

ALIF PRESS

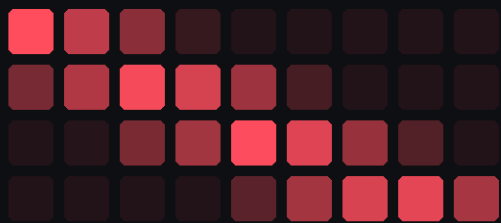
VERSION 1.2 · OCTOBER 2026

The AI Engineer Interview Playbook

**Fast, Practical Prep for AI Engineer and
Forward Deployed Engineer Roles**

Everything they'll ask. Nothing they won't.

- 26 chapters · 14 diagrams · every interview round
- Transformers, RAG, evals, agents, MCP and LangGraph
- Take-homes, AI-assisted coding and the FDE customer round



THE CONCEPTS · THE CODE · THE CUSTOMER ROUNDS

Anees Akhter

Contents

Preface.....	2
How to Use This Book.....	3
1 ML Essentials.....	6
2 Deep Learning Essentials.....	8
3 Embeddings and Tokenization.....	10
4 Transformers and Self-Attention.....	12
5 How LLMs Are Trained.....	15
6 Decoding and Generation.....	17
7 Prompt Engineering Patterns.....	20
8 RAG in Depth.....	22
9 Evaluating LLM Systems.....	24
10 AI Agents and Tool Use.....	27
11 MCP (Model Context Protocol).....	30
12 Building an MCP Server and Agent Skills.....	35
13 LangChain Today.....	39
14 LangGraph: Agents as Graphs.....	41
15 Other Agent Frameworks.....	45
16 Production: Latency, Cost, Guardrails, Monitoring.....	49
17 Serving and Inference.....	52
18 AI System Design: A RAG Chatbot.....	56
19 Take-Home Assignments.....	58
20 The AI-Assisted Coding Round.....	61
21 The FDE Rounds: Decomposition and Role-Play.....	63
22 The Project Deep Dive.....	67
23 Forty Questions to Practise Out Loud.....	69
24 Behavioural Answers for AI and FDE Roles.....	73
25 Interviewing Across Regions.....	75
26 The Night-Before Cheat Sheet.....	77
A A Fundamentals Refresher.....	79
Thank You.....	80
Your Next Steps.....	81
About the Author.....	82

PART 1 · CHAPTER 4

4 Transformers and Self-Attention

Self-attention lets every token look at every other token and decide how much each one matters, all at the same time.

The Intuition

Each token produces three vectors. The **query** says what it's looking for, the **key** says what it offers, and the **value** is the information it hands over. A token compares its query with every key, turns those scores into weights, and takes a weighted average of the values.

In “The animal didn’t cross the street because **it** was tired”, attention is how “it” ends up putting most of its weight on “animal”.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

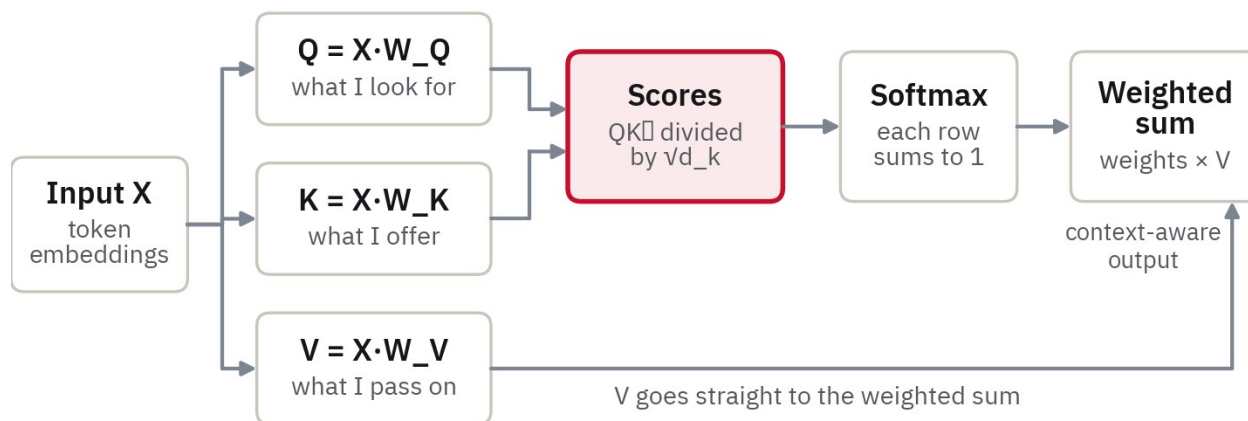


Figure 4.1 · Self-attention in four steps, from projections to a context-aware output.

