30. < 18 keep drilling · 18–24 keep polishing · 25+ interview with confidence.

Bonus: the 48-hour pre-interview plan

- **T-48h:** re-read your project templates; re-run one eval so numbers are fresh.
- **T-24h:** two mock rounds (one RAG design, one agent design); score with the rubric.
- **T-3h:** skim this question bank; rehearse your two best project stories out loud.
- **T-0:** clarify before answering, name trade-offs, cite a metric. Breathe.

Instant red flags to avoid: secrets in code, "the model was just wrong" (no diagnosis), agents where a function would do, no evals, no citations, no cost awareness.

Your next step

1. **Keep sharpening** → subscribe to **Production AI Notes** for a weekly question + concept.
2. **Have projects to talk about** → follow the free **90-Day AI Engineer Roadmap**.
3. **Go all-in** → the **AI Engineer Interview & Portfolio Kit** adds 100+ questions with model answers, reference architectures, resume bullets, and video walkthroughs.

Built by Gururaj · ProductionAIEngineer.com · Ship AI that survives real users. Public/synthetic examples only. Opinions my own.
