The 90-Day AI Engineer Roadmap

From working developer to production-ready AI engineer — a week-by-week plan you can run around a full-time job.

A free guide from ProductionAIEngineer.com · v1.0 · Source file — deliver as the lead-magnet PDF and cross-reference at `/roadmap`.

You already know how to code. You don't need a PhD, a research background, or a math bootcamp. You need a **map**, a **stack of real projects**, and the **production judgment** that gets engineers hired. This roadmap gives you all three in 90 days — 6 stages, 12 working weeks, 5 portfolio projects, and a scorecard that tells you when you're interview-ready.

How to use this roadmap

Who this is for

- Backend, full-stack, cloud, DevOps, and data engineers moving into AI.
- People who learn by shipping, not by watching.
- Anyone who wants an AI role and is willing to put in ~10 focused hours a week.

Who this is *not* for

- People looking for a no-code shortcut or a certificate to frame.
- Researchers optimizing model internals (this is *applied* AI engineering).

The promise (your outcomes by Day 90)

- 2–3 deployed, production-shaped AI projects on GitHub with READMEs and diagrams.
- A working command of RAG, agents, evals, observability, and cost control.
- A rehearsed interview story for each project and a system-design vocabulary.
- A portfolio a hiring manager can skim in 60 seconds and trust.

Prerequisites (don't skip)

- Comfortable in one language (Python or TypeScript recommended for AI work).
- Can use Git, the terminal, and read API docs.
- Have (or can create) accounts for one LLM provider and one cloud host.

Time budget & cadence

- **~10 hrs/week:** weekdays 1 hr (read, code a little, engage) + weekend 4–5 hrs (build).
- Consistency beats intensity. A daily 45-minute streak outperforms weekend cramming.

The weekly operating loop (run this every week)

1. **Monday:** read the week's objective; write one sentence on what "done" means.
2. **Tue–Thu:** build in small commits; keep a 3-line daily log (see template below).
3. **Weekend:** finish the deliverable; write/update the README; push to GitHub.
4. **Sunday:** post one "build-in-public" update; check the week's exit criteria.

The map: 6 stages across 90 days

Stage	Weeks	Focus	Primary deliverable
01 · Concept	1–2	LLM fundamentals *as a developer*	A scratch repo proving the primitives work
02 · Build	3–5	Ship real apps, not notebooks	**Project 1 (RAG service)** + **Project 2 (Chat-with-docs)**
03 · Harden	6–7	The production layer	Evals, tracing, controls, safety on Project 1
04 · Deploy	8	Get it running for real	A public URL + Docker + CI
05 · Explain	9	Communicate like a senior	Architecture diagram + README + trade-off notes
06 · Interview	10–12	Convert work into offers	**Project 3 (agent)**, mock interviews, talking points
★ Launch	13 (buffer)	Polish & apply	Applications out, portfolio live

> The 13th week is a deliberate buffer. Life happens; use it to catch up, polish, or start applying.

Week-by-week plan

Each week has the same shape: **Objectives** → **Do** → **Deliverable** → **Exit criteria** → **Pitfalls.** If you can't tick the exit criteria, repeat the week before moving on — the gates are the point.

Stage 01 · Concept

Week 1 — LLM primitives

- **Objectives:** Understand tokens, context windows, temperature, system vs user messages, and structured (JSON) outputs.
- **Do:** Call an LLM API from code. Get a *reliable* JSON response using a schema. Measure tokens and cost per call. Try the same prompt at temperature 0 vs 1 and note the difference.
- **Deliverable:** `llm-primitives/` scratch repo with 4 tiny scripts (chat, JSON output, token count, streaming).
- **Exit criteria:** You can explain, in your own words, what a context window is and why structured outputs matter for real apps.

- **Pitfalls:** Treating the LLM as deterministic; forgetting that JSON must be *validated*, not trusted.

