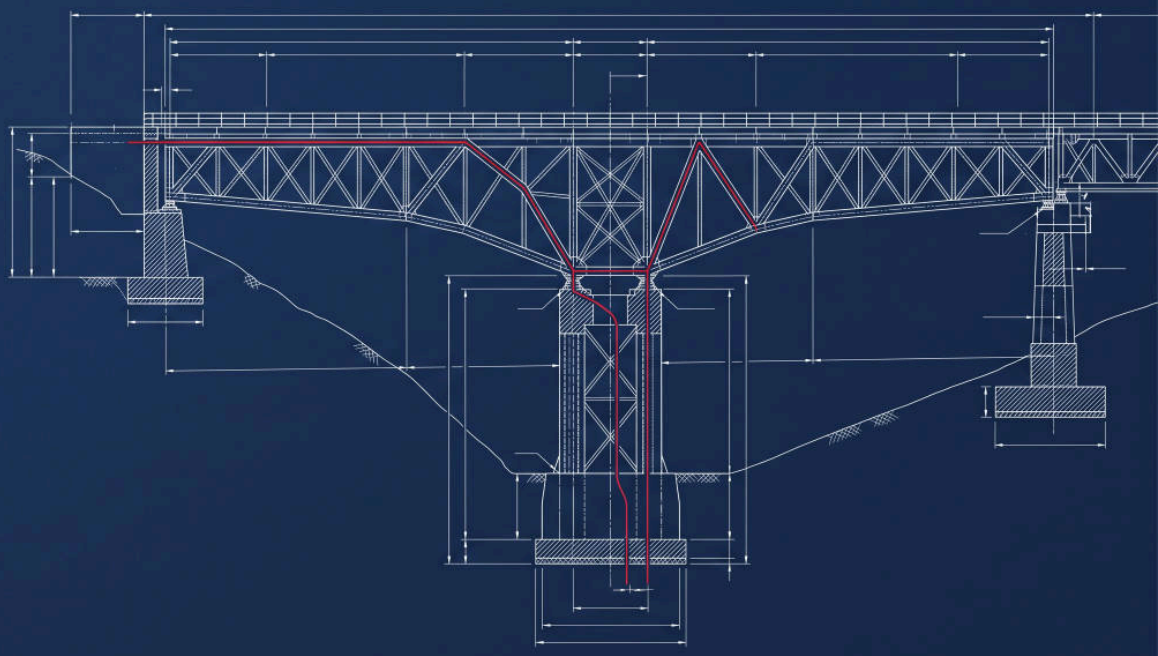

CHAPTER SEVENTEEN

Capstone: The Whole System

SECTIONS 17.1 TO 17.8

Sixteen chapters assembled on one page, then reduced to something more portable: a small number of habits that work on systems built with tools that do not exist yet.

CHAPTER SEVENTEEN

Contents

PART IV · PRODUCTION

Sixteen chapters of architecture, assembled on one page and then reduced to something more portable: a small number of habits that work on systems built with tools that do not exist yet.

