

SKILLNAVORA · OTHER PROGRAMS · CAREER PATH

AI Engineer: The Complete Path

Everything an AI engineer goes through, from first day to production: the role, Python and software foundations, ML basics, LLM engineering, RAG, agents and MCP, fine-tuning, evals, LLMOps and deployment, data pipelines, security and governance, system design, collaboration, and a 12-month roadmap with projects and interview prep.

Edition 2026 · skillnavora.in

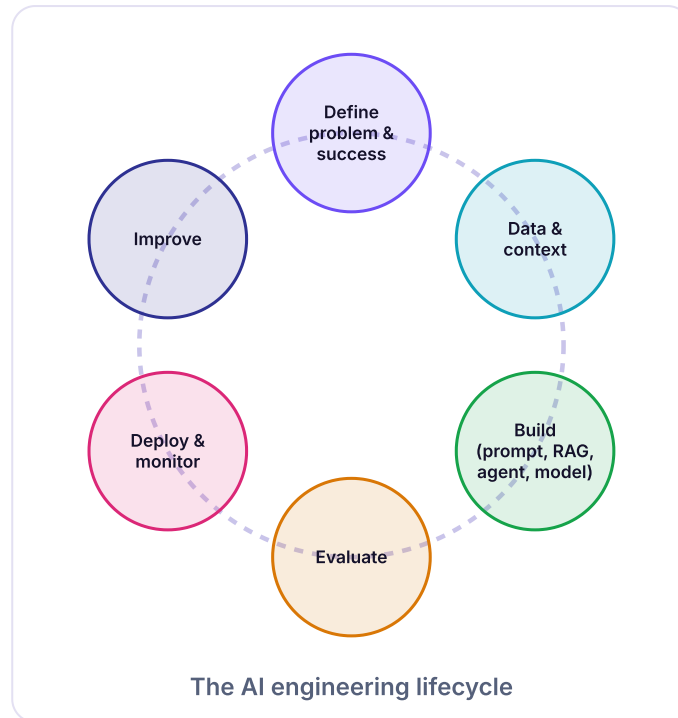
How to use this guide

Each class has goals, concepts with Indian examples, key takeaways and a practice exercise, from basic to advanced. Finish with the final project to build something you can show in interviews.

	Class	Level
Class 1	The AI engineer role and the lifecycle	Basic
Class 2	Foundations: Python, software engineering, Git, APIs, Linux	Basic
Class 3	Math, statistics and machine-learning essentials	Basic
Class 4	LLM engineering: model APIs, prompts, structured output, cost and latency	Intermediate
Class 5	RAG, embeddings and vector databases	Intermediate
Class 6	Agents, tool use and MCP	Intermediate
Class 7	Customising models: fine-tuning, distillation and open-source models	Advanced
Class 8	Evals, observability, guardrails and red-teaming	Advanced
Class 9	Deployment and LLMOps: APIs, containers, cloud, CI/CD, scaling and cost	Advanced
Class 10	Data engineering for AI, security and governance	Advanced
Class 11	System design for AI products and working with the team	Advanced
Class 12	Career: roadmap, portfolio projects, interviews and job descriptions	Advanced
Toolkit: templates, ai engineer tool stack, glossary, 12 interview questions, further reading		

The AI engineer role and the lifecycle

Know what the job is, how it differs from neighbouring roles, and the lifecycle you'll work in.



Role	Mainly does
AI engineer	Builds products on top of models: LLM apps, RAG, agents, evals, deployment
ML engineer	Trains and serves custom models, feature pipelines, MLOps
Data scientist	Analysis, experiments, modelling, insights
Data engineer	Pipelines, warehouses, data quality
Software engineer	Application backend/frontend, systems
Forward-deployed engineer	Builds AI solutions at customer sites
Prompt / context engineer	Designs instructions and context; a skill inside AI engineering

A typical week

- Clarify a use case with the PM; define success metrics and an eval set.
- Prototype with a model API; build retrieval and tools; iterate on failures.
- Write services, tests and CI; deploy; watch dashboards for quality, cost and latency.
- Review incidents, update evals, tune prompts, retrieval and models; document decisions.

Learn it here: [How LLMs work](#) · [Building and shipping a GenAI app](#) · [Forward Deployed Engineer guide](#)

