PRANEETH KALLURI

presents

LLM Engineering Roadmap 2026

8-Week Practical Guide to Building with Large Language Models

PRANEETH KALLURI

Instagram: @praneeth_kalluri | @praneethkalluri | LinkedIn

Next.js • Vercel AI SDK v6 • GPT-5 Family • Multi-Provider

For working professionals & students | 1 hour/day | All free resources

How to Use This Roadmap

- **1 hour/day is enough.** Each week tells you exactly how to split your time across watching, reading, and building.
- **Build every project.** Tutorials teach theory. Projects teach skills. Every week ends with a real project for this reason.
- **One API key is enough.** Get OpenAI, set $5-10/month budget, use **gpt-5-mini** for everything unless specified.
- **Watch first, then build.** Videos build understanding. Docs are reference while coding.
- **Push everything to GitHub.** 8 weeks = 8 projects = a portfolio that proves your skills.
- **Share your progress.** Post on LinkedIn, Twitter, Instagram. Teaching accelerates learning.

API Setup — Do This Before Week 1

- Get an OpenAI API key → https://platform.openai.com/api-keys
- Set budget limit ($5-10/month) → https://platform.openai.com/settings/organization/limits
- **Default model: gpt-5-mini** — cheapest, fastest, great for learning
- Understand model tiers → OpenAI Models Page

GPT-5 Model Tiers (March 2026)

Model	Best For	Speed	Cost
gpt-5-mini	Most tasks, learning, prototyping	Fastest	Cheapest
gpt-5	Complex reasoning, vision, agents	Medium	Medium
gpt-5.4	Flagship — hardest tasks, agentic	Slower	Premium

*GPT-5 has built-in reasoning via **reasoning.effort** (none/low/medium/high). Adjust effort instead of switching models.*

Roadmap Overview

Week 0	Prerequisites
Week 1	Understand LLMs
Week 2	Prompt Engineering & Structured Outputs
Week 3	AI Agents — Concepts & First Agent
Week 4	Tool Use, Function Calling & Workflows
Week 5	RAG — Retrieval Augmented Generation
Week 6	Multi-Modal — Vision & Beyond Text
Week 7	Multi-Provider, Evaluation & Security
Week 8	Ship It — Full Product (Capstone)
Bonus	Build Your Own LLM + Courses

WEEK 0

Prerequisites

Make sure you're comfortable with these before starting. Working developers can skip to Week 1.

- JavaScript / TypeScript basics
- React fundamentals
- Git & GitHub
- Terminal / CLI basics
- REST APIs & HTTP
- Next.js basics (Vercel AI SDK v6 runs on Next.js)

WEEK 1

Beginner Friendly

Understand LLMs

Pace: Days 1-3: Watch videos | Days 4-5: Read docs + set up | Days 6-7: Build project

By the end of this week, you will:

✓ Explain how LLMs work (tokens, context windows, training) to a non-technical person

✓ Set up a Next.js project with Vercel AI SDK v6 and connect to OpenAI

✓ Build a working streaming chatbot from scratch

✓ Understand the difference between GPT-5 model tiers

Watch

• Andrej Karpathy — Intro to Large Language Models (1hr) → https://www.youtube.com/watch?v=zjkBMFhNj_g

• Andrej Karpathy — Software in the Era of AI → https://www.youtube.com/watch?v=LCEmiRjPEtQ

• 3Blue1Brown — Neural Networks + LLM Series → https://www.3blue1brown.com/topics/neural-networks

Read / Study

• Vercel AI SDK v6 — Introduction → https://ai-sdk.dev/docs/introduction

• Vercel AI SDK v6 — Next.js App Router → https://ai-sdk.dev/docs/getting-started/nextjs-app-router

• OpenAI Models (GPT-5 family) → https://developers.openai.com/api/docs/models

PROJECT: "Explain Like I'm 5" Bot

What you're building:

A web app where users type any complex topic — quantum physics, blockchain, machine learning, how the internet works — and get a simple, jargon-free explanation.

Users can pick which GPT-5 model tier generates the response and adjust the complexity level, so they can instantly see how different models and prompts affect output.

Why this matters: This is the 'Hello World' of LLM engineering. Every AI product starts with calling an API and streaming a response. Master this, and everything else builds on top.

