LEARNING PATHWAY

From zero to agentic AI engineer

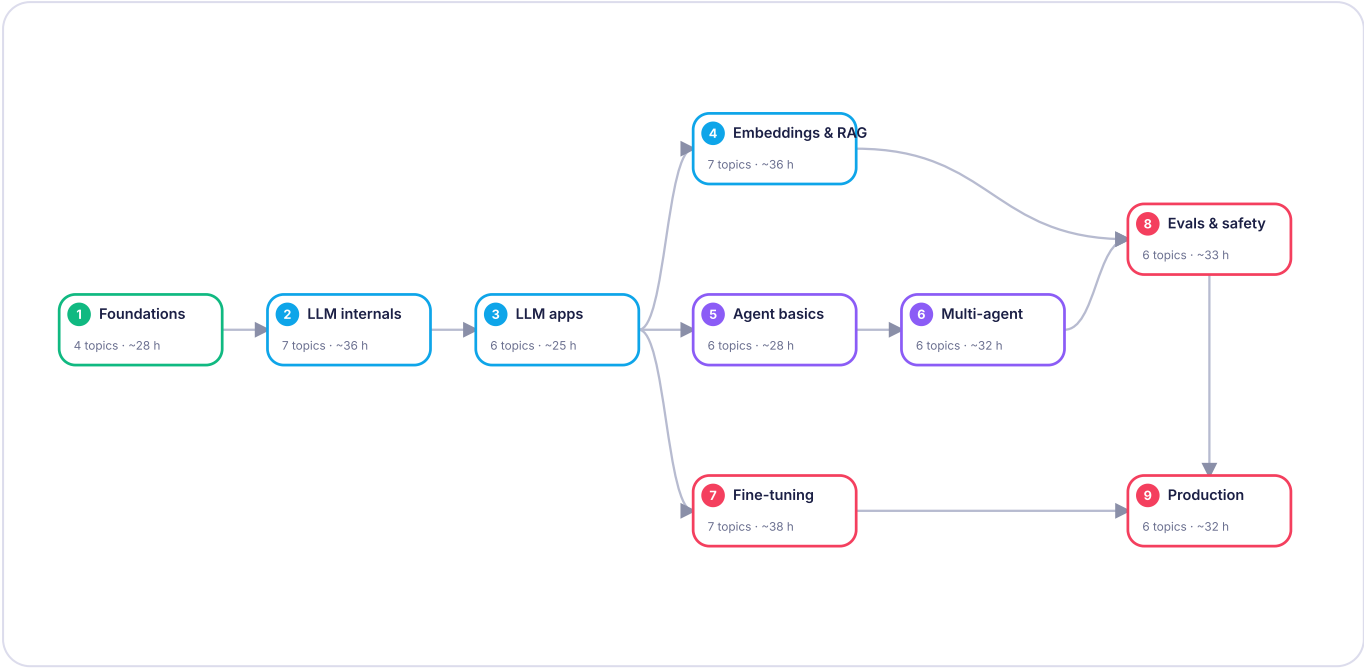
A code-first roadmap for engineers: from Python and backpropagation to transformers, RAG, agents, fine-tuning, evaluation and production. Every topic has working code, curated papers and videos, and a checkpoint.

9 STAGES

55 TOPICS

~288 h OF MATERIAL

Beginner · Intermediate · Advanced · Expert



Arrows point from a stage to the stages that build on it. Online: biditsadhukhan.in/learn

Start here: your first agent in seven days

About 3 to 4 focused hours a day. Assumes you can code but have not built with LLMs before.

1

See inside the machine · 3 to 4 h

Know what a token is and how attention moves information between positions.

Read: Tokenization and byte-pair encoding · Embeddings and self-attention

Build: Train the 25-line BPE on any text you have, then write causal attention from memory and check it against PyTorch's fused version.

Check: You can explain why attention costs $O(T^2)$ and why we divide by $\sqrt{d}$.

2

Assemble a language model · 3 to 4 h

Put the pieces together and watch a model learn.

Read: The transformer block and the full GPT · Decoding and sampling

Build: Stack the blocks into a small GPT, train it for a few minutes on a small corpus, and sample at three temperatures.

Check: Validation loss beats a unigram baseline, and you can describe what temperature does to the distribution.

3

Call models like an engineer · 3 h

Reliable API calls and output you can parse.

Read: Calling LLM APIs robustly · Structured outputs and validation

Build: Write an async runner with retries and token accounting, then extract fields from 20 messy documents into a Pydantic model with a repair loop.