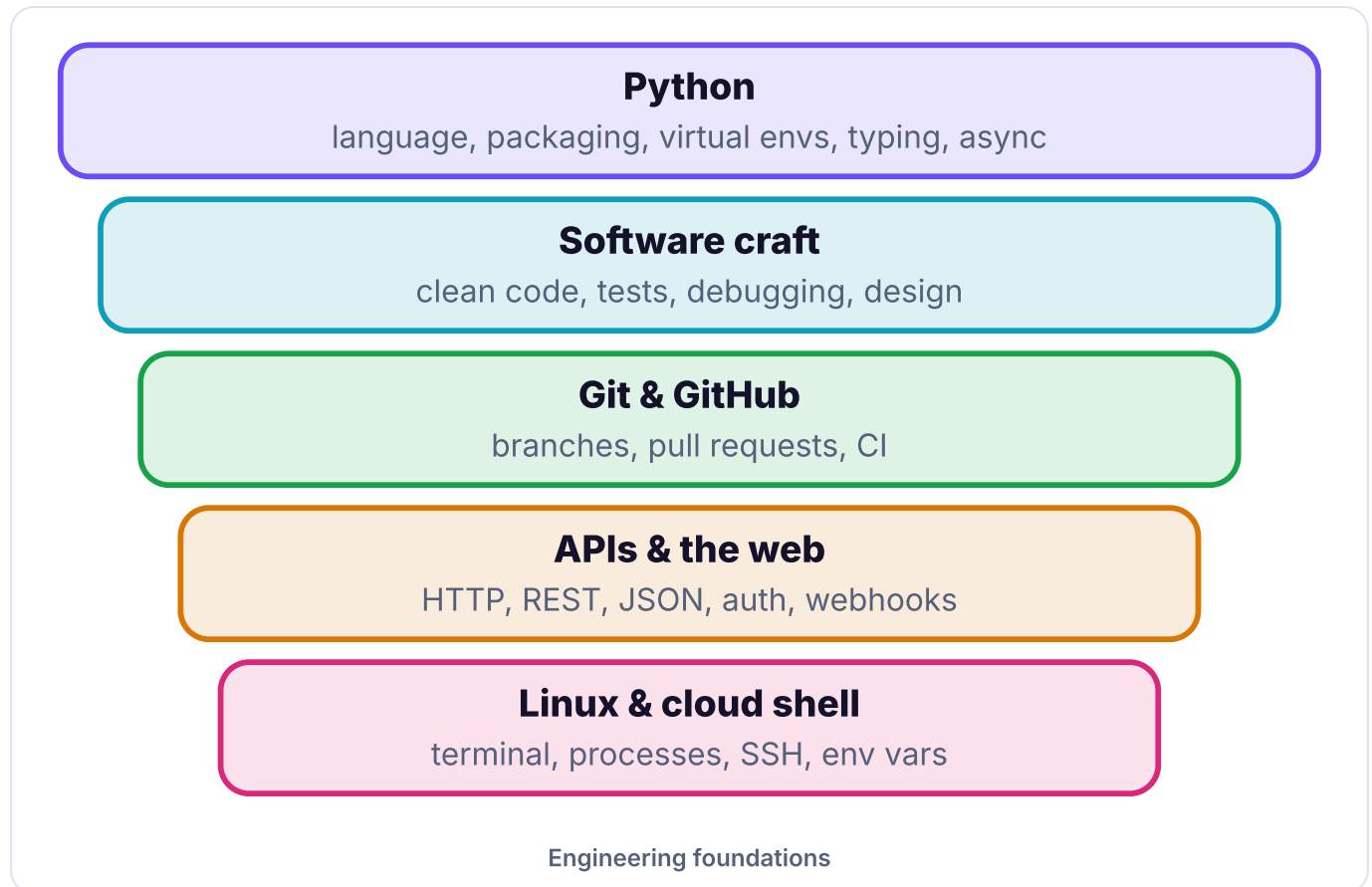
Key takeaways

- AI engineering = software engineering + models + evals + operations.
- Work in a loop: define, build, evaluate, deploy, improve.
- Neighbouring roles overlap; know the boundaries.

Practice exercise: Write a one-page 'day in the life' of an AI engineer for a support-copilot project, with 5 tasks.

Foundations: Python, software engineering, Git, APIs, Linux

Build the engineering base that every AI system sits on.



Skill	Be able to...
Python	Write functions/classes, use virtualenvs and pip/uv, read/write files and JSON, handle errors, use async for many API calls
Testing	pytest, fixtures, mocking API calls, writing tests before you ship
Git	Branch, commit, rebase/merge, open pull requests, resolve conflicts
HTTP/APIs	Call REST APIs, handle auth, rate limits, retries with backoff, webhooks
SQL & data	Joins, aggregates, window functions, indexes; pandas for analysis
Terminal	Navigate, grep, curl, environment variables, secrets handling

Starter script: call a model API with retries

```
import os, time, requests
def call(prompt, tries=3):
    for i in range(tries):
        r = requests.post(URL, headers={"Authorization": f"Bearer {os.environ['API_KEY']}"},
                           json={"model": MODEL, "messages": [{"role": "user", "content":
prompt}}], timeout=60)
        if r.status_code == 200: return r.json()
        if r.status_code in (429, 500, 502, 503): time.sleep(2 ** i); continue
        r.raise_for_status()
    raise RuntimeError("model API unavailable")
```

Learn it here: [Python track \(29 lessons\)](#) · [SQL track](#) · [APIs without code](#) · [APIs, JSON and web scraping](#) · [Python and SQL](#)

Key takeaways

- Python + testing + Git + HTTP are non-negotiable.
- Handle rate limits and retries from day one.
- Never put secrets in code.

Practice exercise: Build a small CLI that calls a model API, handles errors with retries and has 3 pytest tests.

Math, statistics and machine-learning essentials

Know enough ML to choose methods, read metrics and debug models.

Area	Know
Statistics	Distributions, sampling, confidence intervals, hypothesis tests, A/B testing
Linear algebra	Vectors, matrices, dot product (embeddings), similarity
Calculus intuition	Gradients, loss minimisation (how models learn)
ML basics	Supervised/unsupervised, features, train/validation/test, overfitting, cross-validation, regularisation
Metrics	Accuracy, precision, recall, F1, ROC-AUC, MAE/RMSE, calibration
Classic models	Linear/logistic regression, trees, random forest, gradient boosting (XGBoost), k-means
Deep learning	Neural nets, backpropagation, CNNs, RNNs, transformers, attention; PyTorch basics

Learn it here: [Descriptive statistics](#) · [Hypothesis testing](#) · [Cross-validation and tuning](#) · [Gradient boosting](#) · [How LLMs work \(transformers\)](#).

Tools: scikit-learn, pandas, NumPy, PyTorch, Jupyter, Kaggle for practice.

