

TEMPLATE 04 · FOR FOUNDERS BUILDING AI PRODUCTS

AI feature scoping: code, prompt, RAG or fine-tune?

Most AI features are over-engineered before they are tested. Teams reach for fine-tuning when a better prompt would do, or build a vector database for data that already lives in a SQL table. This worksheet walks one feature through five steps: describe it, choose the simplest approach that can work, estimate the effort, check the unit economics, and decide go or no-go.

60–90 minutes per feature

5 steps + worked example

Updated October 2026

Step 1 — Describe the feature

Input → output

What goes in (text, file, record, audio) and exactly what comes out (format, length, where it lands).

Who uses it and how often

User role, tasks per user per day or month, peak volume.

Cost of a wrong answer

Annoying (user edits it), costly (money, time, trust) or dangerous (health, legal, safety)? This sets how much review and evaluation you need.

Latency budget

Real-time (under 1 s), interactive (under 10 s) or background (minutes are fine)?

Step 2 — Choose the simplest approach that can work

Answer the questions in order. Stop at the first "yes" — that is your starting approach. You can always move up a level later; moving down is rare because nobody wants to delete work.

1. **Can the logic be written as rules, math or a lookup?** → Build it in plain code. No model. It is cheaper, faster and testable.
2. **Does the base model already know enough, and the task is about reasoning, writing or extracting from the input you give it?** → Prompt an API model with a clear output schema.
3. **Does it need to read or change structured data (database rows, CRM, calendar, an API)?** → Prompting + tool (function) calling. Do not embed rows into a vector database.
4. **Does it need knowledge from many documents the model has not seen (help center, contracts, wiki, PDFs)?** → RAG: retrieve relevant passages, then prompt with them. Show sources.
5. **Is prompting plateauing on format, style or a narrow classification — and do you have 500+ high-quality examples?** → Consider fine-tuning, usually a smaller model, to cut cost or latency. Keep the prompted version as the baseline.
6. **None of the above?** → Training your own model is almost never an MVP decision. Revisit the problem framing first.

Approach	Use when	Avoid when	Data you need	First version
Plain code	Rules, math, lookups, routing	Inputs are free text with endless variety	Rules and test cases	Days
Prompting	Writing, summarizing, extracting, classifying from given input	Answers depend on private knowledge	20–50 sample inputs for testing	1–3 days
Prompt + tools	Reading or acting on structured systems	You can't give the model safe, scoped permissions	API access, a list of allowed actions	3–10 days
RAG	Answering from a large, changing body of documents	Data is small enough to fit in the prompt, or is tabular	Clean documents, access rules, refresh plan	1–3 weeks
Fine-tuning	Narrow, repetitive task where prompting plateaus; cost or latency must drop	You have under ~500 good examples or the task keeps changing	500–5,000 labelled examples	2–4 weeks + data work
Own model	Research-grade problem, unique data at scale	Almost every MVP	A lot	Months

Your approach and why

Which level did you stop at, and what would make you move up a level later?

Step 3 — Estimate the effort

Ranges are working days for one experienced designer-engineer with modern AI-assisted tooling. S = narrow feature with clean inputs; M = typical; L = messy data, many edge cases or high cost of error. Cross out the rows you don't need and write your own estimate in the last column.

Workstream	S	M	L	Yours
Prompt, output schema and examples	1	2–3	5–8	
Golden set and eval harness	1	2–3	5+	
UI for AI states (streaming, edit, errors, sources)	2	4–6	8–12	
Tool / API integrations (per system)	1–2	3–5	8+	
Retrieval: ingest, chunk, embed, index, refresh	—	4–7	10–15	
Guardrails: moderation, injection, permissions	0.5	1–2	4+	
Cost controls, logging, quotas	0.5	1–2	3	
Fine-tuning: data cleaning, training, comparison	—	5–10	15+	
Total (add 20% for unknowns)				

Step 4 — Check the unit economics

Cost per task = (input tokens × input price + output tokens × output price) ÷ 1,000,000. Then **cost per user per month** = cost per task × tasks per user per month. Measure tokens on real inputs; take prices from your provider's current price list.

Line	Example (illustrative prices)	Yours
Input tokens per task (prompt + context)	3,000	
Output tokens per task	500	
Price per 1M input / output tokens	\$3 / \$15	
Cost per task	\$0.009 + \$0.0075 = \$0.0165	
Tasks per user per month (p95 user)	400	
AI cost per heavy user per month	\$6.60	

Line	Example (illustrative prices)	Yours
Plan price per user per month	\$29	
AI cost as % of price (aim under 20–25%)	23%	

If the heavy-user cost is above a quarter of the price: shorten the context, cache repeated work, route easy tasks to a smaller model, or meter usage above a quota.

Step 5 — Go / no-go

☐ **The feature saves a measurable amount of time or money for one named user**
Not "users will love AI".

☐ **You chose the lowest level from Step 2 that can work**
And wrote down what would make you move up.

☐ **A golden set of at least 30 real inputs exists**
With expected outputs or grading criteria.

☐ **The acceptable error rate is written down**
And the UI makes errors cheap to catch and fix.

☐ **Heavy-user AI cost is under ~25% of the price**
Or there is a quota.

☐ **There is a fallback when the model is down or wrong**
The rest of the product still works.

Worked example — "Draft a reply" in a support inbox

Feature: for a B2B SaaS helpdesk, draft a reply to an incoming ticket that the agent edits and sends. ~120 tickets per agent per week; wrong answer is costly (customer gets wrong instructions) but always reviewed by an agent; interactive latency (under 10 s).

- **Step 2:** Level 1 (rules) no — tickets are free text. Level 2 (prompting) not enough — answers depend on the product's help center (400 articles) and the customer's plan. Plan and account data → **tools** (look up plan, seats, last invoice). Help center → **RAG** with article links shown to the agent. No fine-tuning: content changes weekly.
- **Step 3:** prompt + schema 2–3 d, eval set (60 historical tickets with the agent's actual reply) 3 d, UI for draft + sources + edit 5 d, 2 tool integrations 4 d, retrieval over help center 5 d, guardrails 1 d, logging and quotas 1 d → 21–22 d + 20% ≈ **5 weeks**.
- **Step 4:** ~4,000 input + 400 output tokens per draft → about \$0.018 at the illustrative prices above; 480 drafts per agent per month → **~\$8.60 per agent per month** against a \$49 seat → 18%. Go.
- **Step 5:** success = agents send ≥ 50% of drafts with light edits within 4 weeks; fallback = "draft unavailable" and the normal editor; kill switch per workspace.

Want this done for you?

Stepikin Studio designs and builds AI products end to end and ships MVPs in weeks. Get an estimate: stepikin.com/estimate · Book a call: vizi.club/book/aleksey_stepikin