Check: A validation error is fed back to the model, and you log tokens and cost for every call.

4

Give the model hands · 4 h

Tool calling, then your own agent loop.

Read: Function calling: giving the model tools · Build the agent loop from scratch (ReAct)

Build: Build the framework-free agent with a calculator and a stubbed search tool, a step budget and a JSONL trace.

Check: The agent stops on its budget, returns tool errors to the model, and you can replay a failed run from its trace.

5

Answer from your own data · 4 h

A RAG pipeline that cites its sources and can say 'I don't know'.

Read: Embedding models and similarity · A complete RAG pipeline with citations

Build: Index a small corpus, retrieve with embeddings, answer with numbered citations, and test 10 questions it cannot answer.

Check: Every citation points at a retrieved chunk, and unanswerable questions are refused.

6

Measure it, then attack it · 4 h

An eval gate and a first round of security tests.

Read: Eval-driven development · Guardrails and prompt injection

Build: Write a 30-item golden set for your RAG or agent project and a script that reports accuracy with a confidence interval. Add 10 prompt-injection tests that assert on tool calls.

Check: A deliberately worse prompt fails your gate, and at least one injection test taught you something.

7

Ship one thing and plan the rest · 2 to 3 h

Turn the week into a small portfolio piece and a longer plan.

Read: What is an agent? Workflows versus agents · Staying current: reading papers, a paper trail and a study plan

Build: Write the README for your project (what it does, how to run it, how it was evaluated, its limits). Then choose your next stage and book the hours.

Check: A stranger could run your repository from the README, and you know which stage comes next.

The full pathway

Tick each topic off as you complete it. Folder order is learning order.

Beginner Stage 1 · 4 topics · about 28 hours

Foundations: Python, maths and neural nets

Generative AI is gradient descent on very large networks. This stage gets you fluent in the Python tooling, the three areas of maths you will use every day (linear algebra, calculus and probability), and the training loop that every later stage builds on.

By the end: You can write a small neural network and its training loop in PyTorch, explain backpropagation, and read a loss curve.

- ☐ **Python and tooling for ML work** · Beginner · 4 h
  A reproducible environment, typed async Python, notebooks that don't rot, and GPUs you can rent by the hour.
- ☐ **The maths you actually use** · Beginner · 8 h
  Vectors and matrices, the chain rule, and probability with cross-entropy: just enough to read any LLM paper.
- ☐ **Machine learning essentials** · Beginner · 6 h
  Loss, optimisation, generalisation and honest evaluation, before neural networks make everything bigger.
- ☐ **Neural networks and PyTorch** · Beginner · 10 h
  Tensors, autograd, nn.Module and the training loop, built up from a hand-written autograd engine.

Capstone: Train a character-level model from scratch. Implement a bigram, then an MLP character-level language model that generates names, first by hand with explicit gradients and then in PyTorch. Track train and validation loss and write up what overfitting looked like.

Intermediate Stage 2 · 7 topics · about 36 hours

LLM internals: how transformers work

You will implement byte-pair encoding, causal self-attention and a full GPT, train it on a small corpus, and learn how scale, data and decoding choices change what a model does. Everything here is deliberately small enough to run on a laptop or a free GPU.

By the end: You can explain a decoder-only transformer end to end, implement one in about 150 lines, and estimate the memory and compute a model needs.

- ☐ **Tokenization and byte-pair encoding** · Intermediate · 4 h
  Why models see sub-words, not characters or words, and how many odd behaviours trace back to it.
- ☐ **Embeddings and self-attention** · Intermediate · 6 h
  Queries, keys and values, the causal mask, and multi-head attention, derived and implemented.
- ☐ **The transformer block and the full GPT** · Intermediate · 4 h
  Residual streams, layer norm, the MLP, and stacking blocks into a decoder-only language model.
- ☐ **Build and train a GPT from scratch** · Intermediate · 8 h
  Data loading, the training loop, evaluation and generation: a working language model on your own text.

- [ ] **Pre-training, data and scaling laws** · Advanced · 5 h
  Where capability comes from: data, compute and parameters, the Chinchilla result, and how to budget a run.
- [ ] **Decoding and sampling** · Intermediate · 3 h
  Greedy, temperature, top-k, top-p and beam search, plus logprobs, and how to choose settings for a task.
- [ ] **Modern architecture upgrades: RoPE, GQA, FlashAttention, MoE** · Advanced · 6 h
  What changed between GPT-2 and today's open models, and the KV-cache arithmetic that drives serving cost.