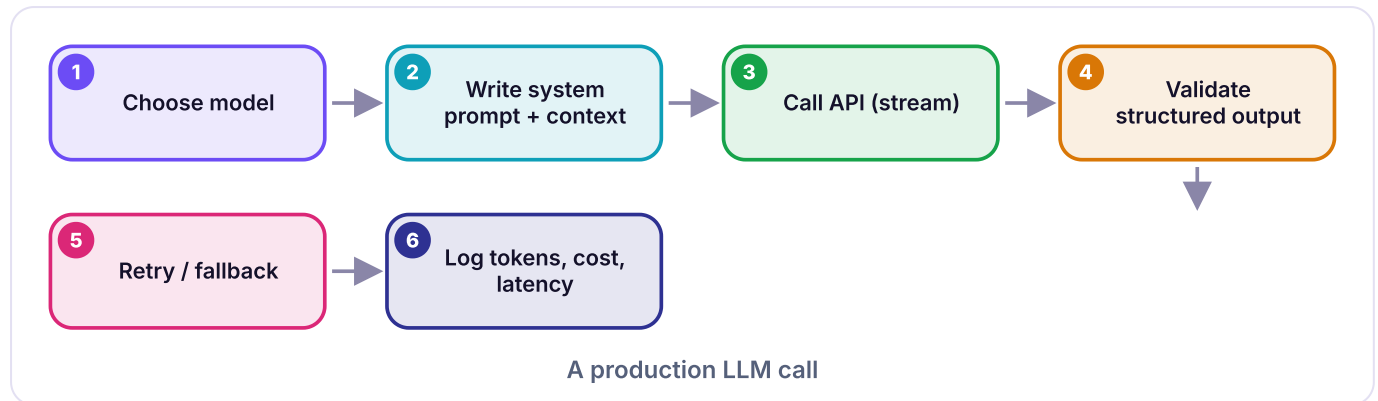
Key takeaways

- You don't need a PhD: need metrics, validation and intuition.
- Embeddings and similarity are linear algebra in use.
- scikit-learn first, PyTorch when needed.

Practice exercise: Train and evaluate a classifier on a Kaggle dataset with proper train/validation/test splits and 3 metrics.

LLM engineering: model APIs, prompts, structured output, cost and latency

Build reliable features on top of model APIs.



- **Providers:** Anthropic Claude, OpenAI, Google Gemini, plus open models (Llama, Mistral, Qwen, DeepSeek) via Hugging Face, Ollama, vLLM, Groq, Together.
- **Key parameters:** system prompt, temperature/sampling (where supported), max tokens, stop sequences, streaming, tool definitions, structured outputs/JSON schema.
- **Tokens & context:** price per input/output token, context window limits, prompt caching, batching for cost.
- **Reliability:** timeouts, retries with backoff, provider fallbacks, circuit breakers, idempotency.
- **Quality:** few-shot examples, chain-of-thought where useful, self-checks, guardrails.
- **Cost & latency:** route simple tasks to small models, cache, trim context, stream for perceived speed. Track cost per successful task.

```

cost per request = input_tokens × price_in + output_tokens × price_out
latency budget   = retrieval + model + tools + post-processing   (set a target for each)
  
```

Learn it here: [Prompt engineering in depth](#) · [Calling LLM APIs from Python](#) · [Prompt & Context Engineering course](#) · [Pricing & Monetising AI](#)

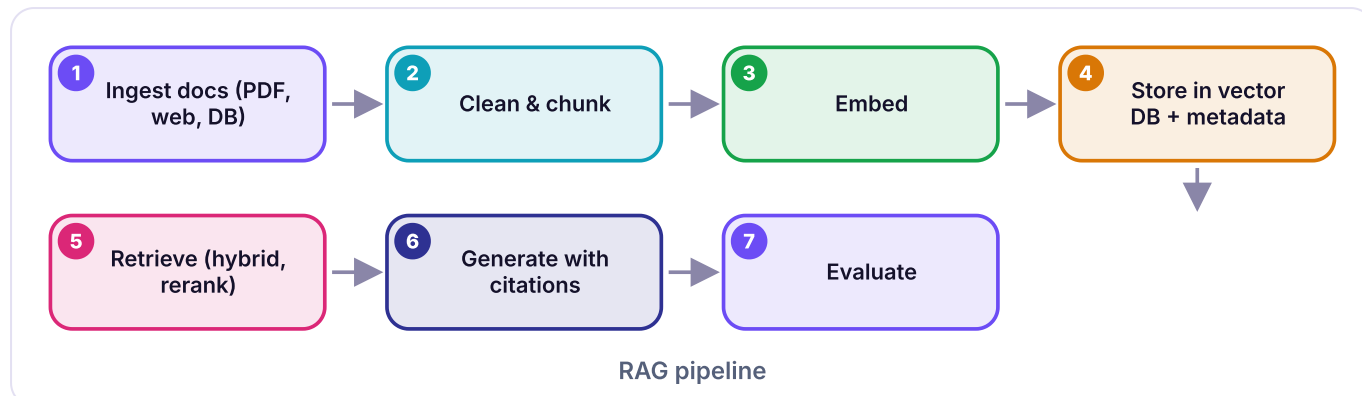
Key takeaways

- Treat the model as an unreliable remote service: validate, retry, fall back.
- Track tokens, cost and latency on every call.
- Structured outputs make LLMs usable in code.

Practice exercise: Build a function that returns validated JSON from an LLM, with retries, a fallback model and a cost log.

RAG, embeddings and vector databases

Build search and question-answering over private data, and measure it.