UI Layout:

Top bar: App title + model selector dropdown (gpt-5-mini / gpt-5 / gpt-5.4)

Center: Large text input ('Enter any topic...') + complexity slider (ELI5 / High School / College / Expert)

Below: Streaming response area — words appear in real-time as the model generates

Bottom bar: Metrics — response time (ms), tokens used, estimated cost

Core Functionality:

- Streaming responses using **streamText** — words appear one by one, not after a long wait
- Model switching — same prompt, different model, see the quality/speed difference instantly
- Complexity slider changes the system prompt (e.g., ELI5: 'Explain like I am a 5 year old child')
- Live metrics display — response time, token count, cost estimate update after each response
- Conversation history — previous explanations stay visible on the page

Build it step by step:

Step 1: Create Next.js project: **npx create-next-app@latest eli5-bot**

Step 2: Install AI SDK: **npm install ai @ai-sdk/openai** and add your OpenAI API key to .env.local

Step 3: Create API route at **app/api/chat/route.ts** — use **streamText** with a system prompt

Step 4: Build the UI: text input, model dropdown, complexity slider using React state

Step 5: Wire UI to API using the **useChat** hook from ai/react

Step 6: Add metrics: measure response time with Date.now(), display token count from the response

What you'll learn: API setup, first AI SDK v6 app, streamText, useChat hook, provider config, model tiers.

Common Mistakes to Avoid

✗ Don't skip the Karpathy videos. Understanding HOW LLMs work makes you 10x faster at debugging later.

✗ Don't use gpt-5.4 for everything — start with gpt-5-mini. It's cheaper and teaches you to write better prompts.

✗ Don't copy-paste boilerplate. Type every line yourself. Muscle memory matters.

WEEK 2

Beginner Friendly

Prompt Engineering & Structured Outputs

Pace: Days 1-2: Watch Karpathy | Days 3-4: Prompt tutorials | Days 5-7: Build project

By the end of this week, you will:

✓ Write system prompts that produce consistent, reliable outputs
✓ Get structured JSON from any LLM using Zod schemas
✓ Understand chain-of-thought, few-shot, and prompt variation
✓ Know how small wording changes dramatically shift quality

Watch

• Andrej Karpathy — Deep Dive into LLMs (3.5hrs) → https://youtu.be/7xTGNNLPyMI
• Andrej Karpathy — How I use LLMs → https://youtu.be/EWvNQjAaOHw

Read / Study

• Anthropic Prompt Engineering Guide → https://platform.claude.com/docs/en/build-with-claude/prompt-engineering/overview
• Anthropic Interactive Tutorial (exercises) → https://github.com/anthropics/prompt-eng-interactive-tutorial
• Vercel AI SDK v6 — Structured Data → https://ai-sdk.dev/docs/ai-sdk-core/generating-structured-data
• Vercel AI SDK v6 — Prompt Engineering → https://ai-sdk.dev/docs/ai-sdk-core/prompt-engineering
• OpenAI Prompt Caching → https://developers.openai.com/api/docs/guides/prompt-caching

PROJECT: AI Code Reviewer

What you're building:

A web app where developers paste any code snippet and receive a detailed, structured code review — as if a senior engineer reviewed their pull request.

The app returns a JSON-structured review with specific issues (with line numbers), improvement suggestions, an overall quality score, and refactored code. Users can toggle between a 'Friendly Mentor' and 'Strict Senior Dev' review style.

Why this matters: Every AI product that returns data (not just text) needs structured outputs. This is the #1 skill companies hire for — making LLMs return reliable, parseable data.

UI Layout:

Left panel: Code input area with syntax highlighting + language dropdown (Python, JS, TS, etc.)

Top right: 'Review Style' toggle — Friendly Mentor vs Strict Senior Dev

Right panel: Review results — Score badge (1-10 with color), Issues list (each with line number, severity tag, description), Suggestions list, Refactored code with diff highlighting

Per issue: A 'Fix This' button that generates corrected code for that specific issue

Core Functionality:

- Structured JSON enforced by Zod — issues array with line/severity/description, suggestions array, score number, refactored_code string
- Two distinct review personas via different system prompts — same code, dramatically different tone
- **generateObject** with Zod schema — the model MUST return data matching your schema
- Per-issue 'Fix This' — clicking generates a targeted fix for just that issue
- Syntax highlighting for both input code and refactored output

Build it step by step:

Step 1: Create Next.js project with AI SDK v6

Step 2: Define Zod schema: z.object({ issues: z.array(z.object({ line: z.number(), severity: z.enum(['critical','warning','info']), description: z.string() })), suggestions: z.array(z.string()), score: z.number(), refactored_code: z.string() })

Step 3: Create API route using **generateObject** — pass the Zod schema so output is guaranteed structured

Step 4: Build code input UI with a textarea and language selector dropdown

Step 5: Add review style toggle that switches between two system prompts

Step 6: Render the structured review: color-coded severity badges, numbered issues, expandable suggestions

Step 7: Add 'Fix This' button per issue — sends the issue + original code to generateObject for a targeted fix

What you'll learn: System prompts, generateObject, Zod schemas, prompt variation, structured outputs.

Common Mistakes to Avoid

✗ Don't write vague prompts like 'review this code'. Specify WHAT to look for, output format, and scoring criteria.

✗ Don't skip the Anthropic interactive tutorial — it's hands-on practice, not passive reading.

✗ Don't trust the model to follow format without Zod. Always enforce structure programmatically.

WEEK 3

Intermediate

AI Agents — Concepts & First Agent

Pace: Days 1-2: Read agent guides | Days 3-4: Study SDK agent docs | Days 5-7: Build project

By the end of this week, you will:

✓ Explain what an agent loop is and how it differs from a single LLM call

✓ Know when to use an agent vs a workflow vs a single prompt

✓ Build a multi-step agent using Vercel AI SDK v6 Agent abstraction

✓ Display agent thinking process in real-time

Read / Study

• Anthropic: Building Effective Agents (best article on agents) → https://www.anthropic.com/research/building-effective-agents

• OpenAI: Practical Guide to Building Agents (PDF) → https://cdn.openai.com/business-guides-and-resources/a-practical-guide-to-building-agents.pdf

• Vercel AI SDK v6 — Agents Overview → https://ai-sdk.dev/docs/agents/overview

• Vercel AI SDK v6 — Building Agents → https://ai-sdk.dev/docs/agents/building-agents

• Vercel AI SDK v6 — Loop Control → https://ai-sdk.dev/docs/agents/loop-control

PROJECT: Smart Study Planner Agent

What you're building:

An AI agent that creates personalized, day-by-day study plans. Users enter what they want to learn, how much time they have, and their current level. The agent autonomously breaks the topic into subtopics, researches the learning order, generates a daily schedule with resources, and creates quiz questions — all in multiple reasoning steps.

Unlike a single prompt (which would just dump everything at once), this agent thinks step by step: first it analyzes the topic, then structures subtopics, then assigns them to days, then finds resources, then creates quizzes. Users watch the agent think in real-time.

Why this matters: Agents are the hottest skill in AI engineering. But most people misuse them. This project teaches you WHEN agents help and HOW the loop works — the #1 thing Anthropic says most engineers get wrong.

UI Layout:

Left panel: Input form — Topic field, Timeframe dropdown (1 week / 2 weeks / 1 month), Hours per day slider, Skill level (Beginner / Intermediate / Advanced)

Right panel: Agent progress area — shows each step the agent takes in real-time ('Analyzing topic...', 'Breaking into subtopics...', 'Generating Day 1 plan...')

Below: Final output — collapsible day-by-day plan. Each day shows: topic, learning objectives, resource links, estimated time, quiz questions

Bottom: 'Regenerate' button to get an alternative plan

Core Functionality:

- Multi-step agent loop using AI SDK v6 **Agent** abstraction with **maxSteps: 10**
- Real-time progress streaming — users see what the agent is doing at each step
- Structured output per day: topic, objectives, resources, time estimate, quiz
- Skill-level adaptation — beginner gets more basics, advanced skips fundamentals
- 'Regenerate' creates a different plan by adjusting the system prompt with randomness

Build it step by step:

Step 1: Create Next.js project with AI SDK v6

Step 2: Define the Agent using **new Agent({ model, system, tools, maxSteps })**

Step 3: Write a detailed system prompt: 'You are a study planner. Step 1: Analyze the topic. Step 2: Break into subtopics. Step 3: Order by difficulty. Step 4: Assign to days. Step 5: Add resources. Step 6: Create quizzes.'

Step 4: Set maxSteps to 10 — the agent decides how many steps it needs

Step 5: Build the input form UI with topic, timeframe, hours, and skill level

Step 6: Stream agent progress to the UI — show each step as it happens using event callbacks

Step 7: Format the final output as collapsible day cards with structured content

Uses only gpt-5-mini. No external APIs.

