

AI in your own projects

Reading time: ~18 min · Prereqs: Lessons 01–07 · 4 interactive demos inside · This is where you go from using AI to building with it.

From user to builder

There is a line between "I use AI" and "I built something with AI." It's not as wide as it looks. You already know how to write good prompts. You already know how to call an API from code. The step into building is mostly about *connecting those pieces intentionally* to solve a problem you actually care about.

This lesson walks you through the decisions every AI project starts with — cloud or local? which project to pick? how to make the model answer from *your* documents? how do you turn a script into an app? — and gives you four golden rules to keep you out of the most common traps.

Decision #1 — Cloud or local?

Before you write a single line of code, you face the first fork in the road: do you send your prompts to a cloud service like Claude or ChatGPT, or do you run a model locally on your own machine using Ollama or LM Studio?

Neither answer is always right. The right answer depends on four things: how sensitive your data is, how much traffic you expect, how fast you need the response, and what hardware you have. The wizard below runs through exactly those factors and gives you a personalised recommendation.

INTERACTIVE 1 OF 4 — CLOUD-VS-LOCAL DECISION WIZARD

1. How sensitive is the data you'll be sending to the model?

Not sensitive (public info, generic text)

Moderately sensitive (internal notes, business data)

Highly sensitive (medical, legal, financial, personal)

2. How many prompts do you expect to send per day?

Occasional (a few, manually triggered)

Moderate (dozens to hundreds, maybe automated)

High volume (thousands, automated pipeline)

3. How important is response speed?

Don't care — even 30 seconds is fine

Fast preferred — under 5 seconds

Near real-time — typing-speed responses

4. What's your budget situation?

Zero budget — must be completely free

Small budget OK — a few dollars a month

Flexible — happy to pay for quality

5. What hardware do you have?

Laptop with 8 GB RAM (older or budget machine)

Laptop / desktop with 16 GB RAM

32 GB+ RAM or an Apple Silicon Mac (M1/M2/M3/M4)

Dedicated GPU (NVIDIA RTX / A-series)

Get my recommendation

Reset

Answer all 5 questions, then click "Get my recommendation." Your answers drive a simple scoring model — see Theory for the logic.

One thing the wizard captures but is worth saying explicitly: **you can start cloud and switch later**. The cloud API and Ollama both speak the OpenAI-compatible format. Moving between them is often a one-line code change — just a different server address. There's no commitment.

The "prototype cloud, run local" pattern

Build fast with the cloud API while you figure out whether your idea actually works. If it does, and if cost or privacy become real concerns, swap the `base_url` in your code to point at your local Ollama instance. You lose a bit of quality; you gain privacy and zero variable cost. Most serious projects end up at this "hybrid" point.

Picking a starter project

The biggest beginner mistake is not knowing what to build first. Ambitious ideas feel more exciting, but they also fail silently: you can't tell if the AI part works when the whole system isn't done yet. The best first project is **small and well-defined** — one input, one output, the whole thing running in under a weekend.

Use the picker below to find the project that matches where you are right now.

INTERACTIVE 2 OF 4 — STARTER PROJECT PICKER

Show all

First weekend project

Has data I care about

Coding required

PDF Summariser

★★★★☆ Beginner

Est. time: 2–4 hours

Key concepts: file reading, prompt construction, loop over files.

Message / Email Sorter

★★★★☆ Beginner

Est. time: 3–5 hours

Key concepts: classification prompt, JSON output parsing, category labels.

What you'd need: Python, sample messages (CSV or text), API key.

What you'd need: Python, `anthropic` or `openai` library, PDFs to summarise, API key.

Personal Q&A Bot

★★★★☆ Intermediate

Est. time: 1–2 days

Key concepts: RAG basics, embedding search or keyword search, chat loop.

What you'd need: AnythingLLM (no-code) OR Python + LangChain/LlamaIndex, your notes folder.

Auto-Reply Drafter

★★★★☆ Beginner

Est. time: 3–5 hours

Key concepts: few-shot prompting, template filling, tone control.

What you'd need: Python, sample incoming messages, your style examples, API key.

Meeting Notes → Actions

★★★★☆ Very easy

Est. time: 1–2 hours

Key concepts: extraction prompt, structured output (JSON), owner assignment.

What you'd need: Your meeting notes (copy-paste or file), API key, a text editor.

Spreadsheet Trends Explainer

★★★★☆ Intermediate

Est. time: 4–8 hours

Key concepts: CSV parsing, summarising numeric data with context, iterating over rows.

What you'd need: Python, `pandas`, your data file, API key.

Click a filter to highlight matching projects. "First weekend project" = completable in one weekend with minimal setup.

Once you've picked a project, resist the urge to add features before the core works. The PDF summariser should summarise one PDF correctly before it handles a whole folder. The message sorter should sort five test messages before it connects to your real inbox. Finish the smallest possible version, confirm it works, then expand.

The RAG idea — chat with your own documents

The single most powerful pattern for "AI in your own projects" is something called **RAG: Retrieval-Augmented Generation**. It's a mouthful, but the idea is simple: the model can only see what's in the prompt. RAG is how you get your private documents into the prompt automatically, at the moment they're needed.

