

Choose your starting point.

A practical workbook for building useful AI applications and preparing for a focused job search.

My recommendation is to begin with applied AI engineering: ordinary software skills plus models, retrieval, evaluation and reliable delivery. Keep one useful project at the center of your learning.

Starting point	First priority	Planning allowance
Brand new	Python, SQL, Git and small tested programs	240-360 focused hours
Experienced engineer	Model behavior, evaluations and AI delivery	120-180 focused hours
Vibe coder	Read, debug, test and operate the apps you generate	160-240 focused hours

These are my study estimates, including practice and a first portfolio cycle. At 10 hours/week, divide the hours by 10. They are not hiring timelines or guarantees. Adjust using the skill checkpoints.

The shared sequence

- Build one model-powered feature with a baseline.
- Add retrieval only when your task needs document evidence.
- Measure quality, inspect failures and keep a held-out test set.
- Deploy a bounded application with useful failure behavior.
- Write a case study, demonstrate your work and practice interviews.

My starting route and why:

My target role / industry:

The three original BASWE.AI classes remain in the library. This workbook is independent, original guidance; the online section contains the full curriculum and source notes.

A week you can actually finish.

Use this page each week. Pick one outcome small enough to demonstrate.

Hours available / calendar blocks:

The one outcome I will finish:

What I need to learn to build it:

What I will build or change:

How I will test it:

Who will try it or give feedback:

Review at the end of the week

What worked? What failed?

What evidence changed my mind?

The smallest useful next step:

Check your foundations.

Use documentation. First try without generated solutions so the gaps are visible.

Area	Demonstrate it	Score
Python	Read JSON, validate records, transform data and handle an error.	0 / 1 / 2
Testing	Write edge-case tests; introduce a bug and catch it.	0 / 1 / 2
HTTP	Validate a request body and return useful responses.	0 / 1 / 2
SQL	Create related tables, join them and aggregate results.	0 / 1 / 2
Git	Make a branch, review a diff, resolve a conflict and revert.	0 / 1 / 2
Operations	Run from a clean environment; explain config and secrets.	0 / 1 / 2
Explanation	Trace a request through the app and identify failure points.	0 / 1 / 2

0 = cannot do it yet. 1 = can do it with help. 2 = can do it and explain it. This is a study diagnostic, not a hiring score.

Choose the bridge you need

Programming gaps: start the beginner route. Comfortable coding but weak testing or operations: use the engineering bridge. Mostly independent: begin the model feature and fill gaps as they appear.

My two weakest areas:

The exercise that will demonstrate improvement:

Link to code / evidence:

Define one useful project.

Default idea: a document assistant for a narrow user group, with traceable answers and a measured search baseline.

Specific user and recurring task:

How the user does it today:

Evidence from conversations / assumptions to validate:

Why AI may help; why simpler search or rules might suffice:

Data source and permission to use it:

One supported workflow and three exclusions:

Baseline and success criterion:

Budget, latency target and unacceptable failure:

Milestones: baseline / AI feature / evaluation / deployment / case study:

Measure before you optimize.

Create a repeatable evaluation set. Keep final test cases out of prompt tuning.

Suggested practice size: 20 development cases and 10 held-out cases, growing toward 50-100 diverse cases. These are learning targets, not production-validation standards. The included synthetic JSONL is only a tiny starting example.

Case type	Expected behavior
Ordinary task	Correct answer or action with the required structure.
Missing evidence	Clear abstention or request for clarification.
Ambiguous input	Useful clarification instead of an invented assumption.
Malformed input	Validation error with recoverable behavior.
Adversarial document	Treat embedded instructions as data; preserve boundaries.

Record every run

Code commit, model identifier, prompt version, corpus/index version, retrieval settings, trial count and date. Do not pass expected answers or labels to the app as inputs.

Measure	Baseline	New version
Task success: passed / total		
Supported answers: passed / total		
Correct abstentions: passed / total		
Median / p95 latency; sample size		
Estimated cost per successful task		
Critical permission / tool failures		

Use code for objective checks and human review for support and usefulness. Calibrate any model grader. Report failures and denominators; one average can hide an important problem.

Explain a failure.

Use one page per meaningful failure. This is often the best material for your project interview.

Case ID / input:

Expected behavior:

Observed behavior and relevant output:

Failure layer: ingestion / retrieval / model / validation / tool / product:

What evidence supports my hypothesis?

The smallest experiment to distinguish causes:

What changed and what stayed fixed?