Week 2 — Embeddings & retrieval intuition

- **Objectives:** Understand embeddings, vector similarity (cosine), and the retrieve → ground → answer pattern.
- **Do:** Embed ~50 text chunks, store them (start with a local vector store or `pgvector`), and run a similarity search. Manually inspect: do the top results actually match the query?
- **Deliverable:** `embeddings-lab/` — a script that indexes a folder of markdown and answers a query with the top-k chunks printed.
- **Exit criteria:** You can articulate why similarity search returns "close" but not always "correct" results — the core motivation for everything in Stage 03.
- **Pitfalls:** Mixing embedding models between indexing and query; ignoring chunk quality.

Stage 02 · Build

Week 3 — Project 1: RAG service (ingestion + retrieval)

- **Objectives:** Turn the embeddings lab into a real service with clean boundaries.
- **Do:** Build an ingestion pipeline (load → chunk on structure → embed → store with metadata: title, URL, section). Expose a `/query` endpoint that retrieves top-k with citations.
- **Deliverable:** A running RAG API (FastAPI/Express) over a real corpus (docs you care about).
- **Exit criteria:** `POST /query` returns an answer **with sources**, and you can trace which chunks produced it.
- **Pitfalls:** Fixed-character chunking that splits mid-sentence; dropping metadata you'll later need for citations.

Week 4 — Project 1: answer quality

- **Objectives:** Make answers grounded and trustworthy.
- **Do:** Add a reranking step, grounded-answer prompting ("answer only from context; if unknown, say so"), and citation formatting. Add a simple relevance filter to drop weak chunks.
- **Deliverable:** RAG v2 that refuses to answer when context is missing and always cites.
- **Exit criteria:** On 10 hand-written questions, answers are grounded and cite the right section ≥ 8/10 times.
- **Pitfalls:** Hallucinated citations; letting the model answer from prior knowledge instead of retrieved context.

Week 5 — Project 2: Chat-with-your-docs (full-stack)

- **Objectives:** Wrap Project 1 in a usable product.
- **Do:** Build a minimal chat UI, conversation state, streaming responses, and a real database. Clean API boundary between UI and RAG service.
- **Deliverable:** A deployed-locally full-stack app; README with an architecture diagram.
- **Exit criteria:** A stranger can clone, run, and ask a question in under 10 minutes using only your README.

- **Pitfalls:** Putting API keys in the client; no error/empty states; UI that hides latency instead of streaming.

Stage 03 · Harden

Week 6 — Evals & error analysis

- **Objectives:** Replace vibes with a number.
- **Do:** Create a labeled test set (25–50 Q/A pairs). Add automated checks (groundedness, retrieval hit-rate, answer relevance). Run them, read every failure, and write an error-analysis note.
- **Deliverable:** An eval script that prints a tracked score + a `error-analysis.md` with the top 3 failure modes.
- **Exit criteria:** You can say "my RAG scores X on groundedness; the top failure is Y; here's my fix."
- **Pitfalls:** Only measuring aggregate accuracy; skipping the read-every-failure step (that's where the insight lives).

Week 7 — Observability, controls & safety

- **Objectives:** Add the layer that separates you from tutorial-followers.
- **Do:** Add tracing (inputs, retrieved context, tokens, latency per step). Add retries, timeouts, a response cache, and a cost budget. Add input validation and basic prompt-injection defense.
- **Deliverable:** A trace you can screenshot for interviews + a short `production-notes.md` (controls + threat model).
- **Exit criteria:** You can pull up a trace and explain where time and tokens go, and name one injection you defend against.
- **Pitfalls:** Logging secrets or full PII; adding caching that serves stale/wrong answers.

Stage 04 · Deploy

Week 8 — Ship it for real

- **Objectives:** A public URL beats a localhost demo every time.
- **Do:** Containerize with Docker. Move secrets to environment/secret manager. Deploy to a real host. Add basic CI (lint + tests + eval smoke check on push).
- **Deliverable:** A live URL + a green CI badge on the repo.
- **Exit criteria:** Someone can use your app from their phone; a push runs CI.
- **Pitfalls:** Secrets in the image; no health check; cold-start latency you never measured.