Capstone: Your own GPT, trained and sampled. Train a small GPT on a text corpus you care about (code, a language, a book collection). Add a KV cache to your sampler and compare perplexity as you vary model width and context length.

Intermediate Stage 3 · 6 topics · about 25 hours

Building with LLMs: APIs, prompts and structured output

This is the applied core. You will call hosted and local models robustly, write prompts as testable artefacts, force machine-readable output, give the model tools, and manage context and cost. Every pattern here reappears inside agents and RAG.

By the end: You can build a reliable single-call LLM feature: validated structured output, retries, streaming, tool use, cost tracking, and a prompt you can regression-test.

- [ ] **Calling LLM APIs robustly** · Intermediate · 4 h
  Messages, streaming, async fan-out, retries and rate limits, token accounting, across Anthropic, OpenAI and Azure.
- [ ] **Prompt engineering as an engineering discipline** · Intermediate · 5 h
  Roles, structure, examples, reasoning, and above all prompts you can test and version.
- [ ] **Structured outputs and validation** · Intermediate · 4 h
  Get typed, validated JSON from a model using schemas, forced tool calls and a validate-and-retry loop.
- [ ] **Function calling: giving the model tools** · Intermediate · 4 h
  The tool-call loop, parallel calls, error handling and the safety rules that apply the moment a model can act.
- [ ] **Context windows, caching and cost control** · Intermediate · 4 h
  Budgeting tokens, where long-context models fail, prompt caching, and routing work to the right model.
- [ ] **Running open models locally with Hugging Face** · Intermediate · 4 h
  Load, prompt and quantise open-weight models with Transformers, and serve them with Ollama or llama.cpp.

Capstone: A typed extraction service. Build an API that turns messy documents (invoices, emails or support tickets) into validated JSON. Use structured output with a validation-and-retry loop, cache the long instructions, stream where it helps, and log tokens and cost per request.

Intermediate Stage 4 · 7 topics · about 36 hours

Embeddings and retrieval-augmented generation

RAG is a search problem wearing a language-model costume. Most failures are retrieval failures, so this stage spends its time on how text becomes vectors, how to index and search them well, and how to measure retrieval and generation separately.

By the end: You can build, evaluate and debug a RAG system, and tell whether a bad answer came from ingestion, retrieval or generation.

- [ ] **Embedding models and similarity** · Intermediate · 4 h
  How text becomes vectors, what cosine similarity means, how to choose a model, and where embeddings mislead.
- [ ] **Parsing, chunking and metadata** · Intermediate · 5 h
  Turning messy documents into retrievable units: parsing PDFs, chunk size and overlap, structure-aware splits and filters.
- [ ] **Vector search: ANN, FAISS and pgvector** · Intermediate · 5 h
  Exact versus approximate nearest-neighbour search, HNSW and IVF, filtering, and when a database beats a library.

- [ ] **Hybrid search and reranking** · Advanced · 5 h
  Combine keyword and vector retrieval, fuse the rankings, and rescore the top results with a cross-encoder.
- [ ] **A complete RAG pipeline with citations** · Intermediate · 6 h
  Retrieve, build a grounded prompt, generate with citations, and refuse when the context does not contain the answer.
- [ ] **Advanced RAG: query transformation, graph, agentic retrieval** · Advanced · 6 h
  Rewrite queries, hypothetical documents, multi-hop, self-correcting retrieval, GraphRAG, and when long context beats retrieval.
- [ ] **Evaluating RAG: retrieval and generation separately** · Advanced · 5 h
  Recall@k, MRR, context precision, faithfulness and answer relevance, and how to build a gold set quickly.

Capstone: A cited question-answering system with its own eval. Index a real corpus (a documentation site, a set of papers, policy PDFs). Implement hybrid retrieval with reranking, answer with citations, and build a 50-question evaluation set that scores retrieval and faithfulness separately.

Advanced Stage 5 · 6 topics · about 28 hours

Agent fundamentals: loops, tools, memory and planning

Before any framework, you will write the agent loop yourself, design tools the model can use well, give it memory, add planning and reflection, and connect it to the outside world with the Model Context Protocol. The recurring theme: constrain autonomy to what you can verify.

By the end: You can build a tool-using agent with budgets, tracing and memory, and say honestly when a plain workflow would be better.

