Stop Renting Intelligence

The Local LLM Primer

Fathom — Fine-Tune A Model

Version 2.0 — April 2026

Contents

Introduction

Somewhere in your company, a finance director is staring at a cloud LLM invoice and doing a slow, unhappy arithmetic. Somewhere else, an engineer is discovering that the twelve-hundred-pound graphics card under her desk can run the model her company is paying sixty thousand a year to rent. Between those two people sits a decision that used to be hard and isn't any more.

This book is about that decision: when the local model wins, when it doesn't, and how to ship one without embarrassing yourself.

We will spare you the ideology. The numbers are enough.

Who this book is for

You are probably the Head of Engineering, VP Engineering, or CTO at a company somewhere between fifty and five hundred people. Your Claude or OpenAI bill crossed some uncomfortable threshold this quarter and your CFO forwarded the invoice with a question mark. Your compliance team has started asking where the prompts go. Your engineers keep pointing at Reddit threads about local LLMs with a kind of wistful energy, and you don't know whether they are right, or whether they are chasing a hobbyist dream that will cost you six weeks and no revenue.

You do not have time to read forty blog posts. You do not want to earn an ML PhD to decide whether to buy a GPU. You want a defensible answer before the next board meeting.

This is that answer.

Who we are

We are Fathom. We are building a fine-tuning service for local LLMs in regulated industries. We are writing this book because we have to know the material cold anyway, and writing it publicly is a shorter path to competence than hoarding it.

We have shipped zero paying fine-tunes. When we wrote the first version of this book, we pretended otherwise. We claimed "50+ deliveries" in a sales chapter. It was aspirational marketing copy that leaked into the sections around it, and then into the outreach templates that would have gone to customers. A technical audit caught it before any of that shipped, and we halted. This version is the book we should have written the first time.

What we have is different from what we claimed: - Three verified inference benchmarks on hardware we paid for (Qwen-2.5-7B at 106 tok/s on an RTX 2070; gemma3:4b at 83 tok/s on the same card; Qwen3-8B at 27 tok/s on an M5 MacBook) - One failed fine-tune (V2 — an LoRA adapter that scored 33% on eval against a 34% base, net negative, documented in the appendix with the training config, the loss curves, and the post-mortem) - One in-progress fine-tune (V3 — the one we expect to actually beat baseline, not yet delivered at the time of writing) - A commercial case study — our first Coalby client, Graphene Products, received three briefs in day one from our specialist agent team - A lot of reading, a lot of arguing, and the hardest-won lesson in the book, which is that most of our prior advice on fine-tuning was wrong because it assumed a success we had not yet earned

If this sounds like thin credentials for an authoritative book: it is. We decided to write the book anyway because the alternative — waiting until we had been shipping for two years — would mean the book was written in 2028 and was about a landscape that had moved on twice.

The bargain we are offering you: we will show you our work, including our failures. You will get the current edition free. You will get every subsequent edition free, because the subsequent editions will be better than this one — honest numbers accumulate — and because a book whose authors are still learning is more useful than a book whose authors have finished.

If you want a book by a 50-delivery veteran, there are several. They will have fewer things to apologise for and fewer things to show you. That is a real trade-off. We are offering you the other side of it.

What you will be able to do after reading this

- Decide whether your workload belongs on Claude, on a local base model, or on a fine-tuned adapter — without needing to run a six-week pilot to find out
- Pick the right hardware for the right job, with the rent-vs-buy math cold
- Understand exactly what fine-tuning changes and what it does not, so you never again hear "can we fine-tune a model to know about our product?" and have to answer "no, that is RAG" without being able to explain why cleanly
- Run an evaluation harness that will tell you whether your fine-tune helped or hurt before you ship it to anyone

- Hand a procurement team the answers they actually care about, including the ones that have nothing to do with inference speed
- Write a one-page board memo justifying any of the above decisions

What this book will not try to do

It will not convert you. If cloud LLMs are the right answer for your workload, this book will tell you so explicitly. There are three whole scenarios in the economics chapter where cloud wins and local does not — one of them by 30x on pure cost. We are not trying to sell you a service; we are trying to earn the right to your attention long enough that if you ever do need what we offer, you already know we won't waste your time.

It will not be neutral. We have opinions, and the opinions are backed by benchmarks we ran ourselves, on hardware we paid for. When we say "Qwen-2.5-7B is the right default for classification tasks at our scale," it is because we measured it on a 2018-era graphics card doing 106 tokens per second with 8/10 task quality — not because it sounds right.

It will not be finished. The pricing tables update live from the Vast.ai API. The model recommendations will shift as new models prove themselves. Every time we catch ourselves wrong, we will publish the correction in the errata appendix, and the version number will tick. If you are reading a version later than the one written today, something got better. If you are reading a version earlier than the one written today, we were more confident than we should have been.

How to read this book

If you want one answer, fast: read Chapter 1 (Economics) and stop. It contains the decision framework in six rules. For 80% of readers, that is enough.

If you are evaluating hardware: Chapter 2 is the hardware buyer's guide. Come back to Chapter 3 when you have chosen a model size and need to know which quantisation fits.

If you are considering fine-tuning: read Chapter 4 carefully, and then read the V2 post-mortem in Appendix A before you spend a single engineer-hour. Chapter 4 is the craft; Appendix A is what happens when you ignore half of it, written by the people who ignored it.

If you are selling this to an internal stakeholder: Chapter 6 is the sales chapter. It contains the three-way math (build vs buy vs consultancy) and a one-page board memo template you can copy.

If you are a procurement or compliance officer who has inherited this PDF from an engineer: Appendix B is written for you. DPA language, PI insurance levels, IP ownership, escrow, data residency. Everything you would normally have to ask three vendors for.

The character in this book

There is a person running through every chapter of this book. Call her Priya. She is the Head of Engineering at a 180-person UK fintech. Her Claude API bill is eight thousand pounds a month and growing. Her compliance officer emailed her last Tuesday asking where the prompts went. She has six open tabs on Reddit, one on Hacker News, and a calendar invite from her CFO that simply says "AI costs — 15 min Thursday."

