

Lesson 08 — Putting it together

From “I use AI in chat” to “I built something with AI.”

A practical road map for real beginners.

AI for Dummies

Module 8

18 slides

You already know the pieces — prompts, the API, real-world examples. This lesson is where they connect into something *you actually ship*.

Decision #1 — Cloud or local?

Four quadrants: Privacy sensitivity (vertical) × Expected volume (horizontal).



The interactive wizard on the Theory page scores your situation across 5 dimensions and gives a personalised recommendation.

Cloud option — deep dive

Pros ✓

- Best output quality available today
- No hardware requirement — a Chromebook works fine
- Ready in minutes (get an API key, write 5 lines)
- Provider handles model updates, scaling, uptime
- Multimodal: text, images, code all in one call

Watch out for ⚠

- Sensitive data leaves your machine
- Pay per token — loops can surprise you
- Internet required at runtime
- Provider could change pricing or terms
- Rate limits on free / cheap tiers

Tools:

[Anthropic Claude API](#)

[OpenAI ChatGPT API](#)

[Google Gemini API](#)

Start with a cheap/fast model (Claude Haiku, GPT-4o Mini) for dev. Upgrade only when you need it.

Local option — deep dive

Pros ✓

- Zero cost per call after installation
- Data never leaves your machine — fully private
- Works offline — no internet needed at runtime
- No rate limits, no usage caps
- Same OpenAI-compatible API — easy to switch

Watch out for ⚠

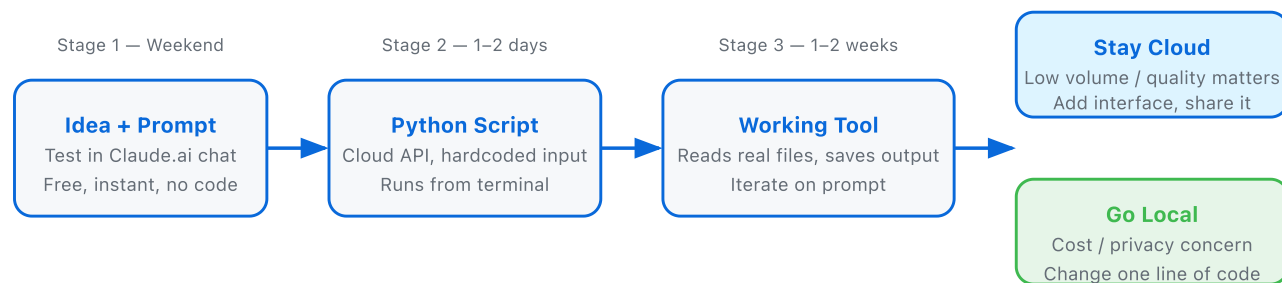
- Needs 16 GB+ RAM; 32 GB ideal
- Slower than cloud on modest hardware
- You manage model downloads and updates
- Quality gap vs. top cloud models (shrinking)
- Initial setup takes 20–40 minutes

Tools: [Ollama](#) (terminal-friendly, great API) [LM Studio](#) (graphical app, no CLI needed)

Apple Silicon Macs are excellent for local models. NVIDIA GPU also works well.

“Prototype cloud, scale local” — the smart arc

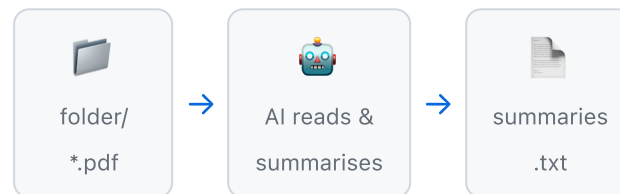
Start fast with cloud quality. Swap to local when cost or privacy become real concerns. *One line of code changes.*



The magic switch: change `base_url` from `https://api.openai.com/v1` to `http://localhost:11434/v1` and the rest of your code is unchanged.