Part	Options and decisions
Ingestion	Parsers for PDF/HTML/Docs, OCR for scans, incremental refresh, access control metadata
Chunking	By headings/sections, 300–800 tokens, overlap, keep tables intact
Embeddings	OpenAI, Cohere, Voyage, Gemini, open models (bge, e5); multilingual for Hindi
Vector stores	pgvector (Postgres), Pinecone, Qdrant, Weaviate, Milvus, Chroma, FAISS, Elasticsearch/OpenSearch
Retrieval	Hybrid (BM25 + vectors), metadata filters, query rewriting, rerankers (Cohere Rerank, cross-encoders), MMR
Advanced	GraphRAG, agentic RAG, multi-vector, parent-document retrieval, caching
Evaluation	Recall@k, MRR, faithfulness, answer relevance (see AI Evals)

Learn it here: [Embeddings and vector search](#) · [RAG: build, evaluate, improve](#) · [AI Evals](#) · [AI Second Brain](#)

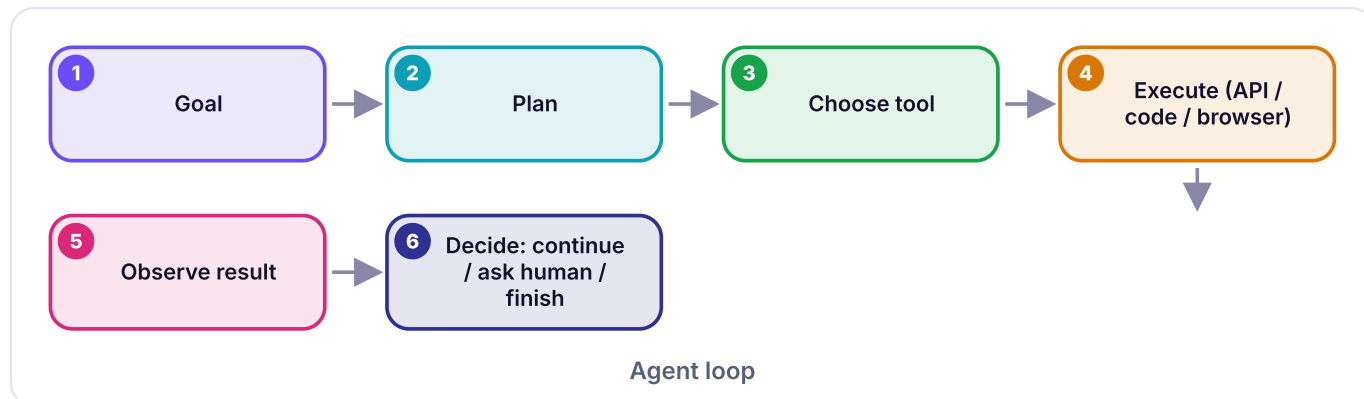
Key takeaways

- Retrieval quality limits answer quality.
- Hybrid search + reranking beat vectors alone.
- Measure retrieval and generation separately.

Practice exercise: Build a RAG Q&A over 20 PDFs with citations; report recall@3 and faithfulness on 30 questions.

Agents, tool use and MCP

Build agents that call tools, with control and observability.



- **Tool/function calling:** describe tools with JSON schemas; the model returns structured calls; your code executes and returns results.
- **MCP (Model Context Protocol):** standard way to expose tools and data to AI assistants; build MCP servers for your product; use existing ones (GitHub, Slack, Playwright, databases).
- **Frameworks:** LangGraph, LangChain, LlamaIndex, OpenAI Agents SDK, Claude Agent SDK, CrewAI, AutoGen, Pydantic AI; or plain code (often the best start).
- **Patterns:** chaining, routing, parallel, orchestrator-workers, evaluator-optimizer (see Orchestration course).
- **Memory:** short-term (conversation), long-term (vector store/db), summaries, user profile.
- **Control:** step/cost limits, approvals for risky tools, sandboxing code execution, tracing every run.
- **Computer & browser use:** Playwright MCP, Browser Use (see Web Automation course).

Learn it here: [Tool use and function calling](#) · [AI agents and agentic AI](#) · [MCP](#) · [Claude Code tutorial](#) · [Agent Orchestration](#) · [Web Automation](#)