Stage 05 · Explain

Week 9 — Communicate like a senior

- **Objectives:** Interviewers hire clarity, not cleverness.
- **Do:** Draw the architecture (boxes + arrows + data flow). Rewrite each README to the template below. Write a trade-offs section ("I chose X over Y because…").

- **Deliverable:** Polished READMEs + one architecture diagram per project + a 90-second demo video.
- **Exit criteria:** A non-AI engineer understands what you built and why from the README alone.
- **Pitfalls:** Diagrams that show tech logos but no data flow; READMEs that explain *how to run* but not *why it's built this way*.

Stage 06 · Interview

Week 10 — Project 3: tool-using agent

- **Objectives:** Show you know when (and when *not*) to use an agent.
- **Do:** Build a minimal agent: tool/function calling, a memory store, a loop with a stop condition, and a human-in-the-loop checkpoint. Add guardrails and a "why not just a chain?" note.
- **Deliverable:** Project 3 repo with the same production polish (evals-lite, tracing, README).
- **Exit criteria:** You can explain your agent's control flow and one failure mode you guard against.
- **Pitfalls:** Unbounded loops; agent-when-a-function-would-do; no cost ceiling.

Week 11 — System design & the role map

- **Objectives:** Speak the language of AI system-design rounds.
- **Do:** Practice designing: a RAG system, an agent system, and an eval pipeline. Map the roles (AI Engineer, GenAI Developer, Agentic AI Engineer, Forward Deployed Engineer) to your projects.
- **Deliverable:** 3 one-page system-design outlines + a role-fit note.
- **Exit criteria:** You can whiteboard a RAG design in 10 minutes covering ingestion, retrieval, evals, and cost.
- **Pitfalls:** Jumping to a diagram before clarifying requirements; ignoring evals and cost in the design.

Week 12 — Talking points & mock interviews

- **Objectives:** Convert work into offers.
- **Do:** Write STAR stories for each project. Prepare project talking points (problem → approach → trade-offs → result). Do 2–3 mock interviews (peer, community, or record yourself) and score them with the rubric below.
- **Deliverable:** A one-page "interview brag sheet" + recorded/scored mocks.
- **Exit criteria:** You score ≥ 3.5/5 on the mock-interview rubric across the board.
- **Pitfalls:** Memorizing scripts instead of understanding trade-offs; no metrics in your stories.

The 5 portfolio projects (detailed specs)

Build the first three during the 90 days; the last two are stretch/after-launch. Each project needs: a README, an architecture diagram, a short demo, evals, and a **"how I'd discuss this in an interview"** section.

1) Production RAG service *(core — Weeks 3–4)*

- **Problem:** Answer questions over a document corpus with citations and no hallucinations.
- **Must-have:** structured chunking, metadata, reranking, grounded answers, citations, an eval score.
- **Stretch:** hybrid search (keyword + vector), query rewriting, caching.
- **Interview story:** "I took retrieval from demo to production — here's my groundedness score and how I raised it."

2) Chat-with-your-docs app *(core — Week 5)*

- **Problem:** Make the RAG service usable by a non-technical person.
- **Must-have:** streaming chat UI, conversation state, real DB, clean API boundary, deployed.
- **Stretch:** auth, per-user document upload, feedback thumbs that feed evals.
- **Interview story:** "Full-stack ownership — I designed the boundary between UI and the AI service."

3) Tool-using agent *(core — Week 10)*

- **Problem:** A task that genuinely needs planning + tools (not just one LLM call).
- **Must-have:** function calling, memory, a bounded loop, human-in-the-loop, guardrails, cost ceiling.
- **Stretch:** multi-tool routing, retry/repair on tool errors, eval of task success rate.
- **Interview story:** "I can explain *when not* to use an agent — and I guard against runaway loops."

4) Eval harness *(stretch)*

- **Problem:** Measure and compare prompt/model/retrieval changes objectively.
- **Must-have:** a dataset, metrics, a runner, a report that compares two configs.
- **Interview story:** "I treat prompts like code — versioned, tested, and measured."