What you'll learn: Agent abstraction, maxSteps, agent loops, streaming progress, when NOT to use agents.

Advanced (optional): Add a web search tool to find real resource links instead of generated ones.

Common Mistakes to Avoid

✗ Don't use agents for everything. Most tasks need a single prompt, not an agent. Read the Anthropic article first.

✗ Don't set maxSteps too high (>20). Infinite loops burn money fast.

✗ Don't hide the agent's thinking. Users need to see progress — a 30-second spinner feels broken.

WEEK 4

Intermediate

Tool Use, Function Calling & Workflows

Pace: Days 1-2: Read tool calling docs | Days 3-4: Study workflows | Days 5-7: Build project

By the end of this week, you will:

✓ Define tools with Zod schemas that the LLM calls reliably
✓ Build multi-tool agents where AI picks the right tool
✓ Understand workflow patterns: chaining, routing, parallelization
✓ Display tool calls and results inline in chat

Read / Study

• Vercel AI SDK v6 — Tool Calling → https://ai-sdk.dev/docs/ai-sdk-core/tools-and-tool-calling
• Vercel AI SDK v6 — Workflows → https://ai-sdk.dev/docs/agents/workflows
• Vercel AI SDK v6 — Subagents → https://ai-sdk.dev/docs/agents/subagents
• Vercel AI SDK v6 — MCP Tools → https://ai-sdk.dev/docs/ai-sdk-core/mcp-tools
• OpenAI Function Calling → https://platform.openai.com/docs/guides/function-calling
• Anthropic Tool Use → https://platform.claude.com/docs/en/intro

PROJECT: Coding Assistant (Mini Cursor)

What you're building:

A chat-based coding assistant with 5 tools that the AI can call autonomously. Users talk to it naturally — 'read my file and find bugs', 'explain this function', 'generate a React component for a login form' — and the AI decides which tool(s) to invoke.

This is a simplified version of how tools like Cursor and GitHub Copilot Chat work. The magic is in the tool definitions — the AI reads your descriptions to decide when to call each tool.

Why this matters: Tool calling is what transforms an LLM from a text generator into something that can take actions. Every production AI app uses tools — search, databases, APIs, code execution.

UI Layout:

Full-width chat interface (like ChatGPT/Cursor) with message input at bottom

Chat messages: User messages on right, AI responses on left, Tool calls shown as collapsible cards between messages ('Called: findBugs → Found 3 issues')

Top bar: App title + file upload button (for readFile tool)

Tool call cards: Show tool name, input params, and result — collapsible to save space

Core Functionality:

- **readFile** — reads uploaded file contents and returns the text
- **searchCodebase** — searches a mock documentation database (hardcode 30 entries from React/Next.js docs)
- **explainCode** — takes code, returns line-by-line explanation
- **findBugs** — analyzes code for bugs, security issues, and anti-patterns
- **generateCode** — generates code from natural language description
- Multi-step workflows: 'read my file and find bugs' → agent calls readFile, then findBugs automatically
- Tool call visualization in chat — users see which tool was called and its result

Build it step by step:

Step 1: Create Next.js project with AI SDK v6

Step 2: Define 5 tools using Zod schemas — each with a clear name, description (explain WHEN to call it), and parameter types

Step 3: Implement each tool as a function: readFile reads from uploaded files, searchCodebase does string matching over hardcoded entries, etc.

Step 4: Create an Agent with all 5 tools. The AI decides which tool to call based on user input

Step 5: Build the chat UI using **useChat** hook. Render tool calls as special cards in the message stream

Step 6: Test multi-step: ask 'read the file and explain what it does' — should trigger readFile then explainCode

Step 7: Add file upload support for the readFile tool

What you'll learn: Tool definitions with Zod, multi-tool routing, workflows, tool call UI, MCP concepts.

Advanced (optional): Add MCP integration to connect to external tools like GitHub API.

Common Mistakes to Avoid

✗ Don't write vague tool descriptions. The LLM uses your description to decide WHEN to call it. Be painfully explicit.

✗ Don't give tools too many parameters. Simple, focused tools work better than complex ones.

✗ Don't forget tool error handling. If a tool fails, the agent should tell the user, not crash.