Result, including any regression:

The new test and remaining limitation:

Ship it with a clear boundary.

Before a second person uses the app, make its setup and failure behavior understandable.

- [] Clean setup works from the README; dependencies are recorded.
- [] Secrets stay server-side and out of source control.
- [] Inputs and outputs are validated.
- [] Private document permissions are checked before retrieval.
- [] Logs identify failures without unnecessary sensitive text.
- [] Timeouts, retries, output size and tool steps are bounded.
- [] Rate limits and cost controls fit the demo.
- [] Missing evidence and provider outages have a useful UI.
- [] Corpus updates, retention and deletion are explained.
- [] Evaluation results, limitations and rollback are documented.

Run a failure drill

Break the provider connection, empty the retrieval index and submit invalid input in separate trials. Record the outcome, restore normal settings and confirm recovery.

What failed gracefully? What needs work?

Who tried the app, and what could they complete?

Three-minute demo

User and task (30s) -> working flow (60s) -> evidence (30s) -> failure (30s) -> tradeoff and next improvement (30s).

Turn the project into evidence.

A portfolio should help someone assess what you did, how it works and what you learned.

Case study outline

- User and problem, including whether needs were validated.
- Baseline, architecture and data provenance.
- Actual measurements with the dataset, sample size and conditions.
- One failure, your investigation and a meaningful tradeoff.
- Your ownership, reviewed AI assistance and the next improvement.
- Links to a runnable repository, demo and evaluation instructions.

Resume bullet template

Built [feature] for [user/task] using [relevant implementation]; evaluated on [dataset and sample size], achieving [actual result] versus [baseline], with [latency/cost/reliability context].

Keep placeholders until you have measurements. Label synthetic data and small user tests. Do not invent customers, production scale, revenue or time savings.

My project in one sentence:

The best measured result, with denominator:

A tradeoff I can defend:

The job requirement this demonstrates:

Practice the hiring conversation.

Use the actual role requirements to choose your practice. Ask the recruiter about format and AI-tool rules.

Practice	Task
Coding / 45 minutes	Parse, transform, validate and test a small dataset. Explain edge cases.
Debugging / 45 minutes	Find why a retrieved answer cites the wrong passage. Isolate the layer.
Design / 45 minutes	Design an internal assistant with permissions, a budget and failure recovery.
Project / 30 minutes	Explain the baseline, show a failure and defend one architecture decision.

Questions to rehearse

- Why retrieval rather than fine-tuning or a larger prompt?
- What is held out, and how representative is the test set?
- How do you enforce access and handle malicious document content?
- What happens when the model changes or the provider fails?
- What did you personally decide, build, test and review?

A sustainable weekly search

As a starting cadence, try 3-5 tailored applications, two thoughtful conversations and one practice session per week. Adjust to your time and available roles. Track screens and technical rounds so you can identify the bottleneck.

My weakest interview area and next exercise:

One relevant team and the evidence I can show them:

Keep the study list small.

Open these official pages for the learning material. The complete annotated source list is in guide 15.

CS50: Introduction to Programming with Python

Free OpenCourseWare; certificate options are separate.

CS50P: Unit Tests

Official lecture, notes and exercises.

CS50: Introduction to Databases with SQL

Free OpenCourseWare; use selected units first.

GitHub: Hello World

Free guide to the repository and pull request workflow.

FastAPI: Tutorial

Free, maintained documentation.

Full Stack LLM Bootcamp: LLM Foundations

Free video; 2023 conceptual material. Model examples and APIs are dated.

Full Stack LLM Bootcamp: Augmented Language Models

Free video; 2023 conceptual material. Use current documentation for implementation.

Full Stack LLM Bootcamp: LLMOps

Free video; 2023 lifecycle concepts, not a current tool shopping list.

Hugging Face: AI Agents Course

Free course; Python and basic LLM knowledge are prerequisites. Inference/hosting can incur costs.

Anthropic: Demystifying evals for AI agents

Engineering article, January 2026.

Google: Machine Learning Crash Course

Free learning material. Study the concepts relevant to your project first.

How to use the recommendations

Start with foundations only if needed. Watch a concept, then build something that proves you understood it. The Full Stack lectures are from 2023; use current SDK docs for implementation. Paid labs and certificates are optional. API and hosting usage may cost money.

The curriculum and time allowances are my synthesis. The job research uses three employer examples, not a representative market survey. Job descriptions, provider access and software change; recheck them as you study.