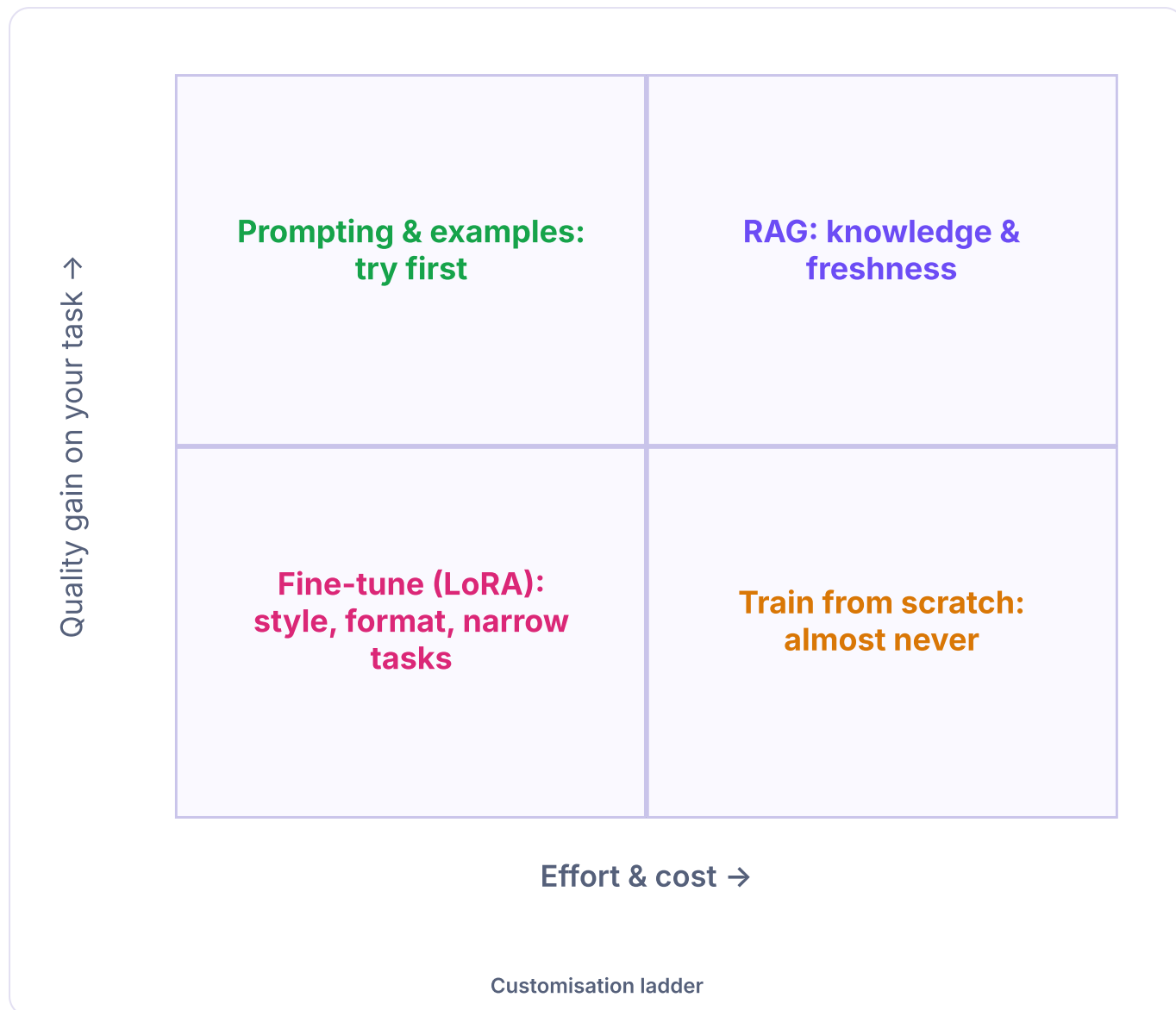
Key takeaways

- Agents = LLM + tools + loop + guardrails.
- Start with simple workflows; add autonomy when evals justify it.
- MCP standardises tool access.

Practice exercise: Build an agent with 3 tools (search, calculator, database lookup) with a step limit and full trace logging.

Customising models: fine-tuning, distillation and open-source models

Decide when to fine-tune, and how, and when not to.



Method	When
Better prompts and examples	Always first
RAG	Knowledge changes or is private; need citations
Supervised fine-tuning (SFT)	Consistent style/format, domain behaviour, shorter prompts, cost reduction
LoRA / QLoRA	Parameter-efficient tuning of open models on one GPU
Preference tuning (DPO, RLHF concepts)	Align outputs with human preferences
Distillation	Teach a small, cheap model from a large one's outputs

Fine-tuning workflow

1. Collect 500–5,000 high-quality examples (inputs and ideal outputs); clean and split train/validation/test.
2. Baseline with prompting; define eval metrics first.
3. Fine-tune (provider fine-tuning APIs, or Hugging Face Transformers, PEFT, TRL, Unsloth, Axolotl).
4. Evaluate against the baseline on held-out data; check regressions and safety.
5. Deploy (vLLM, TGI, Ollama, or provider-hosted); monitor drift; retrain on new data.

Learn it here: [Customising models: prompting vs RAG vs fine-tuning, LoRA](#) · [ML Products for PMs](#)

Key takeaways

- Prompting → RAG → fine-tuning, in that order.
- Data quality beats data quantity.
- Always compare with a prompting baseline.

Practice exercise: Plan (data, metric, baseline, hardware, evaluation) a LoRA fine-tune for classifying support tickets.

Evals, observability, guardrails and red-teaming

Make quality measurable and keep it that way.

- **Evals:** golden dataset, code and LLM-judge graders, human review, regression suite in CI (see AI Evals course).
- **Observability:** trace every request (prompt, retrieved docs, tool calls, tokens, cost, latency); tools: Langfuse, LangSmith, Arize Phoenix, Braintrust, OpenTelemetry.
- **Guardrails:** input checks (PII, injection), output checks (schema, toxicity, policy), allow-lists for tools, rate limits; libraries such as Guardrails AI, NeMo Guardrails, Llama Guard, provider moderation APIs.
- **Red-teaming:** jailbreaks, prompt injection, data exfiltration, harmful content, bias tests; add each finding to the eval set.
- **Online quality:** sample live traffic, judge scores, user feedback, A/B tests.

Learn it here: [Evaluating LLM applications and guardrails](#) · [AI Evals for PMs](#) · [AI Governance](#)

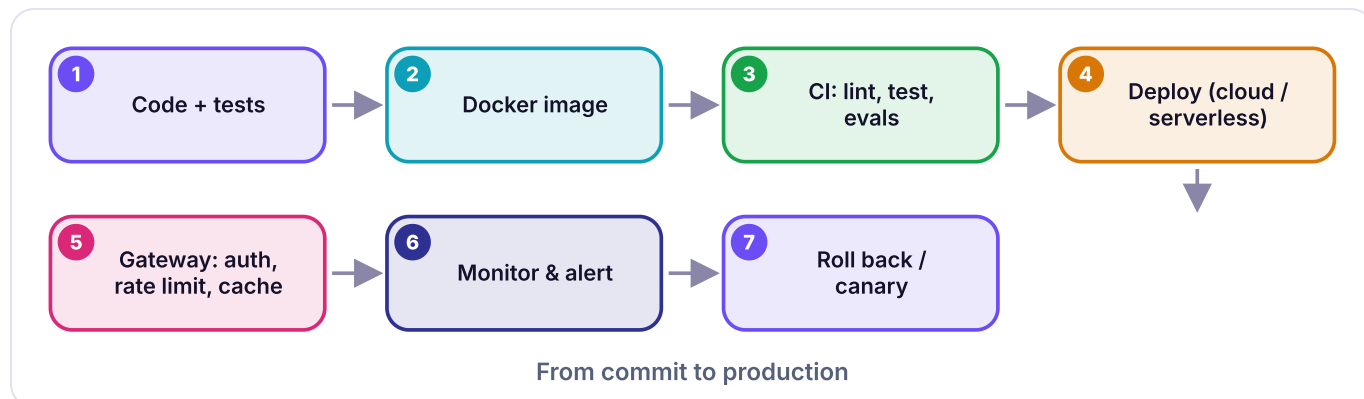
Key takeaways

- No eval, no launch.
- Trace everything to debug fast.
- Guardrails are layers, not a single filter.