WEEK 5

Intermediate

RAG — Retrieval Augmented Generation

Pace: Days 1-2: Read RAG + embeddings docs | Days 3-4: Set up Supabase | Days 5-7: Build project

By the end of this week, you will:

✓ Explain what embeddings are and why similar text produces similar vectors

✓ Chunk documents intelligently

✓ Store and query embeddings in a vector database

✓ Build a RAG pipeline with cited answers

Read / Study

- Vercel AI SDK v6 RAG Guide → https://sdk.vercel.ai/docs/guides/rag-chatbot
- Vercel AI SDK v6 — Embeddings → https://ai-sdk.dev/docs/ai-sdk-core/embeddings
- Vercel AI SDK v6 — Reranking → https://ai-sdk.dev/docs/ai-sdk-core/reranking
- Google Gemini File Search API → https://ai.google.dev/gemini-api/docs/file-search
- Ragas — RAG Evaluation → https://docs.ragas.io/

PROJECT: Chat With Any GitHub Repo

What you're building:

A web app where developers paste a GitHub repo URL. The app fetches the entire codebase, intelligently chunks it (splitting by functions and classes, not arbitrary character limits), creates embeddings, stores them in a vector database, and provides a chat interface where users ask questions about the code.

Every answer includes citations — which file and line range the information came from. Users can click 'Show source' to see the actual code chunk that informed the answer.

Why this matters: RAG is the #1 most in-demand AI engineering skill. Almost every enterprise AI use case — internal knowledge bases, customer support, document Q&A; — is built on RAG.

UI Layout:

Top: URL input field ('Paste a GitHub repo URL...') + 'Index Repo' button with progress bar

Left panel: Indexed files list — collapsible tree view of the repo structure, shows which files were indexed

Right panel: Chat interface — ask questions about the code, get answers with inline citations

Citations: Each answer has clickable [file.ts:L12-L45] references. Click to expand and see the source code chunk

Core Functionality:

- GitHub repo ingestion via GitHub API (public repos, no auth needed)
- Code-aware chunking: split by functions/classes/methods, not arbitrary characters. Split README by sections
- Embeddings using AI SDK's **embedMany** function, stored in Supabase pgvector
- Similarity search: embed the user's question, find top 5 similar chunks
- Reranking using AI SDK's **rerank** for better retrieval quality
- Cited answers: every claim references [filename:lines]
- 'Show source' expands to reveal the actual code chunk

Build it step by step:

Step 1: Set up free Supabase project, enable pgvector extension, create embeddings table

Step 2: Create Next.js project. Build the repo URL input form

Step 3: Fetch repo files using GitHub API — filter to code files only (.ts, .js, .py, .md, etc.)

Step 4: Chunk intelligently: use regex to split by function/class definitions. For markdown, split by headers

Step 5: Embed all chunks using **embedMany** and insert into Supabase with metadata (filename, line numbers)

Step 6: Build chat interface with useChat. On each question: embed query → pgvector similarity search → pass top 5 chunks as context

Step 7: Format answers with citations. Add 'Show source' expandable sections

Prerequisites: Free Supabase account with pgvector extension.

What you'll learn: Embeddings, chunking, vector storage, similarity search, reranking, cited answers.

Common Mistakes to Avoid

✗ Don't chunk at arbitrary character boundaries — you'll split functions in half and get useless retrieval.

✗ Don't embed entire files as one chunk. Large chunks = poor precision.

✗ Don't skip citations. RAG without sources is just a chatbot that might be hallucinating.

✗ Don't blame the LLM for bad answers — check retrieval first. Most RAG failures are retrieval failures.

WEEK 6

Intermediate

Multi-Modal — Vision & Beyond Text

Pace: Days 1-2: Read multi-modal docs | Days 3-7: Build project

By the end of this week, you will:

✓ Send images to LLMs and get structured analysis back

✓ Combine vision with structured outputs (image in, JSON out)

✓ Build iterative refinement flows

Read / Study

• Vercel AI SDK v6 — Prompts (multi-modal) → https://ai-sdk.dev/docs/foundations/prompts

• OpenAI Images & Vision → https://developers.openai.com/api/docs/guides/images-vision

• Anthropic Vision + PDF → https://platform.claude.com/docs/en/intro

PROJECT: UI to Code Converter

What you're building:

A web app where users upload a screenshot of any website, app UI, or even a hand-drawn wireframe. The AI analyzes the visual design — layout, colors, typography, components — and generates clean, production-ready code that recreates it.