We made Priya up, but every decision she makes in this book is a decision we have seen made in the real world by someone with her job. When we introduce a framework, Priya tests it. When we argue for a model choice, Priya is the one buying it. When we show a spreadsheet, it is because Priya needs it on her screen by Thursday.

You probably are not Priya. But if you know someone like her, this book is for them. And if you recognise pieces of your own week in hers, that is because the decision she is making is a more common decision than anyone selling AI services likes to admit.

Fathom — Fine-Tune A Model Version 2.0 alpha Released 2026-04-15. Next revision will be dated. The numbers are real. The methodology is open. The failures are documented. # Chapter 1: Economics — When Local Beats Cloud and By How Much

Priya's calendar invite from her CFO says "AI costs — 15 min Thursday." That is the meeting this chapter prepares you for.

By the end of it you will have: a six-rule decision framework, the crossover math for your specific volume, and a one-page memo template you can screenshot into the CFO's inbox twenty minutes before the meeting. The framework is durable. The specific prices are dated April 2026 and will go stale — we've built a live calculator (see Appendix) that pulls current GPU rental rates so you can refresh the numbers whenever you need them.

Currency note: Cloud API prices are quoted in USD because that is how providers bill. Hardware, salaries, and compliance costs are in GBP. Where a table mixes both, we state an approximate conversion rate of £1 = $1.25 at the top. We will not pretend this complexity away — it is part of the real cost picture.

Cloud API Pricing (April 2026)

Model	Input ($/1M tokens$)	$Output$ (/1M tokens)	Context
GPT-4o	$2.50	$10.00	128K
GPT-4o-mini	$0.15	$0.60	128K
Claude Sonnet 4	$3.00	$15.00	200K
Claude Haiku 3.5	$0.80	$4.00	200K
Gemini 2.0 Flash	$0.10	$0.40	1M
Gemini 2.5 Pro	$1.25	$10.00	1M

Every provider charges 3–5× more for output tokens than input tokens. An application that generates long responses (code, reports, summaries) pays dramatically more than one that extracts short answers from long documents.

Priya's situation: her fintech uses Claude Sonnet 4 for customer-support triage and compliance-document summarisation. Roughly 80% input, 20% output. That is a blended rate of $5.40 per million tokens. At 2M tokens per day, her monthly bill is roughly **$324/month** (~£260/month). At 8M tokens per day (where she is headed), it is **$1,296/month** (~£1,037/month) — and climbing 15% month-on-month as usage spreads.

Batch API discounts

All major providers offer ~50% off for asynchronous batch processing with 24-hour turnaround. Anthropic's prompt-cache discount is more powerful: 90% off for cache hits. A system prompt reused across thousands of calls reduces effective input cost by 70–80%.

If Priya's team is not using batch or caching, the first step is not local models — it is fixing the API billing. This one change can cut the bill 40–60% before touching hardware. We flag this because most "local LLM" conversations skip it, and it is the cheapest intervention available.

The Three Scenarios

Every volume level has a clear winner. The interesting zone is narrow.

Scenario A: 100K tokens/day (small team, 5 users)

Approach	Monthly cost
Claude Sonnet 4 (cloud)	**$16/month** (£13)
RTX 4090 local (amortised over 3 years + electricity)	**~£170–430/month**

Cloud wins by 10–30×. Hardware sits idle 20+ hours a day. There is no economic case for local at this volume unless a regulatory mandate forces data on-premises. Even then, a cheap VPS running a 3B model for £30/month probably beats owning a GPU.

Priya's takeaway: if your team is here, stop reading. Stay on the API. Come back when volume crosses 1M/day.

Scenario B: 1M tokens/day (moderate use, 10–30 users)

Approach	Monthly cost
Claude Sonnet 4	**$162/month** (£130)
Claude with caching (60% cache hit rate)	**$97/month** (£78)
Gemini 2.0 Flash (cheapest frontier)	**$7.50/month** (£6)
RTX 4090 local (amortised + ops)	**~£170–430/month**

Cloud is still cheaper by 2–5× on pure cost. At this volume, the decision is about privacy, latency, or capability — not money. If Priya's compliance team is not asking questions and the bill is manageable, cloud is the right answer.

But note Gemini Flash at $7.50/month. If your workload tolerates Google's data terms and the quality delta is acceptable, switching from Claude Sonnet to Gemini Flash saves 95% with zero hardware. This is the option most local-LLM advocates forget to mention.

Scenario C: 10M tokens/day (production workload, 50+ users or pipeline)

Approach	Monthly cost	Quality	Privacy
Claude Sonnet 4 (full price)	**$1,620** (£1,296)	Frontier	Cloud-hosted
Claude (batch, 50% off)	**$810** (£648)	Frontier	Cloud-hosted
2× RTX 4090 + base 8B model	~**£350–700**	Good (8/10 on our tasks)	On-premises
H100 + Llama 70B	~**£1,700**	Near-frontier	On-premises
Gemini Flash	**$75** (£60)	Very good	Cloud-hosted

The crossover for local vs premium cloud (full price) is roughly 8–9M tokens/day. With batch discounts, the crossover pushes to 15–18M. With Gemini Flash as the comparison, local never wins on pure cost — Google is subsidising inference.

Important note on the hardware costs above: The 2× RTX 4090 figure is our estimate based on a 3-year amortisation of ~£3,200 purchase cost plus electricity at UK commercial rates (~14p/kWh, 900W sustained). We have NOT measured a 4090's inference speed ourselves — Werner's verified measurement is 106 tok/s for Qwen-2.5-7B on a significantly weaker RTX 2070. A 4090 should be meaningfully faster, but we will not quote a specific number we have not measured. Community benchmarks suggest 110–130 tok/s for 8B Q4_K_M on a single 4090.