5) Domain assistant *(stretch)*

- **Problem:** A vertical agent for a specific workflow (use **synthetic/public data only**).
- **Must-have:** a real workflow, retrieval + tools, and a clear evaluation of usefulness.
- **Interview story:** "I can scope an AI product to a domain and prove it works."

The skills checklist (grouped)

Foundations

- [ ] LLM API calls, streaming, and structured (JSON) outputs
- [ ] Prompting patterns: system design, few-shot, grounding, refusal
- [ ] Embeddings + vector search (cosine, top-k, metadata filtering)

RAG

- [ ] Structure-aware chunking + metadata

- [ ] Retrieval, reranking, grounding, citations
- [ ] Hybrid search & query rewriting *(stretch)*

Agents

- [ ] Tools / function calling
- [ ] Memory (short-term + persistent)
- [ ] Bounded loops, planning, human-in-the-loop, guardrails

Production

- [ ] Evals + error analysis (a tracked score)
- [ ] Tracing / observability (inputs, context, tokens, latency)
- [ ] Cost + latency control (caching, budgets, timeouts, retries)
- [ ] Security: input validation, output constraints, prompt-injection defense
- [ ] Docker + deployment + secrets management + basic CI

Communication

- [ ] Architecture diagrams (data flow, not logos)
- [ ] READMEs a stranger can follow
- [ ] Trade-off articulation + STAR stories

Milestones & pass/fail gates

Day	Milestone	Pass criteria (must be true to continue)
14	Primitives proven	You can call an LLM, get validated JSON, and run a similarity search from code
35	Two projects shipped	RAG API cites sources; full-stack app runs from your README in < 10 min
49	Hardened	A tracked eval score exists; you can read a trace and name your top failure mode
56	Deployed	Public URL live; secrets out of code; CI green
63	Explainable	Every project has a diagram, a README, and a trade-offs note
90	Interview-ready	3 projects, scored mocks ≥ 3.5/5, a system-design vocabulary, applications out

---

Self-assessment scorecard

Rate yourself **1–5** on each. Re-score at Day 45 and Day 90. Target: **all ≥ 4** before applying.

Dimension	1 (novice)	3 (competent)	5 (interview-ready)	Day 45	Day 90
LLM fundamentals	Copy-paste prompts	Reliable structured outputs	Can reason about tokens/cost/limits		
RAG	A demo that retrieves	Grounded answers + citations	Reranking + evals + failure analysis		
Agents	Followed a tutorial	Built a bounded tool-user	Knows when *not* to use one		
Evals	"It looks right"	A tracked score	Error analysis drives fixes		
Observability	Print statements	Basic tracing	Explains where time/tokens go		
Deployment	Localhost only	Deployed once	Docker + secrets + CI		
Communication	Code only	README + diagram	Trade-offs + STAR stories		

> **Scoring:** total /35. **< 21** keep building · **21–28** apply to warm/associate roles · **29+** apply broadly with confidence.

Are you interview-ready? (evaluation criteria)

You're ready when **all** of these are true:

- [ ] 3 deployed projects on GitHub, each with a diagram, README, and a demo link.
- [ ] At least one project has a **tracked eval score** and a documented improvement.
- [ ] You can whiteboard a RAG *and* an agent design in 10 minutes each.
- [ ] You can tell a crisp STAR story (with a metric) for every project.
- [ ] You scored ≥ 3.5/5 on the mock-interview rubric.
- [ ] You can explain one security risk and one cost control you implemented.

Printable weekly schedule template

Day	Time	Focus	Done?
Mon	1 hr	Read objective · write "done" sentence	☐
Tue	1 hr	Build (small commit) + daily log	☐
Wed	1 hr	Build (small commit) + daily log	☐
Thu	1 hr	Build (small commit) + daily log	☐
Fri	30 min	Review · unblock · plan weekend	☐

Day	Time	Focus	Done?
Sat	3 hr	Ship the deliverable	☐
Sun	2 hr	README + push + build-in-public post	☐

Daily log (3 lines): *Yesterday I… · Today I'll… · Blocked on…*

Recommended tooling (model-agnostic)

Pick one per row and stay consistent. The *pattern* matters more than the vendor.