Users then refine iteratively: 'make the header blue', 'add a sidebar', 'change the font to monospace'. The AI modifies the code based on feedback, and users see a live preview update in real-time.

Why this matters: Multi-modal AI (vision + text) is the next frontier. Products like v0.dev, Screenshot-to-Code, and Figma AI all use this pattern. This is one of the most impressive demos you can build.

UI Layout:

Left panel: Image upload area (drag & drop or file picker) showing the uploaded screenshot

Top right: Framework toggle (React + Tailwind / Vue / HTML+CSS) + 'Generate Code' button

Right panel (top): Live preview — iframe rendering the generated code in real-time

Right panel (bottom): Code editor showing the generated source with syntax highlighting

Below: Refinement chat — 'Make the header blue', 'Add a login form' — AI modifies code, preview updates

Buttons: 'Copy Code' and 'Download as File'

Core Functionality:

- Image upload with drag-and-drop and file preview
- Vision analysis using gpt-5 — layout recognition, color extraction, component identification
- Framework selector changes the system prompt and output format
- Live preview in iframe — generated code renders immediately
- Iterative refinement via chat — user gives feedback, AI modifies code, preview updates
- 'Copy Code' and 'Download as .tsx / .vue / .html file' buttons

Build it step by step:

Step 1: Create Next.js project with AI SDK v6

Step 2: Build drag-and-drop image upload component using React state

Step 3: Create API route: send image to gpt-5 with system prompt: 'Analyze this UI and generate production-ready [framework] code'

Step 4: Add framework toggle that changes the system prompt's output format

Step 5: Render generated code in an iframe for live preview — use srcdoc attribute

Step 6: Add refinement chat: user types feedback, API sends original image + previous code + feedback to generate updated code

Step 7: Add 'Copy Code' (navigator.clipboard) and 'Download' (Blob + URL.createObjectURL) buttons

Uses gpt-5 (vision is built-in to all GPT-5 models).

What you'll learn: Image inputs, vision capabilities, multi-modal prompts, iterative refinement, code generation from images.

Common Mistakes to Avoid

✗ Don't use gpt-5-mini for vision — use gpt-5 or gpt-5.4 for better image understanding.

✗ Don't send huge images. Resize to ~1200px width max to save tokens and cost.

✗ Don't expect pixel-perfect output. AI generates approximate layouts — that's the starting point, not the end.

WEEK 7

Advanced

Multi-Provider, Evaluation & Security

Pace: Days 1-2: Read provider + eval docs | Days 3-4: Set up Langfuse | Days 5-7: Build project

By the end of this week, you will:

✓ Switch between multiple LLM providers
✓ Trace and debug every LLM call using Langfuse
✓ Understand prompt injection and basic defenses
✓ Compare model speed, cost, and quality

Read / Study

• Vercel AI SDK v6 — Provider Management → https://ai-sdk.dev/docs/ai-sdk-core/provider-management
• Vercel AI SDK v6 — Testing → https://ai-sdk.dev/docs/ai-sdk-core/testing
• Vercel AI SDK v6 — DevTools → https://ai-sdk.dev/docs/ai-sdk-core/devtools
• OpenAI Reasoning Models → https://platform.openai.com/docs/guides/reasoning
• Langfuse — LLM Observability → https://langfuse.com/docs/observability/overview
• OWASP Top 10 for LLM Apps → https://genai.owasp.org/llmrisk/llm01-prompt-injection/

PROJECT: AI Model Arena

What you're building:

A web app that runs the same prompt through 3 different GPT-5 model tiers simultaneously and displays responses side by side with detailed metrics. Users can compare speed, quality, cost, and reasoning depth — then vote on which response they prefer.

Integrated with Langfuse for full observability: every LLM call is traced with prompt, response, latency, token usage, and cost. Includes a 'Security Test' tab where users try prompt injection attacks.

Why this matters: In production, choosing the right model for each task saves thousands of dollars. And if you can't trace your LLM calls, you can't debug, optimize, or improve them.