Q , K and V are each $n \times d_k$, where n is the number of tokens. They come from the input embeddings X multiplied by three learned matrices: $Q = X \cdot W_Q$, $K = X \cdot W_K$, $V = X \cdot W_V$.

1. **Scores.** QK^T gives an $n \times n$ grid of query–key similarities.
2. **Scale.** Divide by $\sqrt{d_k}$ so large dot products don't push softmax into tiny gradients.
3. **Softmax** each row, so every token's weights add up to 1.
4. **Weighted sum.** Multiply by V to get each token's new, context-aware vector.

Worked Example

Three tokens with $d_k = 4$, so $\sqrt{d_k} = 2$, which keeps the scaling step easy to do by hand.

Step	Token 1	Token 2	Token 3
Q row	[1, 0, 1, 0]	[0, 2, 0, 0]	[1, 1, 1, 1]
K row	[1, 0, 1, 0]	[0, 1, 0, 1]	[1, 1, 0, 0]
V row	[4, 0]	[0, 4]	[2, 2]
Scores QK^T	[2, 0, 1]	[0, 2, 2]	[2, 2, 2]
Scaled ($\div 2$)	[1, 0, 0.5]	[0, 1, 1]	[1, 1, 1]
Softmax weights	[0.51, 0.19, 0.31]	[0.16, 0.42, 0.42]	[0.33, 0.33, 0.33]
Output (weights $\times V$)	[2.64, 1.36]	[1.47, 2.53]	[2.00, 2.00]

To check token 1 by hand, use the unrounded weights: $0.506 \times [4, 0] + 0.186 \times [0, 4] + 0.307 \times [2, 2] = [2.64, 1.36]$. Notice token 3: its scores are all equal, so it simply averages the three values.

Multi-Head Attention in Three Lines

- Run h attention heads in parallel, each with its own W_Q , W_K , W_V and $d_k = d_{\text{model}} / h$.
- Concatenate the heads' outputs and project them back to d_{model} with W_O .
- In BERT-base, $d_{\text{model}} = 768$ and $h = 12$, so each head is 64-dimensional.

How Interviewers Ask It

Why divide by $\sqrt{d_k}$? Dot products grow with the dimension. Large scores make softmax nearly one-hot, and then the gradients vanish. Dividing by $\sqrt{d_k}$ keeps the variance of the scores close to 1, which keeps training stable.

Why several heads instead of one big one? Each head can learn a different relationship, such as syntax, who “it” refers to, or nearby words, for the same total cost. A single head would blur those patterns together.

What does attention cost, and how does a KV cache help? Attention is $O(n^2)$ in sequence length because of the $n \times n$ score grid. During generation, past keys and values never change, so a KV cache stores them and each new token costs $O(n)$ instead of a full recompute. The price is memory that grows with the context.

A follow-up to expect: “How would you serve a model with a 128k context window?” Talk about KV-cache memory, grouped-query attention and paged attention.

PART 2 · CHAPTER 14

14 LangGraph: Agents as Graphs

When an agent needs loops, branches, memory that survives a crash and a human sign-off, you stop hoping the model follows a prompt and draw the control flow as a graph.