17.1 The Whole System

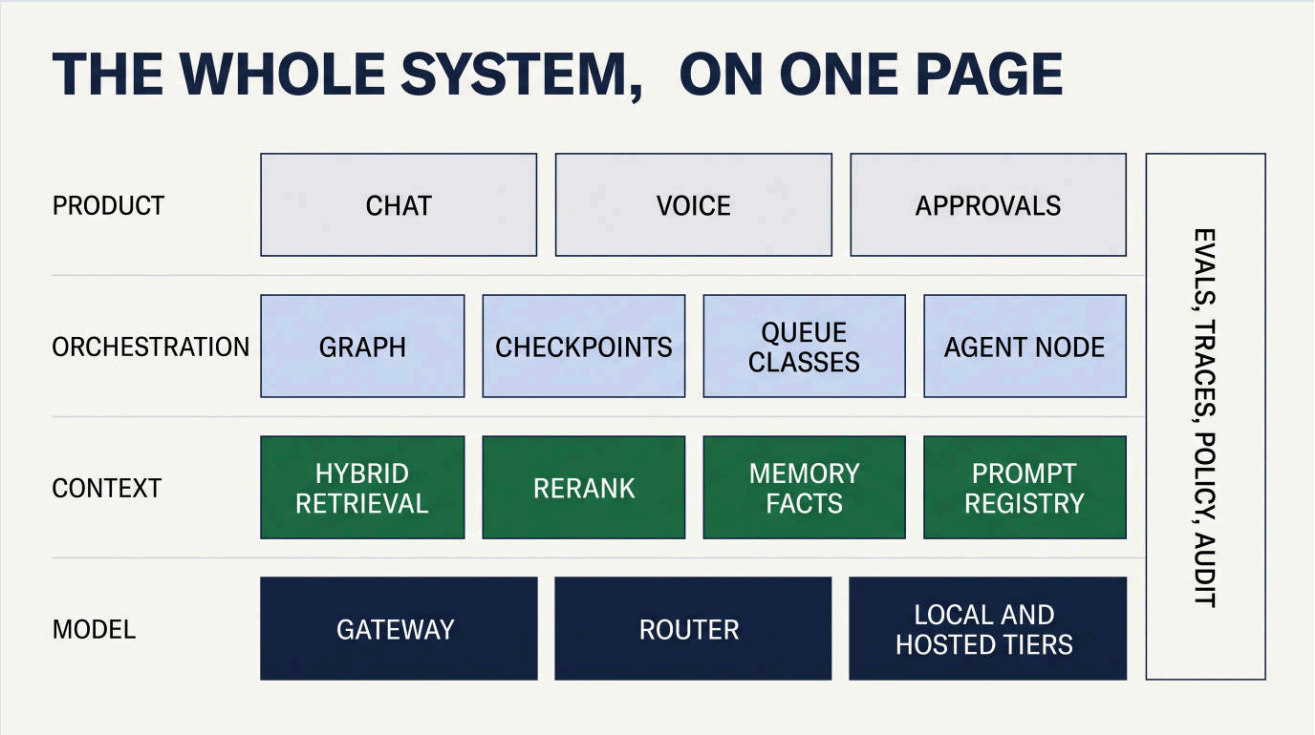

Figure 17.1 Four layers, and the band that cuts through all of them.

Section 1.7 gave the four layers before there was anything in them. Here they are, populated.

Layer	What it contains now	Dominates
Model	Gateway with deadline, jitter, breaker and spend circuit; tier router; local and hosted tiers; declared model capabilities	Cost and latency
Context	Structural chunking; hybrid retrieval and reranking; memory as typed facts; prompt registry; the context budget	**Answer quality**
Orchestration	Declared graph with checkpoints and interrupts; queue classes; bounded agent node; tool registry with risk tiers	Reliability and debuggability
Product	Citations, abstention, service tiers, approval surfaces, voice	Whether it is trusted

Cutting through all four: **evals, traces, policy and audit.** They are not a fifth layer because they are not a stage in the request path; they are the instruments by which the other four are known to work.

It is worth walking one request through the assembled system before compressing it further. A tenant user asks a question. The gateway takes the call under a deadline, and the tier router decides which model the task earns, per section 3.6. Admission happens before any expensive work: section 8.6's endpoint accepts the job into a queue class and answers 202, a receipt rather than a promise.

A worker picks the job up. Retrieval runs broad across the dense and lexical legs and keeps narrow, section 6.7's rule, and section 6.8's assembler packs the survivors into a budgeted prompt with citations attached. The draft that returns is not yet an answer: section 6.9's grounding checks compare it against the cited spans, with abstention as one honest outcome, and section 15.1's output policy inspects the result before anything is allowed to happen because of it.

Alongside every stage, section 1.6's trace seam records versions, tokens, cost and outcome, and section 16.2's attribution puts the spend against the tenant and the feature that caused it. Nothing in the walkthrough is new by this point; what the capstone adds is the observation that the stages compose without negotiation, because each bounds its own dimension. Figure 17.1 is this walkthrough, drawn.