Practice exercise: Add tracing and a 50-case regression eval to your RAG app and run it in CI.

Deployment and LLMOps: APIs, containers, cloud, CI/CD, scaling and cost

Ship AI services that are reliable, fast and affordable.



Area	What to know
Serving	FastAPI/Flask/Node services, streaming (SSE/WebSockets), background queues (Celery, SQS, Pub/Sub), async I/O
Containers	Docker, docker-compose, Kubernetes basics
Cloud	AWS (Bedrock, Lambda, ECS, S3), Google Cloud (Vertex AI, Cloud Run), Azure (OpenAI Service, Container Apps); Cloudflare Workers/Pages
GPUs & model serving	vLLM, TGI, Triton, Ollama; quantisation (GGUF, AWQ); batching; autoscaling
Gateways	LiteLLM, OpenRouter, Cloudflare AI Gateway: routing, fallbacks, caching, budgets
Release safety	Feature flags, canary releases, A/B tests, prompt/model versioning, rollbacks
Reliability	Timeouts, retries, circuit breakers, idempotency, graceful degradation
Cost control	Budgets and alerts, model routing, caching, batching, trimming context, monitoring cost per successful task

Learn it here: [Building and shipping a GenAI app](#) · [n8n tutorial](#) · [Agent Orchestration & Monitoring](#)

Key takeaways

- Containerise, test, evaluate, deploy, monitor.
- Use a gateway for routing, caching and budgets.
- Version prompts and models like code.

Practice exercise: Deploy your RAG API with Docker on a cloud service with a health check, logs and a cost alert.

Data engineering for AI, security and governance

Feed AI systems good data and keep them safe and compliant.

Data for AI

- Pipelines (Airflow, Prefect, dbt) that clean, deduplicate and refresh documents and vector indexes.
- Data quality checks, lineage, versioned datasets; labelling workflows (Label Studio, human review).
- Synthetic data with review; privacy: mask PII before it reaches prompts, logs or training.

Security (OWASP Top 10 for LLM apps)

Risk	Control
Prompt injection (direct and from documents/web)	Treat content as data; separate instructions; restrict tool permissions; approvals
Sensitive data leakage	Access control in retrieval, redaction, no secrets in prompts, tenant isolation
Insecure output handling	Validate/escape model output; never run model-written code unsandboxed
Excessive agency	Least privilege tools, spend/step limits, human approval
Supply chain (models, packages)	Pin versions, scan dependencies, trusted model sources
Denial of wallet	Rate limits, quotas, budgets

Governance

DPDP Act consent and minimisation, EU AI Act risk tiers, audit logs, system cards, incident response (see AI Governance course).

Learn it here: [AI Governance & Responsible AI](#) · [Data modelling](#) · [Web Automation \(secure webhooks\)](#).