Why a Graph

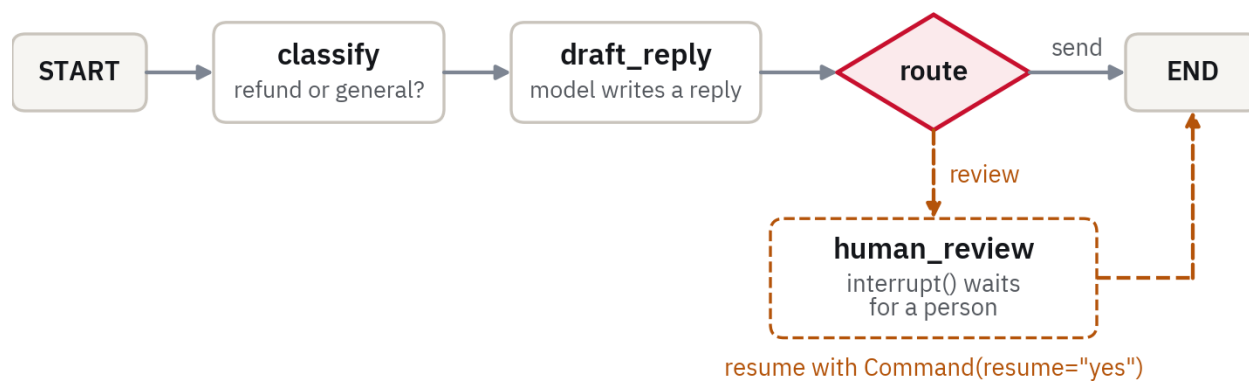
A plain agent loop (Chapter 10) lets the model decide every step. That's flexible, but hard to test, audit or pause. LangGraph makes the flow explicit: **nodes** do work, **edges** decide what happens next, and a shared **state** object carries everything between them. LangChain's own `create_agent` runs on LangGraph underneath, so the two fit together.

The Core Ideas

Concept	What it is	Why it matters
State	A typed dictionary every node reads and updates	One place to see everything the agent knows
Node	A function that takes the state and returns an update	Each step is small and testable on its own
Edge	A fixed link from one node to the next	Predictable order where order matters
Conditional edge	A function that picks the next node from the state	Branches and loops, decided by your code
Reducer	A rule for merging updates into state, such as appending to a list	Parallel nodes can write without overwriting each other
Checkpoint	Saves the state after every step, per conversation thread	Resume after a crash, keep memory, replay past runs
Interrupt	Pauses the graph and waits for a human	Approvals and edits before anything risky happens

Worked Example: A Support Ticket With a Human Check

Refund requests need a person to approve the reply; everything else goes straight out.



Checkpoint: the state is saved after every step under the thread ID, so a paused or crashed run can resume

Figure 14.1 · The graph for the code below. The dashed path pauses for human approval.

```

from typing_extensions import TypedDict
from langgraph.graph import StateGraph, START, END
from langgraph.checkpoint.memory import InMemorySaver
from langgraph.types import interrupt, Command

class Ticket(TypedDict, total=False):
    text: str
    category: str
    draft: str
    approved: bool

def classify(state: Ticket) -> dict:
    category = "refund" if "refund" in state["text"].lower() else "general"
    return {"category": category}

def draft_reply(state: Ticket) -> dict:
    # In a real system this node calls a model.
    return {"draft": f"Thanks for getting in touch about your {state['category']} request."}

def route(state: Ticket) -> str:
    return "review" if state["category"] == "refund" else "send"

def human_review(state: Ticket) -> dict:
    decision = interrupt({"draft": state["draft"], "question": "Send this reply?"})
    return {"approved": decision == "yes"}

builder = StateGraph(Ticket)
builder.add_node("classify", classify)
builder.add_node("draft_reply", draft_reply)
builder.add_node("human_review", human_review)
builder.add_edge(START, "classify")
builder.add_edge("classify", "draft_reply")
builder.add_conditional_edges("draft_reply", route, {"review": "human_review", "send": END})
builder.add_edge("human_review", END)

graph = builder.compile(checkpointer=InMemorySaver())
config = {"configurable": {"thread_id": "ticket-42"}}

graph.invoke({"text": "I want a refund for order 1182"}, config) # pauses at human_review
graph.invoke(Command(resume="yes"), config) # resumes and finishes

```

We ran this against LangGraph 1.2: the first call stops at the review step, and the second finishes with `approved: True`. In production, swap `InMemorySaver` for a database-backed checkpointer such as Postgres so paused threads survive restarts.