The diagnostic from section 1.7 held up across the whole book. When quality is poor, engineers reach for the model layer, because that is the layer with a knob labelled "better model". Section 7.2's failure taxonomy then measured it: on a plausible distribution, eighty-four percent of failures happened before the model was called. The reflex is wrong in a specific, repeatable direction, and knowing that is worth more than most individual techniques in this book.

17.2 The Decision Log

The single most useful artefact to leave behind, and almost nobody writes one. It is not documentation of what the system does, which the code already says. It is a record of *what was decided, what it cost, and what would change the decision.*

```
## D-014: Hybrid retrieval with reciprocal rank fusion
Date: 2026-03-11        Status: active        Owner: retrieval

Decision   Add a BM25 leg alongside dense retrieval; fuse by RRF, not score.
Because    Failure labelling (7.2) put 24% of failures at "not retrieved",
           concentrated on part numbers and clause references.
Rejected   Better embedding model: does not address exact-identifier misses.
           Score normalisation: breaks as the corpus grows (7.6).
Costs      +18 ms p50 retrieval; one more index to operate.
Measured   recall@20 0.81 -> 0.94; end-to-end accuracy 0.79 -> 0.86.
Revisit    If exact-identifier queries fall below 5% of traffic, or if
           index operation becomes a burden without a matching gain.
```

Four fields do the work. **Because** ties the decision to a measurement rather than a preference. **Rejected** stops the same debate recurring every six months. **Costs** makes the trade explicit, so a later reader can tell whether it still holds. And **revisit** names the condition under which the decision expires, which is the difference between a decision and an accumulation.

Write one for every promotion trigger you cross. Sixteen chapters of this book are, in effect, a template for the ones worth writing.

17.3 Reading an Unfamiliar System

A skill worth more than any single technique, because you will inherit more systems than you design. The order matters: it goes from the things that constrain everything to the things that can be changed on a Tuesday.

Start with ingestion, not the prompt. Section 6.2 established that everything downstream inherits the extraction. Look at what a chunk carries: if it has no page, no section and no hash, you already know it cannot cite, cannot re-ingest cheaply and cannot delete. That single dataclass tells you more about the system's ceiling than the model choice does.

Then find the seams. Grep for provider SDK imports. If they appear in forty files, the model layer cannot be changed, and every subsequent recommendation you make will be more expensive than it should be.

Then find the budget. If prompts are assembled by concatenation, the context layer is unbounded and section 2.10's failures are all live and unreported.

Then ask what stops the damage. Section 15.1's question. Trace one untrusted string from a document to an effect and see what it passes through.

Last, look at the prompts. They are what everyone shows you first and they are the cheapest thing to change.