- [ ] **What is an agent? Workflows versus agents** · Intermediate · 3 h
  The five workflow patterns, where true agents begin, and how to decide which one a problem deserves.
- [ ] **Build the agent loop from scratch (ReAct)** · Advanced · 6 h
  Reason, act, observe: a complete tool-using agent with budgets, a schema generator, tracing and failure handling.
- [ ] **Designing tools for agents** · Advanced · 4 h
  The agent-computer interface: names, descriptions, return values, errors, idempotency and sandboxing.
- [ ] **Agent memory: short-term, long-term and episodic** · Advanced · 5 h
  Context management, running summaries, vector-backed memory, and the risks of remembering the wrong things.
- [ ] **Planning, reflection and search over reasoning** · Advanced · 5 h
  Plan-and-execute, replanning, self-critique loops, and tree search, with a clear view of when each pays off.
- [ ] **Model Context Protocol (MCP)** · Advanced · 5 h
  A standard for connecting models to tools and data: servers, clients, transports, and the security model.

Capstone: A research agent with a hard budget. Build an agent that answers multi-step questions using a search tool, a page-reader and a calculator. Give it a step and cost budget, a scratchpad, structured traces, and a final answer that cites what it read. Compare it with a fixed workflow on the same 20 questions.

Advanced Stage 6 · 6 topics · about 32 hours

Agent frameworks, multi-agent systems and durable execution

With the loop understood, frameworks become legible. You will compare the main options, build stateful graphs with checkpoints and human approval, design multi-agent systems with bounded hand-offs, and apply the patterns to coding agents and agents over structured data.

By the end: You can pick an orchestration approach for a problem, build a resumable multi-step agent with human-in-the-loop approval, and explain the failure modes of multi-agent designs.

- [ ] **Choosing an agent framework** · Advanced · 3 h
  Raw SDK, LangGraph, OpenAI Agents SDK, smolagents, AutoGen, Semantic Kernel, CrewAI and managed services: what each is for.

- [ ] **Stateful agents with LangGraph** · Advanced · 6 h
  Graphs, typed state, reducers, conditional edges, checkpointers and threads, with a ReAct agent and a custom workflow.
- [ ] **Multi-agent patterns and their failure modes** · Expert · 7 h
  Router, supervisor-worker, parallel DAG, debate and evaluator, with context hand-off discipline and honest advice on when not to.
- [ ] **Human-in-the-loop and durable execution** · Expert · 5 h
  Interrupts for approval, checkpointing so runs survive crashes, idempotency, retries and queues for long jobs.
- [ ] **Coding agents and computer-use agents** · Expert · 5 h
  Repository agents, sandboxes, test-driven loops, browser and desktop control, and how they are evaluated.
- [ ] **Agents over data: agentic RAG and natural-language-to-SQL** · Expert · 6 h
  Letting an agent query structured data safely: schema grounding, read-only validation, row caps, self-repair and confidence checks.

Capstone: A supervised multi-step agent on real data. Build an agent that answers analytical questions over a database through a router, specialist workers and a synthesiser. Make it resumable with a checkpointer, add a human approval step for any expensive or sensitive action, and trace every run.

Expert Stage 7 · 7 topics · about 38 hours

Fine-tuning and alignment: SFT, LoRA, DPO and RL for reasoning

Fine-tuning changes how a model behaves, not what it reliably knows. You will decide when it is the right tool, run supervised fine-tuning, use LoRA and QLoRA to do it on one GPU, apply preference optimisation and reinforcement learning with verifiable rewards, then quantise and distil for deployment.

By the end: You can decide between prompting, RAG and fine-tuning with evidence, fine-tune an open model with LoRA, align it with DPO, and evaluate whether it actually improved.

- [ ] **When to fine-tune (and when not to)** · Advanced · 3 h
  A decision process across prompting, RAG and fine-tuning, with the costs and risks of each.
- [ ] **Supervised fine-tuning (SFT) with TRL** · Advanced · 6 h
  Chat-format data, loss masking, hyper-parameters and a complete training run with Hugging Face TRL.
- [ ] **LoRA and QLoRA: parameter-efficient fine-tuning** · Expert · 6 h
  Low-rank adapters, their maths, PEFT code, 4-bit QLoRA on a single GPU, and merging adapters for deployment.
- [ ] **Preference optimisation: RLHF, DPO and friends** · Expert · 6 h
  How human or AI preferences shape a model: reward models, PPO, and the simpler DPO objective with code.
- [ ] **Reinforcement learning with verifiable rewards (GRPO)** · Expert · 7 h
  How reasoning models are trained: sampled groups, programmatic rewards, group-relative advantages and what to watch for.
- [ ] **Quantisation and distillation** · Advanced · 5 h
  Shrink models for deployment: int8 and 4-bit quantisation, GPTQ and AWQ, GGUF, and distilling a big model into a small one.
- [ ] **Data curation and synthetic data** · Advanced · 5 h
  Quality filters, deduplication, decontamination, and generating synthetic training data without poisoning your own model.