Starter project #1 — PDF Summariser

Loop through a folder of PDFs, send each one to the model, write summaries to a text file. Perfect first project.



```
import anthropic, pathlib

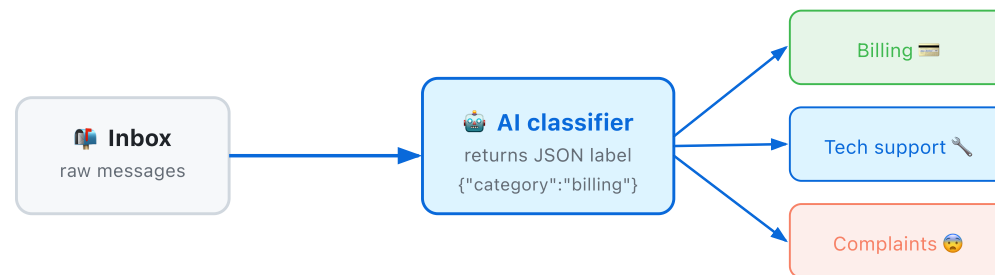
client = anthropic.Anthropic() # reads ANTHROPIC_API_KEY from env

for pdf_path in pathlib.Path("reports/").glob("*.pdf"):
    text = extract_text(pdf_path) # PyMuPDF or pdfplumber
    response = client.messages.create(
        model="claude-haiku-4-5",
        max_tokens=300,
        messages=[{"role": "user", "content":
            f"Summarise in 3 bullet points:\n\n{text[:3000]}"}]
    )
    print(f"=== {pdf_path.name} ===")
    print(response.content[0].text)
```

Time estimate: 2–4 hours · **Difficulty:** ★★☆☆ · **Needs:** Python, `anthropic`, `pdfplumber` or `PyMuPDF`

Starter project #2 — Message Sorter

Classify incoming emails or support tickets into categories. The AI returns a label — you route accordingly.



The classification prompt

Ask the model to return a **single JSON object** with a category field. Easier to parse than free text, and much more reliable.

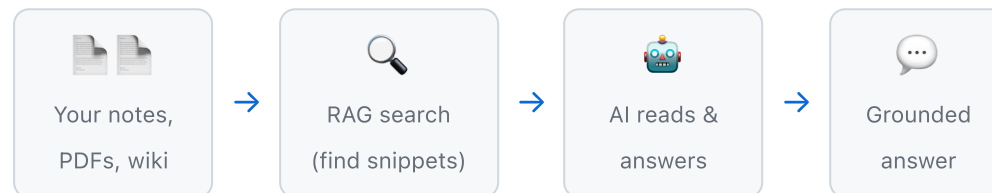
```
{ "category": "billing",  
  "confidence": "high",  
  "reason": "mentions invoice" }
```

Good to know

- Works on emails, tickets, social DMs, chat logs — anything text
- Add `"confidence": "low"` to flag uncertain cases for human review
- Test on 20 real examples before automating anything

Starter project #3 — Personal Q&A Bot

Point an AI at a folder of your own notes or docs. Ask it questions. It answers from *your* material.



No-code path (30 min)

AnythingLLM: free desktop app, point at a folder, get a chat interface. Works with Ollama (local, private) *or* Claude API (cloud). No Python needed.

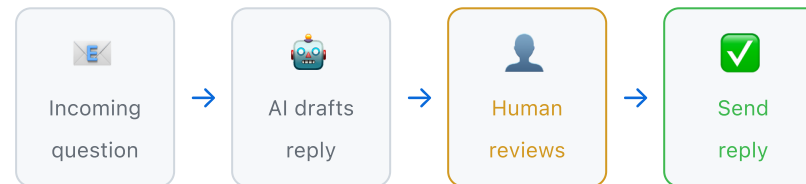
Code path (1–2 days)

Python + **LangChain** or **LlamaIndex**: split docs into chunks, embed them, store in a vector DB (Chroma, FAISS), search on each question, pass snippets to Claude. Full control.

Key insight: AnythingLLM is the right first step. Build a custom RAG pipeline only after you've confirmed the approach solves your problem.

Starter project #4 — Auto-Reply Drafter

Draft replies to common inbound questions in your tone. Human reviews & hits send. Speed up support by 3x.