UI Layout:

Top: Prompt input field (large textarea) + 'Run All' button

Middle: 3-column layout showing responses side by side — each column has: model name badge, streaming response, metrics card (time, tokens, cost, reasoning effort)

Below each response: Vote button ('This is best') — shows win rates over time

Tab bar: 'Compare' tab (main) + 'Security Test' tab (prompt injection testing) + 'Langfuse' link to dashboard

Security tab: Pre-loaded injection prompts to test. Shows how each model handles them

Core Functionality:

- 3 models running in parallel using Promise.all with streaming
- Side-by-side display with live streaming for all 3 simultaneously
- Per-response metrics: response time (ms), token count, estimated cost, reasoning effort level
- Langfuse integration — every call traced with full details in external dashboard
- Voting system with persistent win rate tracking
- Prompt injection test suite with pre-loaded attack patterns

Build it step by step:

Step 1: Create Next.js project with AI SDK v6

Step 2: Configure 3 model instances: gpt-5-mini, gpt-5 (reasoning: medium), gpt-5.4 (reasoning: high)

Step 3: Create API route that runs all 3 in parallel using Promise.all, streaming each response

Step 4: Build 3-column comparison UI — each column streams independently

Step 5: Add metrics: wrap each call with timing, extract token counts from response metadata

Step 6: Integrate Langfuse: install SDK, wrap LLM calls with Langfuse trace, verify traces in dashboard

Step 7: Build voting system: store votes in localStorage, calculate and display win rates

Step 8: Add Security Test tab with 5 pre-loaded injection prompts

Uses only OpenAI API — no multiple provider keys needed.

What you'll learn: Provider switching, reasoning effort, cost/quality tradeoffs, Langfuse observability, prompt injection.

Advanced (optional): Add Anthropic Claude and Groq Llama as additional providers.

Common Mistakes to Avoid

✗ Don't skip Langfuse setup. If you can't see what's happening in production, you can't fix what's broken.

✗ Don't assume expensive = better. gpt-5-mini wins on many simple tasks and is 10x cheaper.

✗ Don't ignore prompt injection. If you ship without knowing about it, users will exploit it.

WEEK 8

Advanced

Ship It — Full Product (Capstone)

Pace: Days 1-2: Plan | Days 3-5: Build MVP | Days 6-7: Deploy + polish

By the end of this week, you will:

✓ Have a deployed, production-ready app on a live URL anyone can visit

✓ Combine agents, tools, RAG, vision, and multi-provider into one product

✓ Handle auth, persistence, and errors like a real product

✓ Have a portfolio centerpiece for interviews and LinkedIn

Read / Study

• Vercel AI SDK v6 — Chatbot Persistence → https://ai-sdk.dev/docs/ai-sdk-ui/chatbot-message-persistence

• Vercel AI SDK v6 — Error Handling → https://ai-sdk.dev/docs/ai-sdk-core/error-handling

• Vercel AI SDK v6 — Telemetry → https://ai-sdk.dev/docs/ai-sdk-core/telemetry

• Matt Pocock — AI SDK Tutorial (16 free lessons) → https://www.aihero.dev/vercel-ai-sdk-tutorial

• Vercel: How to Build AI Agents → https://vercel.com/kb/guide/how-to-build-ai-agents-with-vercel-and-the-ai-sdk

PROJECT: AI-Powered Developer Dashboard

What you're building:

A complete, production-ready Next.js app that combines everything from all 7 weeks into one polished product. Think of it as your personal AI workspace — a single app where you can chat with documents, review code, plan projects, analyze screenshots, and switch between models.

This is not a tutorial project. This is a real product you deploy, share with people, and put on your resume.

Why this matters: This is the project interviewers will look at. A deployed, working app with auth, persistence, and multiple AI features says more than 100 tutorial certificates.

UI Layout:

Sidebar: Navigation — Chat, Documents, Code, Screenshots, Settings. User profile + logout at bottom

Chat page: Conversation list (left) + active chat (right) with streaming. Model selector dropdown in chat header

Documents page: File upload area + indexed documents list + Q&A; chat with citations (RAG)

Code page: Code input + tool-powered assistant (review, explain, generate, find bugs)

Screenshots page: Image upload + AI analysis + code generation with live preview

Settings: Default model selection, API key management, conversation history clear

MVP Features (must have):

- Chat interface with streaming + conversation history persistence
- Document Q&A; — upload PDF/text, ask questions with cited answers (RAG from Week 5)
- Code assistant — review, explain, and generate code using tools (Week 4)
- Model selector — switch between gpt-5-mini / gpt-5 / gpt-5.4 per conversation (Week 7)
- Auth — user accounts with NextAuth or Clerk
- Deploy on Vercel — live URL anyone can visit