The Crossover Table

Daily volume	Cloud cost/mo (Sonnet 4, full)	Local cost/mo (2× RTX 4090)	Winner
100K	~£13	~£170–430	Cloud
1M	~£130	~£170–430	Cloud
10M	~£1,296	~£350–700	**Local catches up**
30M+	~£3,888+	~£350–700	**Local wins by 5×+**

Local hardware cost is roughly fixed once amortised. Cloud cost scales linearly with volume. The lines cross somewhere between 5M and 10M tokens/day for premium cloud, and never cross if Gemini Flash is acceptable.

The Privacy Premium (Why Regulated Industries Flip the Math)

For Priya, the API bill is not the only cost of staying on the cloud. Her compliance team is asking because the compliance team's budget is also a cost of the cloud decision.

What regulated UK mid-market companies actually pay to keep AI data in the cloud:

Compliance line item	Annual cost (GBP)
GDPR DPO + programme	£60,000–£150,000
Per-system DPIA (data protection impact assessment)	£5,000–£15,000
Breach risk (risk-adjusted annual expectation)	£10,000–£40,000
Total compliance overhead	**£50,000–£120,000/year**

Add this to the cloud API bill before comparing. For Priya's fintech at £260/month in Claude API, the effective cloud cost with compliance overhead is closer to **£4,400–£10,260/month**. Against that number, a local setup at £350–700/month is the obvious choice at any volume.

This is why the decision framework below has a separate rule for regulated industries. If you have GDPR, HIPAA, FCA, or DSPT exposure, the economics of local vs cloud are not comparable on API cost alone. The compliance premium makes local viable from day one, regardless of volume.

The compliance officer does not care that Claude is SOC 2 certified. She cares that privileged data left the building, crossed the Atlantic, was processed on a GPU in Virginia, and came back as a summary paragraph — and that nobody can tell her, on paper, what happened to the copy in between. The answer she wants is boring and physical: the model sits on this machine, the machine sits in this room, nothing else touches it. That answer is worth £50,000–£120,000/year in compliance cost avoided.

Hybrid Routing (The 70/30 Compromise)

Most workloads are not all-or-nothing. A routing classifier (typically a 1B model running on CPU at <5ms per decision) splits queries into "easy" (sent to the local model) and "hard" (sent to the frontier API). The local model handles volume; the frontier handles edge cases.

```
Request → Routing classifier (1B, CPU, <5ms)
    → Easy (65-75% of queries) → Local 8B model
    → Hard (25-35% of queries) → Claude/GPT frontier
```

At 10M tokens/day, 70/30 split

Component	Monthly cost
Local model (70% of traffic)	~£500
Cloud frontier (30% of traffic)	~£390
Routing classifier	~£12
Hybrid total	**~£900**
Cloud-only (Sonnet 4)	**~£1,296**
Monthly saving	**~£396 (31%)**

At an 80/20 split (more aggressive local routing): saving rises to ~£610/month (47%).

The sweet spot is 65–75% local. More aggressive routing requires higher local-model quality, because the queries you route locally are now harder. Monitor misrouting by sampling the local model's outputs weekly against the frontier's answers for the same queries.

The Six Rules (Priya's Decision Framework)

This is the framework you take into Thursday's meeting. Memorise it or screenshot it.

1. **Under 1M tokens/day, no regulation:** Stay on cloud APIs. Optimise billing (batch, caching) first. Do not buy hardware.
2. **1–10M tokens/day, no regulation:** Consider hybrid routing (70% local, 30% frontier). Fine-tune only if a specific task exceeds 5M tokens/day and the base model is not accurate enough.
3. **Any volume with GDPR/HIPAA/FCA/DSPT:** Local is the defensible choice. The compliance premium (£50K–£120K/year) makes local economically viable from day one regardless of token volume.
4. **Above 10M tokens/day:** Two consumer GPUs and a fine-tuned 8B model beat Claude on pure cost by month 3. This is the volume where every week on cloud is money set on fire.
5. **Above 30M tokens/day:** Local at every tier. Economics are overwhelming. The only question is which hardware.
6. **GPU ownership:** Rent until you use it more than 5 hours/day consistently. Below 3 hours/day, always rent. Between 3–5, run the electricity math for your specific site. For H100-class hardware, rent indefinitely — the break-even on a £25,000 card is 450+ hours/month.

Priya's One-Page CFO Memo

```
TO:      CFO
FROM:    Priya (Head of Engineering)
RE:      AI infrastructure costs - options and recommendation
DATE:    [Thursday]

SITUATION
Our Claude API bill is £[X]/month and growing ~15% month-on-month.
Compliance has flagged data residency concerns. Current posture: all
prompts processed on Anthropic's US infrastructure under our DPA.

THREE OPTIONS

1. OPTIMISE CLOUD (lowest effort, quickest win)
   - Enable batch API + prompt caching → estimated 40-60% bill reduction
   - Timeline: 1 week
   - Cost: nil
   - Does NOT resolve compliance concern
```

```
2. HYBRID LOCAL + CLOUD (recommended)
   - Route 70% of queries to a local model on owned/rented hardware
   - Keep 30% on Claude for complex queries
   - Estimated monthly saving: £[X] (31-47% of current bill)
   - Resolves compliance for 70% of data flow
   - Hardware cost: £[X] one-off or £[X]/month rental
   - Timeline: 4-6 weeks to production

3. FULL LOCAL (maximum privacy, higher complexity)
   - All inference on owned/rented hardware
   - Estimated monthly cost: £[X] (vs £[X] current)
   - Fully resolves compliance concern
   - Requires fine-tuning for task-specific quality (additional £3,500-£9,500 one-off)
   - Timeline: 6-10 weeks to production

RECOMMENDATION
Option 1 immediately (this week). Option 2 in parallel (4-6 weeks).
Revisit Option 3 only if compliance requires full data residency.

RISK
Cloud API prices have fallen 40-60% per year since 2023. The cost
advantage of local may narrow. Privacy and compliance arguments are
volume-independent and do not erode with price changes.
```

This memo is a template. Fill in your numbers using the crossover table above or the live calculator in the appendix. The structure — situation, three options, recommendation, risk — is designed for a finance audience that wants to make a decision, not read a technical paper.