The prompt pattern

Use **few-shot prompting**: include 2–3 examples of real questions and your best past replies. The model learns your tone automatically. Add a final line: *"Match the tone of the examples above."*

Good categories to automate drafts for

- Pricing / refund questions
- "How do I reset my password?"
- Feature availability questions
- Shipping / delivery status

Never auto-send without human review for anything customer-facing.

What is RAG? — the core idea, simply

RAG = Retrieval-Augmented Generation.

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 knows only its training data (frozen at cutoff). It has never seen your company wiki, personal notes, or product docs. If you ask about them, it guesses or says it doesn't know.

With RAG

A search step runs first, finds the most relevant chunks from your documents, and pastes them into the prompt as context. The model reads them like a cheat sheet and answers from material it can actually see.

Why not just paste everything? Most document collections are far too large to fit in a context window — and sending hundreds of pages with every query would be very expensive. RAG picks only what's relevant, typically just a few hundred words.

RAG flow — 5 stages (animated)

Watch the 5 stages appear in sequence. Each is explained below the diagram.

Replay animation

The animated RAG diagram on the Theory page has hover tooltips explaining each stage in detail.

“Chat with your own docs” — tools that handle RAG for you

You don't need to build the search engine. These tools do the whole RAG pipeline with a GUI.



AnythingLLM (recommended first step)

Free desktop app for Mac, Windows, Linux. Point it at a folder of PDFs or notes. Gives you a chat interface over them instantly. Works with Ollama (local & private) or Claude/OpenAI API (cloud). Zero coding.



OpenWebUI

Web-based chat front-end for Ollama with built-in document RAG. Run it locally via Docker. Great for teams who want a ChatGPT-like experience over private models.



Flowise / LangFlow

Visual drag-and-drop pipeline builders. Connect nodes: “load PDF” → “split text” → “embed” → “store” → “chat”. Good once you need custom logic beyond the defaults.



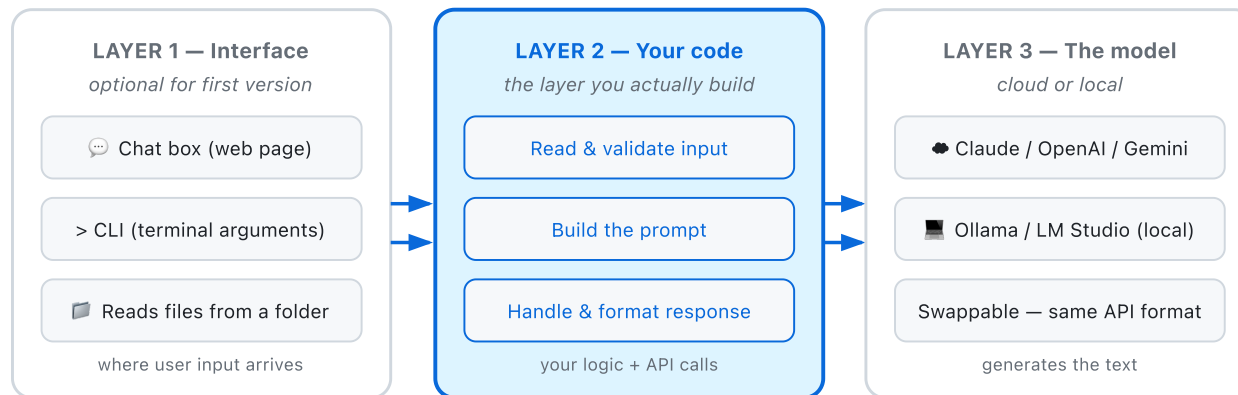
Build your own (code path)

LangChain or **LlamaIndex** in Python. Full control. Use a vector store (Chroma, FAISS). Embed with OpenAI or a

local sentence-transformer. 100–200 lines of code to get started.

The 3-layer app architecture

Every AI application, from a weekend script to a shipped product, is made of the same three layers.



Beginner tip: your first project is probably just Layer 2 alone — a Python script with no web page, reading from a hardcoded string. That is completely fine. Build Layer 1 only when the core logic works reliably.

Claude Code can scaffold this for you

Describe the app you want in plain English. The agent builds the boilerplate, wires the API calls, and sets up the folder structure. You steer and review.