17.4 The Twenty-Minute Review

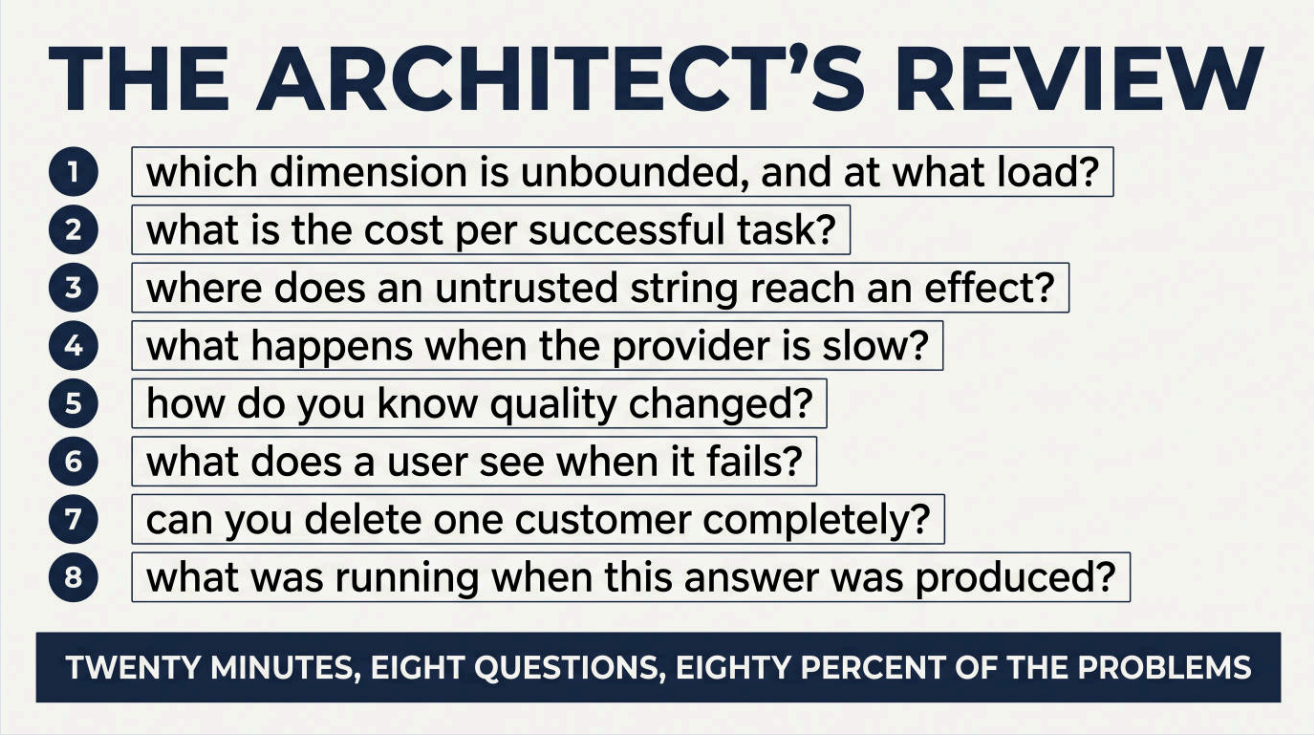

Figure 17.2 Eight questions. They work on any system in this field, including ones built with tools that do not exist yet.

Eight questions, each tied to a chapter, each answerable in about two minutes by someone who built the system. The value is less in the answers than in which ones produce a pause.

Question	From
Which dimension is unbounded, and what load exercises it?	1.1
What is the cost per *successful* task, and how do you know?	1.6
Where does an untrusted string reach an effect?	5.9, 15.1
What happens when the provider gets slow rather than fails?	3.6
How would you know quality changed if nothing was deployed?	14.5
What does a user see when the system cannot answer?	6.9, 15.5
Can you delete one customer completely, including traces?	15.4
What versions were running when this answer was produced?	1.2

Judge the review by the hesitations, not the answers. A confident wrong answer to any of these is a smaller problem than a pause, because the pause marks a question the team has never had to answer, which means the mechanism to answer it does not exist. Section 1.1 said the architect's job is knowing which dimension is unbounded and at what load; these eight questions are that job in a form you can run on a system you met an hour ago.