Capstone: Fine-tune a small model and prove it beat the baseline. Pick a narrow task (structured extraction, a house style, a domain classifier). Establish a prompted baseline, build a clean dataset, fine-tune a 1B to 8B model with LoRA, optionally apply DPO, then compare against the baseline on a held-out set and report cost per correct answer.

Expert Stage 8 · 6 topics · about 33 hours

Evaluation, observability and safety

LLM systems fail quietly: the output is fluent and wrong. You will build evaluation sets and regression gates, use LLM judges without fooling yourself, evaluate agents on trajectories, trace every run, and defend against prompt injection and data leakage with layered controls and a governance routine.

By the end: You can put an eval gate in CI, debug any production run from its trace, and describe the threat model of your system with mitigations that do not depend on the model behaving.

- ☐ **Eval-driven development** · Advanced · 6 h
  Golden sets, task-specific metrics, confidence intervals and a CI gate that blocks regressions.
- ☐ **LLM-as-judge without fooling yourself** · Expert · 5 h
  Rubrics, pairwise comparison, position and length bias, calibration against humans, and when not to use a judge.
- ☐ **Evaluating agents: trajectories, pass^k and cost** · Expert · 5 h
  Outcome and process metrics, repeated trials, environment-based graders and the benchmarks worth knowing.
- ☐ **Observability: tracing LLM and agent runs** · Advanced · 5 h
  Spans for model and tool calls, token and cost attributes, structured logs, redaction and debugging from a trace.
- ☐ **Guardrails and prompt injection** · Expert · 7 h
  The lethal trifecta, direct and indirect injection, layered defences, PII and output controls, and designing so a fooled model cannot do damage.
- ☐ **Red-teaming, threat modelling and governance** · Expert · 5 h
  Structured adversarial testing, a threat model for LLM systems, and the policies, records and oversight that keep a deployment defensible.

Capstone: Harden and instrument one of your earlier projects. Take your RAG or agent capstone. Add a golden evaluation set with a CI gate, OpenTelemetry tracing with token and cost attributes, a layered guardrail gateway, and a red-team test suite of at least 30 attacks. Write the threat model and a one-page risk register.

Expert Stage 9 · 6 topics · about 32 hours

Production and scale: serving, batch, cost and operations

Shipping an LLM feature is mostly systems engineering. You will understand what limits inference throughput, design resumable batch pipelines for very large jobs, engineer cost and latency deliberately, release prompts and models with the same discipline as code, and assemble everything into a reference architecture.

By the end: You can size and serve a model, build a fault-tolerant batch pipeline, budget cost per task, and run releases with canaries, evals and rollbacks.

- ☐ **Serving models: KV cache, batching and vLLM** · Expert · 7 h
  Prefill versus decode, continuous batching, PagedAttention, speculative decoding, and how to size and benchmark an inference server.
- ☐ **Batch LLM pipelines: chunking, checkpoints and cost** · Expert · 7 h
  Processing 100,000+ items with provider batch APIs: payload design, size-aware chunking, resumable state, error retries and validation.
- ☐ **Cost and latency engineering** · Advanced · 5 h
  Unit economics, caching, model cascades, shorter outputs, parallelism and the budgets and alerts that keep spend predictable.
- ☐ **LLMOps: releases, canaries and rollback** · Expert · 5 h
  Version prompts, models and data together; gate releases with evals; deploy gradually; and handle model deprecations.
- ☐ **A reference architecture for GenAI systems** · Expert · 4 h
  How the pieces fit: gateway, orchestrator, retrieval, tools, guardrails, observability and the evaluation loop, with the trade-offs to record.
- ☐ **Staying current: reading papers, a paper trail and a study plan** · Advanced · 4 h
  How to read research efficiently, the papers that explain today's systems, where to follow the field, and a 12-week plan.

Capstone: Take one system to production-grade. Choose your best earlier project. Serve any self-hosted model behind an OpenAI-compatible API with vLLM, put a gateway in front, add caching and a model cascade, set a per-task cost budget with alerts, build a resumable batch job for bulk work, and ship changes through a canary with an eval gate and a rollback.

Written by Bidit Sadhukhan. Every topic page, with code, diagrams and links, is at biditsadhukhan.in/learn.