What This Chapter Does Not Cover

- Hardware selection (Chapter 2)
- Which model to pick (Chapter 3)
- Whether to fine-tune and how (Chapter 4)
- How to prove it works before shipping (Chapter 4, evaluation section)
- How to price and sell this capability (Chapter 6)

If you are Priya, you now have enough to survive Thursday. The next chapter is for the following Monday, when the CFO says "OK, do it" and you need to know what to buy.

Prices dated April 2026. Cloud rates from provider pricing pages. Hardware costs from UK retail and Vast.ai marketplace rates. Compliance cost ranges from GDPR programme benchmarks published by UK DPO consultancies 2024–2025. All crossover calculations assume sustained daily volume; bursty workloads shift the break-even toward cloud. Use the live calculator for current numbers. # Chapter 2: Hardware — Rent, Buy, or Use What You Have

Priya's CFO said "OK, do it." Now she needs to know what to buy — or whether to buy anything

at all.

This chapter is a buyer's guide, not a spec sheet. Every recommendation answers one question: **for Priya's workload and budget, what is the least expensive path to running a local model at acceptable speed?** We start with the one constraint that eliminates 80% of the decision tree, then work through four hardware tiers.

The One Constraint: VRAM

VRAM is the hard gate. A model must fit in GPU memory or inference speed collapses from "usable" to "painful." There is no gradual degradation — the cliff is sharp.

Model size	VRAM required (Q4_K_M)	VRAM required (Q8_0)
7B / 8B	~5 GB	~9 GB
14B	~9 GB	~16 GB
32B	~20 GB	~36 GB
70B	~42 GB	~75 GB

If your GPU has less VRAM than the Q4_K_M column, the model will not run at interactive speeds. Full stop. The rest of the spec (clock speed, memory bandwidth, tensor cores) matters for *how fast* — but VRAM determines *whether*.

Priya's situation: she wants to run an 8B model for customer-support triage. That needs ~5 GB VRAM. Almost any GPU from the last six years can do this. The question for her is speed and cost, not feasibility.

Tier 1: Consumer Desktop GPUs

NVIDIA dominates because every major inference stack (llama.cpp, Ollama, vLLM) is CUDA-first. AMD ROCm support exists but is patchy — avoid it for production unless you have a specific reason and an engineer who enjoys pain.

What we have measured ourselves

GPU	Model	Measured tok/s	Quality	Source
RTX 2070 8GB	Qwen-2.5-7B Q4_K_M	**106 tok/s**	8/10	Werner, Sessions 7–15
RTX 2070 8GB	gemma3:4b Q4_K_M	**83 tok/s** (warm)	7/10	Werner, Session 15

These are our only verified GPU inference measurements. The RTX 2070 is a 2018 card that costs £140–200 on eBay. At 106 tok/s on an 8B model, it is proof that viable local inference does not require expensive hardware.

What we have NOT measured (community benchmarks, use with caution)

The following speeds are sourced from community benchmarks (llama.cpp discussions, Reddit r/LocalLLaMA) and manufacturer specs. We have not verified them on our own hardware. Treat them as estimates — real-world performance varies by 20–40% depending on build flags, thermal throttling, concurrent processes, and quantisation format.

GPU	Buy price (approx)	VRAM	Est. tok/s (8B Q4)	Best for
RTX 3060 12GB	£200–280 used	12 GB	~30–40	Budget 14B (barely fits)
RTX 4060 Ti 16GB	£380–430 new	16 GB	~35–45	Comfortable 14B
RTX 4070 Ti Super 16GB	£600–680 new	16 GB	~65–80	Sweet spot for 8B–14B daily use
RTX 3090 24GB	£700–900 used	24 GB	~70–90	Budget 32B (24GB headroom)
RTX 4090 24GB	£1,800–2,200 new	24 GB	~110–130	The consumer reference card
RTX 5090 32GB	£2,000–2,800 new	32 GB	~160–200	32B comfortably; the new baseline

The RTX 4060 Ti 16GB has a bandwidth trap. Its 128-bit memory bus (288 GB/s) is narrower than the older RTX 3060's 192-bit bus (360 GB/s). NVIDIA offset this with more L2 cache, but the speed gap is real for sustained inference. If you need 16GB VRAM, the 4070 Ti Super at £600 is a better buy.

Two RTX 4090s (48GB combined) via tensor parallelism can run a 70B model entirely in VRAM at ~25–35 tok/s. Cost: £3,600–4,400. Often cheaper than a used A6000 Ada, but adds multi-GPU complexity.

When consumer GPUs are right

- Solo developers and small teams (1–5 concurrent users)
- Models 32B with a single 5090, or 14B with a 4070 Ti Super
- You own the machine and run it 6 hours/day (below that, rent — see Chapter 1)
- Data residency requirements mean cloud is off the table

When they are wrong

- Multi-user serving (10 concurrent) — consumer GPUs lack the memory bandwidth for batched inference
- 70B models at interactive speed — needs 48GB+ VRAM
- Production uptime SLAs — consumer cards are not designed for 24/7 operation

---

Tier 2: Workstation and Server GPUs

RTX A6000 Ada 48GB

Buy: ~£6,500–7,500 new, ~£4,500–5,500 used | **VRAM:** 48 GB ECC | **Bandwidth:** 960 GB/s

48GB is the critical threshold: 70B Q4_K_M (~42GB) fits entirely in VRAM. No offloading, no multi-GPU coordination.

Community-reported speeds: 8B ~95–110 tok/s, 70B ~22–30 tok/s (fully in VRAM).

Verdict: The single best card for 70B self-hosted inference. Expensive but eliminates two-GPU complexity. If you know you need 70B, this is the answer.

H100 80GB

Buy: ~£25,000–35,000 | **Rent:** $1.80–3.50/hr (Vast/RunPod/Lambda)

80GB enables 70B FP16 in a single card. 2–4× faster than A100 for production workloads with TensorRT-LLM.