17.5 What to Build First

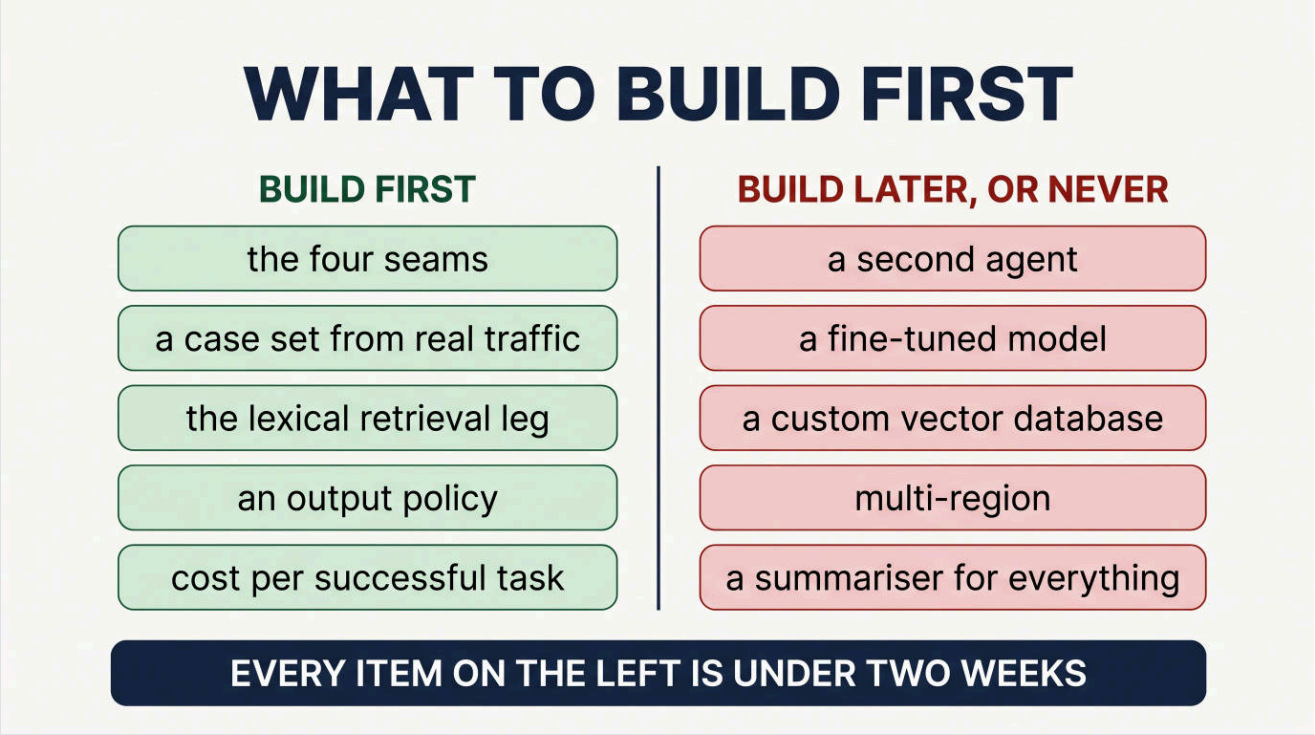

Figure 17.3 Every item on the left is under two weeks. Most of the right-hand column is never.

Five things, in order. Each is under two weeks, each is cheap now and expensive to retrofit, and together they are most of the difference between a system that can be improved and one that can only be replaced.

The four seams. Model, prompt, trace, eval. Section 1.6: about an hour each during a prototype, about a quarter each afterwards. Everything else in this list depends on at least one of them existing.

A case set from real traffic. Eighty labelled cases from your own traces. Section 14.2. Without it, every subsequent decision in this book is a preference, and every argument about quality is settled by seniority.

The lexical retrieval leg. Section 7.5: a day of work, eighty lines, no new infrastructure, and it addresses a failure class no embedding upgrade touches. The best ratio of gain to effort in the book.

An output policy. Section 15.1: one chokepoint, four checks, before any effect. Two weeks, and it converts unbounded failure into bounded failure.

Cost per successful task. Section 1.6. Not cost per call. It requires a measured success rate, which requires the case set, which is why it comes last and why it validates the rest.

17.6 What Not to Build

The less popular list, and the one that saves more time. Each of these is sometimes correct; each is usually reached for too early, and the reason is worth stating.

Deferred	Why, and what to do instead
A second agent	Section 9.9. Four agents at 95 percent complete correctly 81 percent of the time before any of them takes a step. Add an agent only to add a boundary, and different tools is not a boundary.
A fine-tuned model	Section 5.2's stopping rule comes first. Most teams reaching for fine-tuning have a context problem, and section 1.7's distribution says the context layer dominates.
A custom vector store	Section 2.9. Measure your index recall against exact search before concluding the store is the constraint. It usually is not, and the default parameters usually are.
Multi-region	Section 16.5. Ask whether the requirement is "serve from X" or "do not store in Y". The second is weeks; the first is quarters.
Summarising everything	Section 11.3. Lossy compression with no error signal. Prefer structured state, and summarise only what will not fit a schema.
Self hosting to save money	Section 4.11. Against a cheap hosted model it loses at every utilisation. Against a frontier model it depends entirely on duty cycle, which is a property of your traffic rather than your hardware.