Patterns You'll Be Asked About

Pattern	Shape	Use it for
Router	One node classifies, conditional edges send work to specialists	Triage and intent routing
Loop with a limit	A node retries or refines until a check passes or a counter runs out	Self-correction, research loops
Human in the loop	<code>interrupt</code> before the risky node, <code>Command(resume=...)</code> to continue	Payments, emails, data changes
Supervisor	A coordinating agent hands tasks to worker agents and combines results	Multi-agent work with clear roles
Subgraph	A whole graph used as a node inside another graph	Reusing a tested workflow

How Interviewers Ask It

When would you use LangGraph instead of a simple agent loop? When you need guaranteed steps, branches your code controls, pausing for people, or resuming long-running work. For a single model call plus a tool or two, a plain loop is simpler.

How does human-in-the-loop work? The node calls `interrupt`, the checkpointer saves the state under the thread ID, and the graph stops. Later, `Command(resume=...)` reloads that state and carries on from the same node.

What is a reducer, and when do you need one? It defines how updates merge into state. You need one when several nodes write to the same field, for example appending messages instead of replacing them.

Your agent crashed halfway through a 20-step job. What happens? With a durable checkpointer, you rerun with the same thread ID and it continues from the last saved step instead of starting over.

PART 3 · CHAPTER 21

21 The FDE Rounds: Decomposition and Role-Play

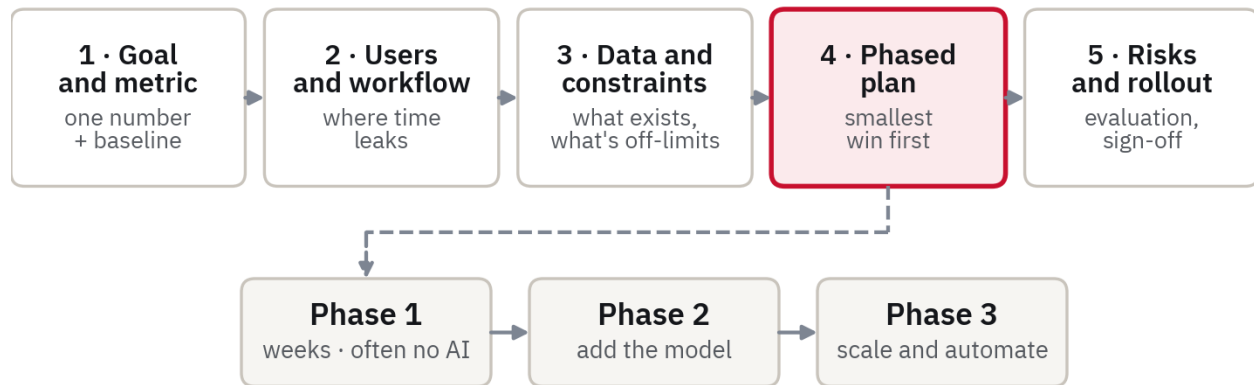
You get a vague business problem, the interviewer plays the customer, and you're scored on how you think out loud, not on finding the perfect solution.

What They're Scoring

Interviewers watch for four things. Do you ask before you build? Do you tie everything to a number the customer cares about? Do you ship something small first? Do you name the risks without being prompted? Jumping straight to “let’s fine-tune an LLM” is the most common way to fail.

A Five-Step Framework (say Each Step Out loud)

1. **Goal and metric.** “What does success look like in three months, and how do you measure it today?” Leave with one number.
2. **Users and workflow.** Who does this work now, step by step? Where exactly do time or money leak?
3. **Data and constraints.** What data exists, where it lives, who owns it, and what’s off-limits: privacy, compliance, on-premises rules, budget.
4. **A phased solution.** Phase 1 is the smallest thing that moves the metric within weeks, and it’s often not AI. Phase 2 adds the model. Phase 3 scales and automates.
5. **Risks, evaluation and rollout.** How you’ll know it works, what could break, and who has to say yes.



Say each step out loud. Interviewers score the thinking, not a perfect answer.

Figure 21.1 · The five-step decomposition, and how its answer becomes a phased plan.

Scenario A: "Our ER Wait Times Are Too Long" (a Hospital COO)

Clarify: What's the median door-to-doctor time today, and what's the target? Is the bottleneck triage, beds or discharge? Which systems hold the data, such as the EHR and bed management?