Layer	Options (choose one)
LLM provider	OpenAI · Anthropic · an open model via a hosted API
Embeddings	Provider embeddings · open-source (e.g. `bge`, `e5`)
Vector store	`pgvector` (start here) · Qdrant · Weaviate
Framework	None (raw SDK) · LlamaIndex · LangChain — start with *less*
Evals	`promptfoo` · Ragas · a small custom harness
Tracing	Langfuse · Phoenix · OpenTelemetry
App	FastAPI (Py) · Next.js/Express (TS)
Deploy	A container host of your choice + Docker

> **Rule:** start with the *least* framework you can. You learn more wiring one call by hand than importing ten abstractions.

Common pitfalls (and the fix)

- **Notebook trap** → ship services and apps, not `.ipynb` demos. *Fix:* every project deploys.
- **Tutorial hell** → watching ≠ building. *Fix:* one deliverable per week, no exceptions.
- **Chasing frameworks** → collecting tools instead of skills. *Fix:* raw SDK first.
- **No evals** → you can't improve what you don't measure. *Fix:* a tracked score by Week 6.
- **Silent failures** → no tracing = no story. *Fix:* add observability before deploy.
- **Secrets in code** → instant red flag. *Fix:* env/secret manager from Week 8.
- **Diagrams of logos** → show *data flow*. *Fix:* boxes, arrows, and where data moves.
- **Perfectionism** → polishing week 1 forever. *Fix:* the gates, then move on.

Build in public (compounding advantage)

- Post one update per week: what you built, one thing that broke, what you learned.
- Pin your best project repo to your GitHub profile.
- Write a short blog/LinkedIn post per project (this becomes interview fuel).
- Public/synthetic data only — never employer-internal or client-confidential material.

After Day 90

- Apply while polishing (don't wait for "perfect"). The scorecard tells you when.
- Deepen one track: RAG depth, agentic systems, or evals/LLMOps.
- Contribute to an open-source AI project for signal and network.
- Keep a "learning in public" cadence — it's the flywheel that keeps compounding.

Bonus templates

A) Portfolio README template

```
# <Project Name>
One line: what it does and for whom.

## Problem
Why this exists (2–3 sentences).

## Architecture
[diagram] + 3 bullets on data flow.

## Key decisions & trade-offs
- Chose X over Y because…
- Known limitation: …

## Results
- Eval score: … (metric, dataset size)
- Latency / cost: …

## Run it
Prereqs → setup → run (must work in < 10 min).

## How I'd discuss this in an interview
Problem → approach → trade-offs → result (with a metric).
```

B) Architecture-diagram checklist

- [ ] Entry point (user/request) and exit (response) are shown
- [ ] Every external system (LLM, vector store, DB) is a labeled box

- [ ] Arrows show **data flow**, not just connections
- [ ] The "hard part" (retrieval, agent loop, evals) is visible
- [ ] Where secrets live and where caching happens is marked

C) "Explain it" one-pager (per project)

- **In one sentence:** …
- **The hardest problem:** … **How I solved it:** …
- **One trade-off I made:** …
- **One thing I'd do with more time:** …
- **The metric that proves it works:** …

D) Daily standup log

```
YYYY-MM-DD
- Yesterday: …
- Today: …
- Blocked on: …
- Win/learning: …
```

Your next step

You have the map. Now keep the momentum:

1. **Get one practical lesson a week** → subscribe to **Production AI Notes**, the newsletter that walks this roadmap one build at a time.
2. **Nail the hardest project first** → grab the free **Production RAG Checklist** and take Project 1 from demo to production.
3. **When you're ready to accelerate** → the **AI Engineer Interview & Portfolio Kit** turns this roadmap into interview-ready projects (specs, diagrams, question bank, walkthroughs).

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