Verdict: Do not buy one. The rental market is liquid. Rent until your monthly spend exceeds ~£2,500 consistently, then revisit. The break-even on a £30,000 card at $3.29/hr rental is 380+ days of 24/7 use.

Tier 3: Apple Silicon

Apple's unified memory architecture is under-discussed for LLM inference. The CPU, GPU, and Neural Engine share one memory pool with ~400+ GB/s bandwidth. No PCIe bottleneck. Silent operation. 60–80W power draw.

What we have measured ourselves

Device	Model	Measured tok/s	Source
MacBook Air M5 16GB	Qwen3-8B Q4_K_M	**27 tok/s**	Werner, Session 13

The M5 measurement caveat

27 tok/s on a MacBook Air M5 is significantly below community reports for M4 Max and M4 Ultra on 8B models (85–140 tok/s claimed). Possible explanations: the Air has lower sustained GPU bandwidth than the Pro/Studio, thermal throttling on a fanless chassis, or the M5 chip has different performance characteristics than M4 for this workload. **We have not verified the higher claims and will not repeat them as fact.**

When Apple Silicon is right

- **70B inference at a reasonable price.** An M4 Ultra Mac Studio with 192GB unified memory (~£6,000) is the quietest, most power-efficient way to run a 70B model at (community-reported) 20–28 tok/s. An RTX 4090 cannot fit 70B without painful CPU offloading.

- **Development and testing.** Engineers who want to iterate on prompts against a local model without spinning up a cloud instance.
- **Low power draw matters.** 60–80W vs 450–575W for consumer NVIDIA. At UK commercial electricity rates, that is a £340/year difference.

When Apple Silicon is wrong

- **Multi-GPU scaling.** You cannot add a second Apple GPU.
- **CUDA ecosystem.** Many training and fine-tuning tools require CUDA. MLX is improving but is not at parity.
- **14B on 16GB RAM swaps to disk and collapses to ~5 tok/s.** Always verify the customer's Mac RAM before quoting 14B performance. 32GB+ handles 14B comfortably. 64GB+ handles 32B.

Tier 4: GPU Cloud Rental

When to rent

- Any workload you run less than 5 hours/day
- Experimentation and fine-tuning (checkpointable; interruptions are OK)
- Burst capacity for production spikes
- H100-class hardware (almost always rent, never buy)

When to own

- Any workload you run more than 5 hours/day consistently
- RTX 4090 class: break-even with rental at ~7.5 months of 8hrs/day use
- Consumer hardware where the purchase price is 1–3 months of rental

Provider landscape (April 2026)

GPU rental pricing is volatile — rates shift weekly on marketplace platforms. Rather than printing specific hourly rates that will be stale when you read this, we built a **live pricing tool** that pulls current Vast.ai rates for 44 GPU types. Run `python3 /home/a/Projects/fathom/book/server.py` and visit the pricing tab, or use the numbers from the interactive version of this book.

General provider positioning:

Provider	Cheapest?	Most reliable?	EU data residency?
Vast.ai	Yes (marketplace)	Variable (filter >95% uptime, "verified" tier)	Some hosts
RunPod Community	Similar to Vast	Variable	Some hosts
RunPod Secure	40–60% more than Community	Datacenter-grade, SLAs	Limited
Lambda Labs	Premium	Clean UX, predictable	No

Provider	Cheapest?	Most reliable?	EU data residency?
Hetzner	Mid-range	German datacenters	**Yes — GDPR-native**

For Priya's regulated fintech: Hetzner is the default answer. German-hosted hardware, GDPR-compliant datacenters, predictable pricing. The GPU lineup is narrower than marketplace providers, but "we run inference on a Hetzner server in Frankfurt" is the answer her compliance officer wants to hear.

For experimentation and fine-tuning: Vast.ai is cheapest. Filter by uptime score >95% and "verified" tier. Never use marketplace GPUs for production serving — interruption rates are 15–25%.

Edge and Embedded (Brief)

NVIDIA Jetson Orin series: 8–64GB shared memory, 7–60W, £300–1,500. Practical for 1B–3B models at <25W. Nothing above 13B is interactive.

Raspberry Pi 5: CPU-only, 7B at ~0.5–1.5 tok/s. Unusable for interactive inference. Fine for embeddings.

When edge is the answer: physical deployment (robot, kiosk, controller), no network available, sub-5W power budget, or regulatory requirement that the model never leaves the device. For everything else, a Mac Mini M4 (~£599, 16GB) beats Jetson on every dimension except power draw.

Priya's Decision

Priya runs an 8B model for customer-support triage at her 180-person fintech. She needs: - GDPR-compliant hosting (rules out Vast.ai marketplace; points to Hetzner or owned hardware) - Single-user initially, scaling to 5–10 concurrent - Budget: under £1,000 upfront or under £200/month rental

Her options:

Option	Cost	Speed (est.)	Privacy	Complexity
RTX 4070 Ti Super (owned)	£650 one-off + electricity	~65–80 tok/s	On-premises	Low — one card, one machine
RTX 3090 (used, owned)	£700–900 one-off + electricity	~70–90 tok/s	On-premises	Low
Hetzner GPU server (rented)	~£150–300/month	Varies by GPU	EU datacenter (Frankfurt)	Medium — remote server management
Mac Mini M4 (owned)	£599 one-off	~27+ tok/s (est. for 8B)	On-premises	Low, silent

Recommendation for Priya: buy a used RTX 3090 (£700–900), install it in an existing office server or desktop. 24GB VRAM gives headroom for 14B later. 70–90 tok/s estimated on 8B is more than adequate for triage. Total ongoing cost: electricity only (~£15/month at UK commercial rates for 8hrs/day). If she cannot install a GPU on-premises, Hetzner rental is the fallback.

This is not the optimal choice. It is the **cheapest defensible choice** that resolves both the cost concern (one-off £800, no ongoing API bill) and the compliance concern (data never leaves the office). Priya can upgrade later if volume demands it. The RTX 3090 is a £800 decision, not a £8,000 one.