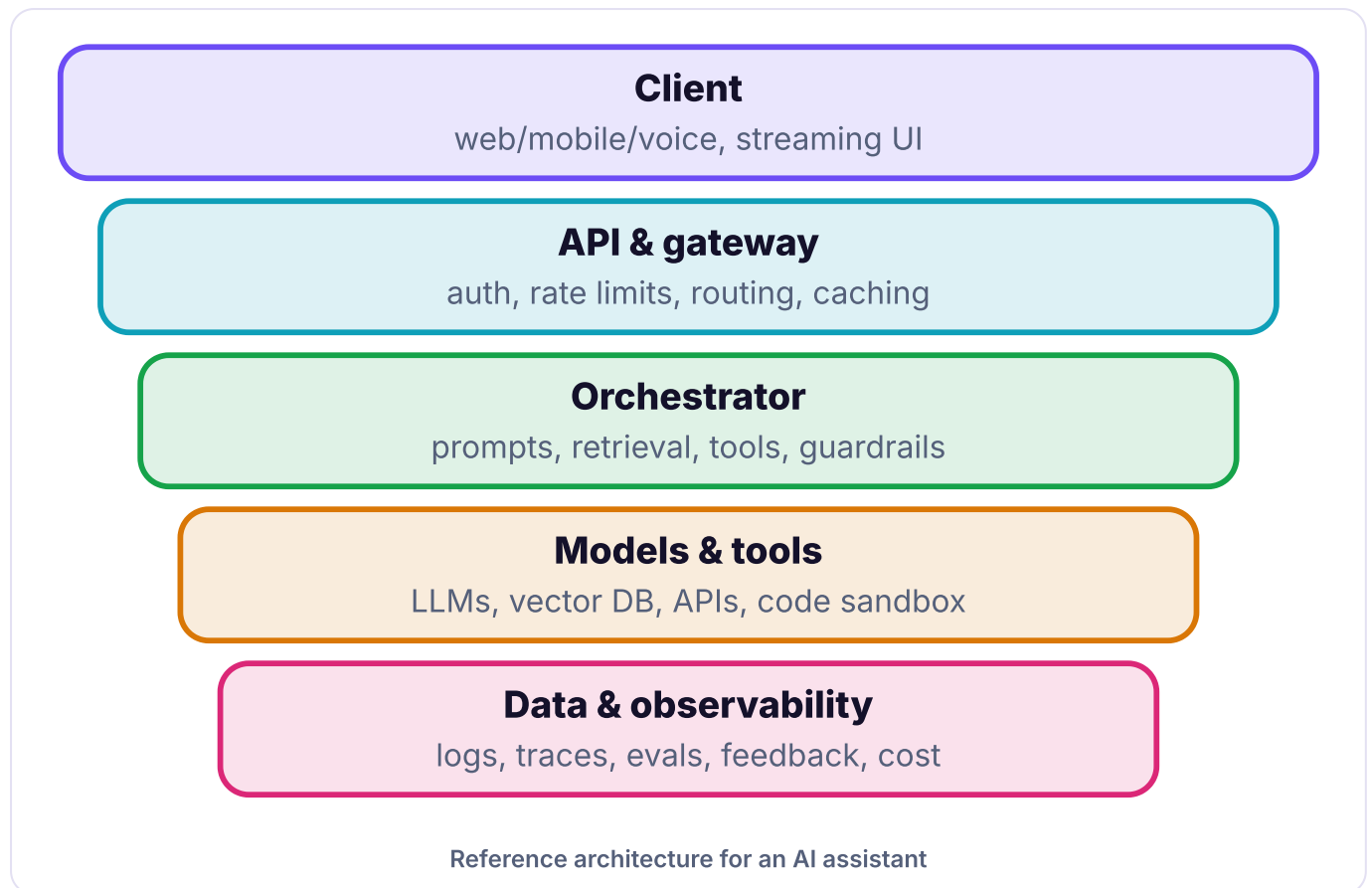
Key takeaways

- Good data pipelines keep RAG fresh and safe.
- Know the OWASP LLM risks and their controls.
- Least privilege for every agent tool.

Practice exercise: Write a threat model for an AI support agent that can issue refunds.

System design for AI products and working with the team

Design end-to-end AI systems and collaborate across roles.



How to answer an AI system-design question

1. Clarify users, scale, languages, latency and quality targets, risks.
2. Sketch the architecture (above) and the data flow.
3. Pick approaches with trade-offs: prompting vs RAG vs fine-tune; one model vs routing; sync vs async.
4. Cover evals, safety, observability, cost and failure handling.
5. Discuss rollout: shadow, canary, A/B, human fallback.

Common designs to practise

- Customer-support copilot (RAG + tools + handoff)
- Document processing pipeline (OCR → extraction → validation → human review)
- Voice agent for appointments (STT → LLM → TTS, barge-in, latency)
- Code assistant (indexing, context selection, sandbox, tests)
- Internal knowledge search with permissions
- Recommendation or search ranking with LLM reranking

Working with PMs, designers and data teams

- Turn PRDs into eval sets and acceptance criteria; push back with data on feasibility and cost.
- Demo early with real outputs including failures; document assumptions and limits.
- Be on-call for AI quality: incident reviews, postmortems, new tests.

Learn it here: [AI Product Strategy](#) · [Designing AI Experiences](#) · [Platform & API PM](#)

Key takeaways

- Always state trade-offs and failure handling.
- Evals, safety and cost belong in every design.
- Great AI engineers translate between product and models.

Practice exercise: Design a document-processing AI system for insurance claims and present it in 10 minutes.

Career: roadmap, portfolio projects, interviews and job descriptions

Plan 12 months, build proof, and prepare for hiring loops.

12-month roadmap

Months	Focus	Proof
1-2	Python, Git, SQL, HTTP/APIs, testing	3 small tools on GitHub
3-4	ML basics, statistics, scikit-learn, one Kaggle project	Notebook with metrics and error analysis
5-6	LLM APIs, prompts, structured output, RAG	RAG app with citations and evals
7-8	Agents, tools, MCP, workflows	Agent with tracing and approvals
9-10	Evals, deployment, Docker, cloud, monitoring	Deployed app with CI and dashboards
11-12	Fine-tuning, system design, interviews, applications	Capstone + case studies + mock interviews

10 portfolio projects

1. Chat with your documents (RAG, citations)
2. Support copilot with ticket tools
3. Text-to-SQL analyst with guardrails
4. Resume/job matcher with embeddings
5. Voice appointment bot
6. Browser research agent with Playwright MCP
7. Document extraction pipeline (OCR → JSON → validation)
8. LoRA fine-tuned classifier vs prompting baseline
9. LLM eval harness with CI
10. Cost-aware model router with a dashboard

Interview loop

- **Coding:** Python, data structures, API handling; (see DSA lessons).
- **ML:** metrics, overfitting, validation, embeddings, transformers.
- **LLM:** RAG design, prompt/context, evals, hallucination, cost/latency, guardrails.
- **System design:** one AI architecture end to end.
- **Behavioural:** ownership, incidents, collaboration with product.

What job descriptions usually ask for

Python; LLM APIs; RAG and vector databases; agents/tool use; evaluation and testing; cloud and containers; CI/CD; SQL and data pipelines; security and responsible AI; strong communication. (Read real postings and tick off what you can already show.)

Learn it here: [Data Scientist, ML and GenAI interview preparation](#) · [Portfolio, GitHub and resume](#) · [Your 90-day plan](#) · [DSA I](#) · [Top GenAI questions](#) · [Practice Lab](#) · [Job board](#)

Key takeaways

- Build proof, not just courses: 10 projects.
- Prepare coding, ML, LLM, system design and behavioural rounds.
- Match your portfolio to real job descriptions.

Practice exercise: Pick 3 projects, build them over the next 8 weeks, and publish each with a README, demo and eval results.