What you type

Build a Python script that:

1. Reads every .txt file in ./emails/
2. Classifies each as billing, support, or spam
3. Outputs a CSV: filename, category, confidence
4. Uses Claude Haiku, reads ANTHROPIC_API_KEY from env

What Claude Code does

- Creates project folder & `requirements.txt`
- Writes the main script with a classification loop
- Adds JSON output parsing + error handling
- Writes a `.env.example` for the API key
- Adds a `README.md` with run instructions

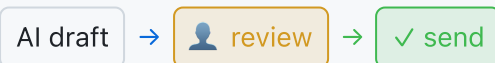
Always: review every file it creates before running. Work inside a git repo so you can undo anything. Test on a small sample before pointing it at real data.

Golden Rules #1 & #2

Rule #1 — Keep a human in the loop

Anything that touches **money, health, law, or goes to real customers** needs a person reviewing the output before it acts.

Run manually for at least 20–50 examples. Understand the failure modes. Only automate fully once you accept the cost of a bad output slipping through.



? Rule #2 — Never trust output blindly

Validate AI output — especially **numbers, facts, and code**. The model sounds equally confident whether it is right or completely wrong.

✗ Don't

- Copy-paste output directly
- Trust numbers at face value
- Ship AI-written code unread

✓ Do

- Check facts against a source
- Run code on test data first
- Ask AI to show its reasoning

Golden Rules #3 & #4

🔒 Rule #3 — Protect keys & data

An exposed API key is a stolen credit card. Keys can rack up large charges in minutes if found by a bot scanning GitHub.

❌ Never do

- Key in code
- Key in a chat prompt
- Commit `.env` to git

✅ Always do

- Key in `.env` file
- `.env` in `.gitignore`
- Read via `os.environ`

Sensitive data (medical, legal, personal)? Use a local model so nothing leaves your machine.

🔍 Rule #4 — Start cheap, measure, scale

Confirm the core logic works on **10 examples** before running it on 10,000. Cloud APIs bill per token — a loop with a bug can cost surprisingly fast.

Dev: cheap model, 10 examples ~\$0.001



Test: mid model, 100 examples ~\$0.05



Production: top model, 100k/day ~\$50+/day



Set a spending cap in your API dashboard from day one. Most providers offer this.

A first-weekend project plan

The “Meeting Notes → Action Items” project. Completable in 90 minutes with a free Claude account and a text editor.

- 1 **Write sample input.** Paste 5 minutes of meeting notes (real or invented) into a text file. Example: “Alice to send the budget by Friday. Bob will review the designs by EOD Thursday...”
- 2 **Get the prompt right in chat first (30 min).** Open Claude.ai. Paste your notes. Prompt: “Extract every action item as a JSON list with fields: owner, task, due_date.” Iterate until reliable.
- 3 **Wrap in a Python script (45 min).** Read the notes file. Call the Claude API with the working prompt. Print the JSON. Done — that's a real AI tool.
- 4 **Test on 5 real meetings.** Check every output manually. Where does it go wrong? Fix the prompt. Add handling for empty input or missing fields.
- 5 **Optional: add a CSV output.** Write the action items to `actions.csv` so you can open it in a spreadsheet. Now it's actually useful for your team.

The lesson: every project after this is faster because you know the rhythm — prompt in chat, wrap in code, test with real data, iterate.

Recap & next: Safety

What we covered

- Cloud vs local decision matrix — 4 quadrants
- Cloud & local deep-dives — pros, cons, tools
- 4 starter projects with flow diagrams
- RAG explained + animated 5-stage flow
- The 3-layer app architecture
- Claude Code for scaffolding
- 4 golden rules + verification flows
- A concrete first-weekend project plan

Lesson 09 — Safety preview

- **Hallucinations:** why they happen, how to catch them
- **Privacy:** what data leaves your machine and when
- **Bias:** what it means & why it matters
- **Cost:** spending caps, token budgets
- **Agents & the leash:** safe limits for agentic tools
- **AI is not a professional:** law, medicine, finance

Your homework: pick one project from slides 6–9. Spend 30 minutes getting the prompt right in Claude.ai. The homework page has three guided exercises to help you ship your first real AI tool.