The Cheat Sheet

I need…	Buy this	Or rent this
Fast 8B, budget-first	RTX 3060 12GB (~£250 used)	Vast.ai cheapest available
Fast 8B–14B, daily use	RTX 4070 Ti Super (~£650)	Hetzner GPU server
Fast 32B, daily use	RTX 5090 (~£2,600)	RunPod Secure L40S
70B in full VRAM	A6000 Ada (~£5K used) or 2× 5090	H100 on RunPod ~$2.50/hr
70B, silent + low power	Mac Studio M4 Ultra (~£6K)	—
EU data residency	Hetzner dedicated GPU	Hetzner/OVH cloud GPU
Experimentation	—	Vast.ai (cheapest)
Embeddings only	Any £20/mo VPS	Any cheap CPU VPS

Two opinions

Rent first, buy with conviction. Renting for 3 months while you establish actual usage patterns is almost always correct. The break-even math only works if your usage estimate is right — and it rarely is in month one.

The RTX 5090 changes the game for 32B. At 32GB VRAM and ~1,792 GB/s bandwidth, a single consumer card comfortably runs 32B models that previously required server-class hardware. If you are buying in 2026 and plan to stay on 14B–32B, this is the card.

All prices April 2026 estimates. GPU prices are volatile; check current retail and marketplace rates before purchasing. Community benchmark speeds are sourced from r/LocalLLaMA and llama.cpp discussions and may vary 20–40% from your specific configuration. Our own measurements (RTX 2070, M5 MacBook Air) are marked explicitly throughout. # Chapter 3: Model Selection — Which Model, Which Size, Which Format

Priya has hardware. Now she needs a model. This chapter gives her the decision in three steps: pick a size tier, pick a model family, pick a quantisation format. In that order — because VRAM determines size, size determines family, and family determines format options.

Step 1: Pick the Size Tier

Model size is not a quality slider. It is a capability threshold. Each tier unlocks a category of tasks that smaller tiers cannot do reliably. You do not need a bigger model — you need the smallest model that handles your task.

Tier	VRAM (Q4)	Representative models	Can do well	Cannot do reliably
0.5–2B	0.5–1 GB	Qwen2.5-0.5B, Llama-3.2-1B	Binary classification, intent routing, keyword extraction, language detection	Multi-hop reasoning, summarisation, code gen
3–4B	2–3 GB	gemma3:4b, Phi-4-mini 3.8B	Short-doc summarisation, basic QA, code completion, email drafting	Complex reasoning, novel code generation
7–8B	4–5 GB	Qwen-2.5-7B, Llama-3.1-8B	Most NLP tasks, good code gen, SQL, JSON, domain chat	Complex multi-step reasoning, creative long-form
13–14B	7–9 GB	Qwen-2.5-14B, DeepSeek-R1-14B	Multi-step reasoning, legal/technical summarisation, complex planning	Rivalling frontier on open-ended tasks
32–35B	16–18 GB	Qwen-2.5-32B, DeepSeek-R1-32B	Near-GPT-4 on narrow benchmarks, strong math, complex code	Open-ended creative
70B+	35–40 GB	Llama-3.1-70B, Qwen-2.5-72B	Broad generalist, near-frontier on most tasks	Matching frontier on creative/agentic

The important finding from our own testing

Werner's quality benchmark (10 questions covering math, logic, code, regex, concepts) produced a result that surprised us:

Model	Size	Quality score	Speed	Hardware
Claude Sonnet 4.6	Frontier	**10/10**	instant	Cloud
gemma3:4b	4B	**7/10**	68 tok/s	RTX 2070

Model	Size	Quality score	Speed	Hardware
Qwen-2.5-7B	7B	**6.5/10**	24 tok/s	RTX 2070

The 4B model beat the 7B model. Smaller, faster, and higher quality on our test suite.

This does not mean 4B is universally better than 7B. It means model family and architecture matter more than raw parameter count. gemma3 benefits from Google's distillation pipeline; Qwen-2.5-7B is a general-purpose model that excels at different tasks (structured JSON, code, classification). The right model depends on the task, not the parameter count.

Priya's takeaway: do not assume bigger is better. Benchmark your specific task on two or three models at different sizes before committing. A 15-minute test on your actual data is worth more than any leaderboard.

The 7–8B tier is where most production deploys should start

~80% of production fine-tuned deployments should begin at 7–8B. Reasons: - Fits on the cheapest viable GPUs (RTX 3060 12GB and up) - Quality is good enough for most classification, extraction, and structured-output tasks - Fine-tuning at this size is fast and cheap (~1–2 hours on a single RTX 4090) - If 7–8B is not good enough after prompt engineering and fine-tuning, go to 14B — do not skip to 70B

The exception: **reasoning-heavy tasks.** The quality jump from 7B to 14B on multi-step reasoning is the largest gain-per-dollar in the lineup. DeepSeek-R1-14B vs R1-7B on reasoning benchmarks is a 14+ point improvement. Start at 14B for math, planning, debugging, and multi-step SQL.

Step 2: Pick the Model Family

The three families that matter in 2026

Qwen 2.5 / Qwen 3 (Alibaba) — the default for most new deployments.

Licensed Apache 2.0 — the decisive commercial advantage over Llama. Per-parameter efficiency leads the field: Qwen-2.5-7B consistently outperforms Llama-3.1-8B on classification, extraction, and code tasks in our testing (8/10 vs ~7/10 on our internal eval). Multilingual capability is strong. Qwen-2.5-Coder variants are the best open-source code models at 7B.

Qwen 3 adds MoE (Mixture of Experts) variants: Qwen3-235B-A22B activates ~22B parameters from 235B total — near-70B quality at 22B compute cost. Qwen3 also supports explicit reasoning toggle via system prompt (`/think` and `/no_think`), which is critical for mixed workloads.

Llama 3.x (Meta) — the safe, widely-supported default.

Most deployed open model family. Broad tooling support. Licensing permits commercial use but restricts services with >700M MAUs — not Apache 2.0, cannot be relicensed.