Stretch Goals (nice to have):

- Screenshot analysis — upload images for UI code generation (Week 6)
- Agent-powered planning — multi-step research and planning (Week 3)
- Langfuse integration for tracing (Week 7)
- Rate limiting and cost tracking per user
- Dark mode and polished UI

Build it step by step:

Step 1: Plan architecture: pages, API routes, database schema, MVP vs stretch features

Step 2: Set up Next.js + AI SDK v6 + auth (NextAuth or Clerk) + database (Supabase or Prisma)

Step 3: Build chat interface with useChat hook + message persistence to database

Step 4: Add Document Q&A;: file upload → chunk → embed → store → RAG query with citations

Step 5: Add Code Assistant: wire up tools from Week 4 (readFile, explainCode, findBugs, generateCode)

Step 6: Add model selector that changes provider config per conversation

Step 7: Deploy to Vercel. Test with 3 real users. Fix the bugs they find.

Step 8: Polish UI, write a proper README, push to GitHub, share on LinkedIn

This is your portfolio centerpiece. Make it look professional. First impressions matter.

Common Mistakes to Avoid

✗ Don't build every feature at once. Ship MVP first, then add stretch goals one at a time.

✗ Don't skip auth. A deployed app without auth = anyone burns your API credits overnight.

✗ Don't make it ugly. Spend time on UI — this is what interviewers see first.

✗ Don't forget the README. Explain what it does, tech used, how to run it, and what you learned.

BONUS

Additional Resources & Courses

Build Your Own LLM (Optional Deep Dive)

For those who want to understand what's happening under the hood. Requested by 25% of the audience.

- MIT OCW — Hands-on Deep Learning → https://ocw.mit.edu/courses/15-773-hands-on-deep-learning-spring-2024/
- Book — Build a Large Language Model from Scratch (Raschka) → https://www.amazon.in/Build-Large-Language-Model-Scratch/dp/1633437167
- Andrej Karpathy — Let's Build GPT from Scratch → https://www.youtube.com/watch?v=kCc8FmEb1nY

Free Official Courses from AI Companies

Learn directly from the companies building the models. All free or free-to-audit.

Anthropic (Claude)

- AI Fluency Framework: Foundations → https://anthropic.skilljar.com/ai-fluency-framework-foundations

OpenAI (GPT-5)

- OpenAI Academy (free courses + upcoming certifications) → https://academy.openai.com/
- Build AI Agents with OpenAI Tools — Coursera Professional Certificate (free to audit) → https://www.coursera.org/professional-certificates/build-ai-agents-with-openai-tools

Google (Gemini)

- Gemini API by Google — Udacity (free, by UC Berkeley) → https://www.udacity.com/course/gemini-API-by-google--cd13416
- Google AI Professional Certificate — Coursera (free to audit) → https://www.coursera.org/professional-certificates/google-ai

What You Walk Away With

- **8 working projects** on your GitHub portfolio
- Hands-on experience with **GPT-5 family** and Vercel AI SDK v6
- A **deployed capstone** — live URL you share anywhere
- **Agent building** — loops, tools, workflows, and when NOT to use agents
- **RAG systems** — embeddings, vector DBs, reranking, citations
- **Multi-modal** — vision, code generation from screenshots
- **Evaluation & security** — Langfuse tracing, prompt injection defense
- Enough depth to **confidently discuss LLMs in any interview**

What's Next After This Roadmap?

You've built 8 projects and deployed a real product. Here's what to do next:

- **Apply for jobs.** You have a portfolio, deployed projects, and real skills. Don't wait until you feel 'ready' — start now.
- **Freelance.** Offer AI integration to businesses. RAG systems, chatbots, and automation are in massive demand.
- **Keep building.** Pick a niche (legal AI, dev tools, healthcare) and build something specific. Niche expertise > general knowledge.
- **Share your work.** Post weekly on LinkedIn and Twitter. The best opportunities come to visible people.
- **Go deeper.** Fine-tuning, evaluation pipelines, production scaling. The bonus section is your starting point.

Join the Community

Want guided help, project reviews, and a community of builders?

I'm building a community for serious AI learners. Stay tuned — follow me for the announcement:

Instagram → @praneeth_kalluri

YouTube → @praneethkalluri

LinkedIn → Praneeth Kalluri

If this roadmap helped you, share it with someone who needs direction.

The best way to learn is to build. Start today.