Without RAG, the model's knowledge is frozen at its training cutoff — it has never seen your company's internal wiki, your personal notes, your product documentation, or your customer support history. With RAG, a search step runs first, finds the relevant bits of your documents, and passes them to the model as extra context. The model reads those snippets like a cheat sheet and answers from what it can actually see.

The diagram below shows the five-step flow. Watch it run, then hover over each stage for more detail.

INTERACTIVE 3 OF 4 — ANIMATED RAG FLOW DIAGRAM

Replay animation

Each stage appears in sequence. Hover (or tap) any stage box to see what's happening at that step.

RAG without writing any code

You don't need to build the search engine from scratch. Several beginner-friendly tools wrap the entire RAG pipeline in a point-and-click interface:

- **AnythingLLM** — a free desktop app. Point it at a folder of PDFs or notes, and it gives you a chat interface over them. Works with local Ollama models or cloud APIs.
- **OpenWebUI** — a web-based chat front-end for Ollama with built-in document RAG.
- **Flowise / LangFlow** — visual drag-and-drop builders for more complex pipelines.

Start with one of these before building your own RAG code. They let you test whether the pattern actually solves your problem before you invest engineering time.

Why not just paste all documents into the prompt?

Two reasons. First, many documents together won't fit in a context window. Second, more text in the prompt means higher cost (if cloud) and slower responses. RAG solves both problems by sending only the relevant snippets — typically just a few hundred words, not the entire corpus.

From script to app — the three layers

Every AI application, no matter how polished, is built from the same three layers:

1. **Interface** — how the user gives input and sees output. Could be a chat box, a web form, a command-line argument, or even a script that reads files from a folder.
2. **Your code** — the logic that reads the input, builds the prompt, calls the model, and does something with the response.
3. **The model** — cloud (Claude, OpenAI) or local (Ollama). This is the layer you've been learning to talk to.

A beginner's first project often has *no layer 1* at all — just a Python script that you run from the terminal with a hardcoded input. That is completely fine. Add an interface only when the core logic (layer 2 + layer 3) works reliably.

Scaffolding with a coding agent

Building layer 2 from scratch is faster than it used to be. A coding agent like **Claude Code** can do the boilerplate for you: describe what you want in plain English, and it scaffolds the project structure, writes the API calls, adds basic error handling, and

hooks up the layers. Your job is to review what it produces, test it with real inputs, and steer it when something is off.

Don't skip the review step

Coding agents can write code that looks right but has subtle bugs — or that does something you didn't intend. Read every file it creates or modifies before running it. Run it on test data before pointing it at real data. Work inside a git repository so you can always roll back.

The four golden rules

Every AI project, however small, benefits from four rules. They're not bureaucratic — each one prevents a specific, common failure. Read through each one in the checklist below and check it off. Your progress is saved between visits.

INTERACTIVE 4 OF 4 — GOLDEN RULES CHECKLIST

0 / 4 reviewed

- ☐ 1. Keep a human in the loop
- ☐ 2. Never trust AI output blindly — validate it
- ☐ 3. Protect your API keys and sensitive data
- ☐ 4. Start cheap and small, then measure and scale

Click any rule to expand the explanation and mark it read. Progress is saved in your browser.

Putting it all together: a simple project arc

Here is the full arc from "I have an idea" to "I have a working thing," compressed into the minimum number of steps:

1. **Define the problem in one sentence.** "Given a meeting transcript, extract every action item as a JSON list with owner, task, and due date." If you can't write it in one sentence, the scope is too wide.
2. **Test the prompt in chat first.** Before writing any code, paste a sample input into Claude.ai or ChatGPT and iterate until the prompt reliably produces the output you want. This is free and fast.
3. **Wrap the prompt in a script.** Put the prompt that worked in a Python script, read the real input from a file, write the output to a file. Run it once on a real example.
4. **Handle errors and edge cases.** What happens if the input is empty? What if the model returns something that isn't valid JSON? Add basic checks.
5. **Add an interface if needed.** Only at this point — once the core works — add a web page, a command-line interface, or an integration with another tool.
6. **Review the outputs manually.** Use the tool on 20–50 real examples. Look for patterns in the failures. Fix the prompt or the validation logic.

The 90-minute first project challenge

Pick the "Meeting Notes → Actions" project. You need: a free Claude.ai account, 5 minutes of meeting notes (real or made up), and a text editor. Spend 30 minutes getting the prompt right in chat. Spend 60 minutes wrapping it in a Python script. That's it — you've built your first AI tool. From there, every subsequent project is faster because you know the rhythm.

Summary

Building with AI is not fundamentally different from building anything else: define the problem clearly, start with the simplest possible version, confirm it works, then add complexity. What's different is that your "core logic" is mostly a well-crafted prompt — and the model does the heavy lifting.

The cloud-vs-local decision comes first (use the wizard), picking the right starter project comes second (use the picker), understanding RAG comes third (watch the diagram), and then the four golden rules keep you out of trouble as you scale. That's the full map.

The homework page has three concrete exercises. The first one can be done in under two hours, with no new installs. The second one gets you to a running Python script. The third one asks you to sketch a real project of your own.