Weaknesses: multilingual lags Qwen on non-English tasks. Long-context retrieval degrades beyond ~32K tokens in practice despite 128K nominal window. Code generation consistently beaten by Qwen-2.5-Coder at equivalent sizes.

DeepSeek R1 (distilled variants) — the reasoning specialist.

Changed the reasoning-model conversation in 2025. Distilled variants at 1.5B through 70B. The 14B distill is the sweet spot for reasoning tasks — beats many larger non-reasoning models on math and multi-step logic.

Use when: the task requires chain-of-thought reasoning (math, planning, debugging, complex SQL). Do not use when: latency matters more than accuracy — reasoning models generate 1,000+ thinking tokens before the first response word.

Others worth knowing

- **Gemma 2/3 (Google):** gemma3:4b is the best sub-5B model for edge and constrained deployments. Licensing is more restrictive than Apache 2.0 — review terms before production.
- **Phi-3/Phi-4 (Microsoft):** Strong reasoning in tiny models (Phi-4-mini under 4B). English-only. Weak on creative and multilingual tasks.
- **Mistral/Mixtral:** MoE architecture gives quality-per-active-parameter, but all experts must be resident in VRAM (Mixtral 8x7B needs ~26GB despite activating only ~13B). Better suited for teams with VRAM headroom.
- **Command R (Cohere):** Built for RAG with native grounding. Justified for high-stakes information retrieval where citation fidelity matters. Otherwise, Qwen-2.5-14B + RAG is sufficient.

What died in 2025

Falcon, Llama 2, MPT, Vicuna/Alpaca, BLOOM, GPT-J/GPT-NeoX, StableLM/StableCode. All outclassed by the current generation. If your codebase still references any of these, upgrade.

Step 3: Pick the Quantisation Format

Quantisation compresses model weights from full precision (FP16/BF16, 2 bytes per parameter) to lower precision (4-bit, 0.5 bytes per parameter). A 7B model shrinks from ~14GB to ~4–5GB. This is what makes consumer GPUs viable.

GGUF is the delivery format

GGUF (GPT-Generated Unified Format) is the standard for local inference via llama.cpp, Ollama, and LM Studio. Every model we deliver ships as a GGUF file. It is the MP3 of local LLMs — not the highest quality, but the most compatible and most practical.

The quantisation ladder

Quant	Bits/weight	7B size	Quality vs FP16	When to use
Q2_K	~2.5	~3 GB	Noticeable degradation	Emergency VRAM constraints only

Quant	Bits/weight	7B size	Quality vs FP16	When to use
Q3_K_M	~3.5	~3.5 GB	Slight degradation	Tight VRAM, acceptable quality trade
Q4_K_M	~4.5	**~4.5 GB**	**Negligible for most tasks**	**Default. Start here.**
Q5_K_M	~5.5	~5.5 GB	Very close to FP16	When quality ceiling matters
Q6_K	~6.5	~6.5 GB	Near-lossless	VRAM permits, maximum quality
Q8_0	8.0	~8 GB	Effectively lossless	Reference/eval comparisons
FP16	16.0	~14 GB	Baseline	Fine-tuning and training only

Q4_K_M is the right default for almost all production deployments. Modern models (Qwen-2.5, Llama-3, Gemma-3) show perplexity increase of ~0.5–2% at Q4_K_M compared to FP16. For classification, extraction, and structured output tasks, this difference is unmeasurable in practice.

The quality cliff warning

Reasoning models and MoE models are more sensitive to aggressive quantisation than dense models. DeepSeek-R1-distill at Q4_K_M loses noticeably more quality than Qwen-2.5 at the same quant level. Mixtral 8x7B has similar fragility. For reasoning-heavy workloads, start at Q5_K_M or Q6_K and benchmark the difference.

KV cache — the hidden VRAM consumer

During inference, the model stores attention state for every token in context. This KV (key-value) cache grows with context length and can consume significant VRAM on top of the model weights.

Important: Modern models use Grouped-Query Attention (GQA), which dramatically reduces KV cache size compared to older full-attention models. Llama-3.1-8B uses 8 KV heads (not 32) — its KV cache at 32K context in FP16 is approximately **2.1 GB**, not the 8.6 GB that a naive calculation with 32 heads would suggest. Always check the model's actual `num_kv_heads` before budgeting VRAM.

Model	KV heads	KV cache at 32K (FP16)	With Q8 KV
Llama-3.1-8B	8 (GQA)	~2.1 GB	~1.1 GB
Qwen-2.5-7B	4 (GQA)	~1.1 GB	~0.5 GB
Mixtral 8x7B	8 (GQA)	~2.1 GB	~1.1 GB

Practical rule: budget VRAM as model weights (from quant table) + KV cache (check the model card for `num_kv_heads`) + 1 GB buffer. If this total exceeds your GPU's VRAM, drop to a smaller quant or a smaller model.

Inference Stacks (How to Actually Run the Model)

Ollama — the right default for getting started

Wraps llama.cpp in a clean CLI and API. `ollama run qwen2.5:7b` is one command. Ships an OpenAI-compatible API endpoint at `localhost:11434`. Model management, automatic quant selection, and GPU detection are handled.

Use Ollama when: you are running a single model for a small team, you want the fastest path from "nothing installed" to "model running," or you are evaluating models interactively.

vLLM — the right default for production serving

PagedAttention enables efficient memory sharing across concurrent requests. Continuous batching handles variable-length inputs without padding waste. 3–8× throughput improvement over Ollama at 10+ concurrent users.

Use vLLM when: you are serving more than 5 concurrent users, need maximised throughput, or are deploying behind a load balancer.

Caveat (not in most guides): vLLM's PagedAttention adds per-token latency overhead. For single-user, low-concurrency workloads, llama.cpp/Ollama is faster on both time-to-first-token and generation speed. vLLM wins at concurrency, not at single-user speed.