THE PATTERN BEHIND THE LIST

Every entry is a case where the appealing move addresses a real problem that the team does not actually have, while the unglamorous move addresses the one they do. The common cause is that *the visible layer is not the dominant layer*: prompts and models are what you look at, and context and orchestration are what determines the outcome.

That is why section 7.2's failure labelling is the single most valuable afternoon in this book. It is the cheapest available correction to a bias that otherwise costs quarters.

17.7 The Architect's Checklist

The eight questions of section 17.4 are for reviewing a system. These are for building one, grouped by the layer they belong to, and each traceable to a chapter that argued for it.

Model layer. One module owns every provider call. Deadlines govern retries, not attempt counts. Jitter on backoff. A spend circuit with a per-minute window, not only daily. Dated model versions, and the returned model recorded. A second provider that has been tested, not merely configured.

Context layer. Chunks carry page, section, tenant, version and content hash. The embedded text is not the displayed text. Retrieval is broad and the prompt is narrow. A lexical leg exists. The context is a budget with named regions and overflow as an event. Memory writes are gated and provenanced.

Orchestration layer. Topology is declared and validated at startup. State is typed and checkpointed between nodes, so a run survives a deploy and resumes without repeating paid work (section 10.5). Every loop has a per-node counter. Budgets bound steps, cost and wall clock. Tools are allowlisted per node through the registry's intersected scopes (section 12.4) and risk-tiered. A third-party tool server is untrusted code: its schema and description are pinned, so capability drift arrives as a diff to review rather than a surprise (sections 12.3 and 12.9). Irreversible actions require approval and never auto-retry.

Product layer. Citations resolve to something a user can check. Abstention is a distinct, visible state that says what was searched. Degradation is a ladder with tiers on the response object. Refusals carry a route forward. Where there is a voice, the budget is time to first audio (section 13.1), and the failures nobody can see are measured rather than listened for (section 13.7).

Across all four. One trace record per model call, carrying versions, tokens, cost and outcome. A case set from real traffic. Deterministic scorers before judges. A gate with deliberate thresholds and no tolerance for injection. Cost attributed per tenant and feature. An audit trail with its own retention. A runbook someone other than its author has executed.

If you keep one sentence from this book, keep the reframing. Stop asking how to get the documents to the model, and start asking what the minimum evidence is and whether you can prove the answer came from it. The first question produces a pipeline. The second produces an architecture, and it is the one that can be trusted by someone whose job depends on the answer.

17.8 Where the Field Is Moving

A book written about a field this fast has an obligation to say which of its claims are durable and which are dated. Here is my honest division.

What will change

The numbers. Every price, context window, latency figure and model name in this book is perishable. The arithmetic that produced them is not: section 2.11's KV cache formula, section 8.1's ceiling, section 4.11's utilisation sweep and section 1.6's cost per successful task all take current values as inputs.

Some techniques become unnecessary. Longer contexts and cheaper tokens will reduce the pressure that makes some of Part II's machinery worth building. Reranking is the likeliest to be absorbed: it is a pure quality-for-latency trade with no product surface, so as cross-encoders become cheap enough to run inside every retrieval call, providers will fold them in and section 7.7's decision becomes a default you inherit. What survives is the measurement, because a default you cannot evaluate is section 2.9's index problem again. Chunking may matter less as windows grow, but section 7.10 is the reason it will not vanish: the constraint was never only fitting the window, it was that answer accuracy falls as distractors accumulate. Whole-document context does not repeal dilution; it industrialises it.

Voice collapses inward. Native audio-to-audio models will absorb more of chapter 13's cascade, and the latency engineering that made it fit the budget goes with it. What they do not absorb is the text seam: grounding checks, the output policy and the audit trail all operate on strings, and section 13.1's trade, latency against controllability, does not disappear because one side got cheaper. Expect the seam to move, as section 13.1's hybrid already shows, from

between every stage to around the decisions that carry risk. Where it sits will keep being a design decision; that it must exist somewhere is the durable part.