A model answer: Suppose the median wait is 95 minutes and the goal is 60. Phase 1 is a dashboard joining EHR timestamps with bed status to find the slowest step; often it's discharge delays upstairs, not the ER itself. Phase 2 is a model that forecasts arrivals four to eight hours ahead so staffing and beds can be planned. Phase 3 is an LLM that drafts discharge summaries, with a clinician signing off, so beds free up faster. **Risks:** patient data (HIPAA), clinicians' trust, and never letting the model make a triage decision on its own.

Scenario B: "Too Many Deliveries Are Missed" (a Logistics VP)

Clarify: What counts as missed: late, a failed attempt, or lost? What's the current rate and the cost per failure? Do drivers use an app?

A model answer: Suppose 8% of deliveries fail and most are "customer not home". Phase 1 sends an ETA text with a reschedule link, with no machine learning at all. Phase 2 is a model that predicts each delivery's risk of failing from address history and the time window, so risky stops get a confirmation call. Phase 3 optimises routes using those predictions. **Evaluate** with an A/B test by region. **Risks:** messaging costs and customers opting out.

Scenario C: "We Want an AI Assistant for Our Support Team" (a bank)

Clarify: Which number hurts: handle time, backlog or quality? What's in the knowledge base, and how up to date is it? Can customer data leave the bank's cloud?

A model answer: Start with something the support agents use, not the customers, because it's lower risk. Phase 1 is RAG over the policy documents, suggesting answers with citations that agents accept or edit. Phase 2 summarises calls automatically and pre-fills tickets. Phase 3 is a customer-facing bot, but only for low-risk requests. **Evaluate** with a 200-question golden set plus the rate at which agents accept suggestions. **Risks:** stale documents, made-up policy, personal data in prompts, and regulator review.

The Customer Role-Play

In many FDE loops the interviewer plays the customer, either inside the case or in a round of its own. They might be friendly, frustrated or completely non-technical, and the conversation tends to escalate through three moments, each testing something different.

Moment	The “customer” says	A weak reply	A strong reply	What’s being tested
Discovery	“We just need AI for our claims process. Can you build it?”	“Sure, we’ll use RAG with a fine-tuned model.”	“Happy to. Before we pick anything, can you walk me through a claim from the moment it arrives? Where does it get stuck, and how do you measure that today?”	Asking before building
Scope dispute	“Can it handle fraud detection too? That should be quick to add.”	“Yes, no problem.”	“We can, but it’s a different problem, with different data and a higher cost for mistakes. Adding it now would push the claims work back by several weeks. Shall we ship claims first and scope fraud as phase two?”	Protecting scope by offering options, not refusals
Live failure	“Your demo just told my VP about a policy we don’t have. This is embarrassing.”	“That’s a known limitation of LLMs.”	“You’re right, and I’m sorry it happened in front of your VP. The answer wasn’t in the documents, so the assistant guessed. Today I’ll make it say it doesn’t know when it can’t find a source, and tomorrow I’ll send you the test results, including that exact question.”	Ownership, calm and a concrete fix

Rules That Carry You Through

- **Repeat back what you heard** before you answer: “So the real problem is that claims sit for days waiting for a human.”
- **Put trade-offs in business terms:** time, risk and money, not model names.
- **Never promise what you can’t deliver** just to calm the room. Offer options instead.
- **When something goes wrong**, own it, explain it in plain words, and give a next step with a date.
- **Drop the jargon with non-technical customers.** “It searches your documents first, then answers from what it finds” lands better than “RAG”.

How to practise: ask a friend to play the customer using the three lines above, and to switch moods each time: friendly, impatient, sceptical.

Phrases That Earn Points

- “Before I propose anything, can I ask a few questions about how this works today?”
- “If we only fix one thing in the first month, I’d fix...”
- “I’d want a human in the loop here until we’ve measured accuracy.”
- “Here’s what I’d need from your team to start on Monday.”

You've read the sample.

Now get the whole book.

26 chapters covering every round of AI Engineer and Forward Deployed Engineer interviews, from transformers to the customer role-play, plus a night-before cheat sheet.

The Book · \$27 · PDF and EPUB

Book + Interview Pack · \$47

alifanalytics.com/ai-engineer-interview-playbook