Templates

A. AI project one-pager

Field	Fill in
Problem & user	
Success metric & eval set	
Approach (prompt / RAG / agent / fine-tune)	
Data sources & access control	
Architecture sketch	
Risks & guardrails	
Cost & latency budget	
Deployment & monitoring	

B. Production readiness checklist

☐ Evals pass in CI ☐ Tracing on ☐ Secrets in env/secret manager ☐ Rate limits & budgets
☐ Prompt/model versions pinned ☐ Fallback & timeouts ☐ PII handling ☐ Rollback plan ☐ On-call owner

C. 12-month tracker

month | focus | project | proof link | done

AI engineer tool stack

Term / tool	Meaning / use
Languages & libs	Python, TypeScript; requests/httpx, pydantic, FastAPI, pytest, pandas, NumPy, scikit-learn, PyTorch
Model APIs	Anthropic Claude, OpenAI, Google Gemini; open models via Hugging Face, Ollama, vLLM, Groq, Together
Frameworks	LangGraph, LangChain, LlamaIndex, OpenAI Agents SDK, Claude Agent SDK, CrewAI, Pydantic AI
Vector stores	pgvector, Pinecone, Qdrant, Weaviate, Milvus, Chroma, FAISS, OpenSearch
Evals & tracing	promptfoo, Ragas, DeepEval, Inspect, Langfuse, LangSmith, Arize Phoenix, Braintrust
Fine-tuning	Hugging Face Transformers, PEFT, TRL, Unsloth, Axolotl; provider fine-tuning APIs
Serving & gateways	FastAPI, vLLM, TGI, Triton, Ollama, LiteLLM, OpenRouter, Cloudflare AI Gateway
Infra	Docker, Kubernetes, AWS/GCP/Azure, Cloudflare, GitHub Actions
Data	Postgres, DuckDB, Airflow, Prefect, dbt, Label Studio
Safety	Guardrails AI, NeMo Guardrails, Llama Guard, OWASP LLM Top 10
Automation	n8n, Zapier, Make, Playwright, Playwright MCP

Glossary

LLMOps	Operating LLM apps in production: deployment, monitoring, evals, cost
Embedding	Vector capturing the meaning of text
Reranker	Model that reorders retrieved results by relevance
LoRA	Parameter-efficient fine-tuning method
Quantisation	Reducing model precision to run cheaper/faster
Gateway	Layer for routing, auth, caching and budgets across model providers
Tracing	Recording each step of a request for debugging

12 interview questions with answers

Q1. What does an AI engineer do?

Builds, evaluates, deploys and operates AI-powered software: model APIs, RAG, agents and tools, evals, guardrails, cloud deployment and monitoring, with attention to cost, latency and safety.

Q2. AI engineer vs ML engineer?

AI engineers mostly build products on existing models (prompts, RAG, agents, evals); ML engineers train and serve custom models and run ML pipelines. The roles overlap.

Q3. How would you design a RAG system?

Ingest and chunk documents with metadata, embed, store in a vector DB, retrieve with hybrid search and reranking, generate with citations, enforce access control, evaluate retrieval and faithfulness, monitor and refresh.

Q4. When do you fine-tune?

After prompting and RAG fall short: for consistent style or format, narrow tasks, or to lower cost with a smaller model; compare to a prompting baseline on held-out data.

Q5. How do you evaluate an LLM application?

Golden dataset with slices, code and LLM-judge graders calibrated to humans, regression tests in CI, online metrics and sampled judging.

Q6. How do you reduce hallucinations?

Ground answers in retrieved sources, require citations and 'not found', use structured outputs, add verification steps and evals, and escalate low-confidence cases.

Q7. How do you control cost and latency?

Route to smaller models, cache, trim context, stream, batch, set budgets, and track cost per successful task.

Q8. What are the main security risks of LLM apps?

Prompt injection, data leakage, insecure output handling, excessive agency, supply chain and denial of wallet; mitigate with least privilege, validation, sandboxing, approvals and rate limits.

Q9. Explain MCP.

The Model Context Protocol standardises how AI assistants connect to tools and data; you can build MCP servers to expose your product and use existing ones.

Q10. Walk through deploying an AI service.

FastAPI service in Docker, CI with tests and evals, deploy to cloud, gateway for auth/limits/routing, tracing and alerts, canary rollout and rollback.

Q11. How do you handle a quality regression in production?

Check traces, compare against the eval set, identify cause (prompt, retrieval, model update, data drift), roll back, fix, add a regression test.

Q12. What projects would you show in a portfolio?

A cited RAG app, an agent with tracing, an eval harness in CI, a deployed service with monitoring, and a fine-tune vs prompt comparison.

Further reading and tools

- Hugging Face LLM course: <https://huggingface.co/learn/llm-course>
- DeepLearning.AI short courses: <https://www.deeplearning.ai/short-courses/>
- LLM Zoomcamp (DataTalks.Club): <https://github.com/DataTalksClub/llm-zoomcamp>
- MLOps Zoomcamp: <https://github.com/DataTalksClub/mlops-zoomcamp>
- fast.ai Practical Deep Learning: <https://course.fast.ai/>
- Andrej Karpathy: Neural Networks Zero to Hero: <https://karpathy.ai/zero-to-hero.html>
- Anthropic: Building effective agents: <https://www.anthropic.com/research/building-effective-agents>
- OpenAI Cookbook: <https://cookbook.openai.com/>
- LangChain Academy: <https://academy.langchain.com/>
- OWASP Top 10 for LLM apps: <https://genai.owasp.org/>
- Chip Huyen: AI Engineering (book site): <https://huyenchip.com/books/>
- Full Stack Deep Learning: <https://fullstackdeeplearning.com/>
- Eugene Yan: patterns for LLM systems: <https://eugeneyan.com/writing/llm-patterns/>

© SkillNavora · skillnavora.in · Part of the AI Product Management and GenAI Launchpad programs. Practise with the free AI Tutor, AI PM Toolkit and Practice Lab on skillnavora.in.