The cost tricks get absorbed; the metric does not. Provider-side prompt caching and batch pricing are doing to Part I's cost engineering what managed vector stores did to Part II's infrastructure: turning technique into a checkbox. That is welcome, and it changes nothing about section 16.2's discipline, because an invoice is still not an attribution. Cost per successful task survives every pricing model a provider can invent, because its denominator is measured by your evals rather than billed by anyone.

Model capability rises. Per-step reliability improves, which moves section 9.1's compound arithmetic in favour of longer loops. It does not remove the arithmetic; at 99 percent per step, twenty steps is still 0.818.

What will not

The five properties of section 1.6. Marginal cost per request, unbounded latency proportional to output, statistical correctness, one channel for instruction and data, and dependencies that change underneath you. These are properties of the paradigm rather than of any generation of models.

The reframing. Instructions and data share a channel, so enforcement lives on the output side. Every improvement in model capability makes an injected instruction *more* capably followed, not less.

Compounding. Reliability multiplies over steps, cost multiplies over turns, and both are arithmetic rather than engineering.

The layer distribution. Context will keep dominating quality, because what the model sees will keep mattering more than which model sees it.

Measurement. A system whose quality is unknown will keep being a system that cannot be improved, whatever it is built from.

> THE BET WORTH MAKING ABOUT THE NEXT FEW YEARS
>
> The gap between teams will not be model access, which is commoditising, nor technique, which is published. It will be **whether they can tell what their system is doing.** The teams with a case set from real traffic, a trace that names versions, and a cost denominated in successful tasks will adopt each new capability in days. The teams without will adopt it, observe that something changed, and be unable to say whether it helped.
>
> That is not a prediction about AI. It is the same thing that separated teams before this, and the reason it is worth saying is that the tooling in this field makes it unusually easy to skip.

A Closing Note

The book has argued throughout that the interesting work in this field is not prompt craft or model selection, but the ordinary discipline of bounding what is unbounded, measuring what is claimed, and deciding what an output is permitted to cause.

Most of what made the difference across these seventeen chapters was unglamorous: a tokenizer regex that keeps `A-1042/B` in one piece, a checkpoint written after a node rather than before, a field called `stated` on a memory record, a form feed that hid a duplicated figure number. None of it is the part anyone demonstrates. All of it is the part that determines whether a system can be improved by the person who inherits it.

The findings worth carrying, each computed rather than asserted:

Finding	Where
Grouped query attention is a 4.6x serving difference on an identical model, reported by no leaderboard	2.8
Output length dominates interactive latency, not prompt length	2.11
Routing by task type saved 80 percent of spend with no quality trade	3.6
Self hosting is decided by utilisation, not by the hourly rate	4.11
The optimal few-shot count was two, and the fourth example was net-negative	5.2
84 percent of retrieval failures happened before the model was called	7.2
Recall climbed while answer accuracy fell 22 points	7.10
Sixteen workers at fourteen seconds is 1.1 requests per second, and 50 needs 700 workers	8.1
A fully saturated worker pool reads ten percent CPU to the autoscaler	8.11, 16.4
Five steps at 95 percent beats ten steps at 97 percent	9.1
A run resumed after approval re-ran two nodes of five; the work already paid for was not paid for again	10.7
The per-node loop guard stopped a stuck run at three steps, where the global budget would have allowed a hundred	10.9

Each of those contradicted something I expected before running it. That is the argument for the method rather than for the findings: **compute the number, encode it as a test, and let a future change that breaks the argument fail loudly.**

The systems in this book are called thinking systems, which is a convenient shorthand and not quite true. They are systems that produce plausible text, wrapped in enough structure that the plausible text can be trusted, checked, bounded and paid for. The structure is the engineering. It is also, on the evidence of these seventeen chapters, most of the work.