Speculative decoding — the 2026 technique most guides omit

A small "draft" model (e.g., 1.5B) proposes several tokens at once; the large model verifies them in a single forward pass. When the draft model's predictions are good (which they often are for common patterns), throughput increases 1.5–3×.

Supported in vLLM and TensorRT-LLM. Not yet in Ollama's default pipeline. For high-throughput production serving, speculative decoding is a significant win that most local-LLM guides written before mid-2025 do not mention.

The Task-to-Model Quick Reference

Priya's task	Model	Size	Why
Customer-support triage (8 categories)	gemma3:4b or Qwen-2.5-7B	4–7B	Fast, accurate on classification. Test both — our benchmark showed 4B can beat 7B.
Compliance-document summarisation	Qwen-2.5-14B	14B	Technical summarisation needs the 14B quality jump

Priya's task	Model	Size	Why
Structured JSON extraction	Any 7B+ instruct	7B+	Use grammar-constrained decoding (llama.cpp GBNF) for >99% valid JSON
Code review bot (internal framework)	Qwen-2.5-Coder-7B + LoRA	7B	Specialisation matters more than size for code
Fraud alert reasoning	DeepSeek-R1-distill-14B	14B	Multi-step reasoning justifies reasoning model
Embeddings for RAG retrieval	bge-m3 or nomic-embed-text	100–300M	Encoder-only. Do NOT use a generative model for embeddings.

The embeddings point is a common category error. Generative models (GPT, Qwen, Llama) are not embedding models. Dedicated encoder-only models (bge-m3, nomic-embed) run on CPU, return vectors in milliseconds, and outperform generative models at retrieval. If someone suggests using Qwen for embeddings, they are confused about the task type.

Priya's Decision

Priya needs a model for customer-support triage (8 categories) running on the RTX 3090 she just bought.

Our recommendation: start with **gemma3:4b Q4_K_M** via Ollama. Werner's benchmark showed it scoring 7/10 on our quality test at 68 tok/s — beating the larger Qwen-2.5-7B (6.5/10, 24 tok/s on the same test). On a 3090, the 4B model will be even faster.

Run it for one week on Priya's real support tickets. If accuracy is above her threshold (she should define this before testing — see Chapter 4), she is done. Hardware cost: already paid. Model cost: free. Ongoing API cost: zero.

If accuracy is below threshold, try Qwen-2.5-7B on the same eval set. If both fail, the task may need fine-tuning (Chapter 4) or a 14B model.

Do not start at 14B because it "might be better." Start at the smallest model that might work, measure, and go up only with evidence.

Model capabilities and benchmark scores are a snapshot. New models release monthly. The decision framework (size tier → family → quant) is durable; the specific model recommendations will shift. Check the model comparison table in the interactive edition for current rankings. # Chapter 4: The Craft — Fine-Tuning and Evaluation as One Discipline

Priya's gemma3:4b model is running on her RTX 3090. It handles 70% of support tickets well. The other 30% — the ones that require understanding her company's specific product terminology, her

internal escalation categories, her compliance-flagging rules — it gets wrong.

She is considering fine-tuning. This chapter will help her decide whether to do it, and if so, how to avoid the mistake we made.

The Hierarchy Nobody Follows

The correct order is: prompt engineering first, RAG second, fine-tuning third. This is not conservatism. It is economics.

Approach	Cost	Time	When it's right
Prompt engineering	An afternoon	Hours	The model has the knowledge, just needs the right instructions
RAG (retrieval)	A week + a vector database	Days	The bottleneck is knowledge the model doesn't have
Fine-tuning	Data collection + compute + eval + maintenance	Weeks	The model needs persistent behavioural changes that prompts can't express

Priya should try prompt engineering first. Specifically: write a detailed system prompt with her 8 ticket categories, 3 examples per category, and explicit escalation rules. Run it against 200 real tickets. If accuracy crosses her threshold, she is done — no fine-tuning needed, no ongoing maintenance cost.

Our own experience validates this: Werner's auto-improved prompts hit **100% accuracy on our internal classification tasks** — better than the fine-tuned V2 adapter, which scored net-negative. Prompt engineering is not a consolation prize. It is frequently the correct answer.

When prompt engineering fails

Prompt engineering fails when: - The behaviour you need cannot be expressed in a prompt (rare but real — e.g., a specific output distribution that no instruction produces) - The prompt is so long it hurts latency or cost (>2,000 tokens of instructions on every call) - The task requires the model to "just know" your domain vocabulary without being told each time - You need consistency across thousands of calls that a slightly-rephrased prompt would break

When RAG fails

RAG (Retrieval-Augmented Generation) fails when: - The task requires implicit behavioural patterns, not explicit facts - The retriever is bad (wrong chunks returned = wrong answers regardless of model quality) - The knowledge base changes faster than you can re-embed

When fine-tuning is the right answer

Fine-tuning is justified when all three are true: 1. Prompt engineering genuinely fails at production scale (not "I tried for 10 minutes") 2. You have **500+ high-quality, task-specific examples** (below this, the fine-tune will overfit) 3. The task distribution is stable enough that a model trained today will still be useful in six months

What fine-tuning changes: output distribution, format and style, domain vocabulary, default behaviour on ambiguous inputs, response structure.

What fine-tuning does not change: fundamental reasoning capacity (7B does not become 70B), knowledge cutoff, context length limits, performance on tasks absent from training data.

The most common mistake: expecting a behavioural improvement to substitute for a capability improvement. **Fine-tuning amplifies existing capabilities. It does not conjure new ones.** If the base model cannot do the task at all with a perfect prompt, fine-tuning will not fix it. Go up a model size.

The Three Questions Before You Start

1. **Have I actually tried prompt engineering at production scale?** Not ten minutes. Real systematic iteration with a proper eval set.
2. **Do I have at least 500 high-quality, task-specific examples?** If no, the fine-tune will overfit and make things worse. We proved this ourselves — see Appendix A.
3. **Is the task distribution stable enough that a trained model will be useful in six months?** If chasing a rapidly moving target, fine-tuning leaves you permanently behind.

How Fine-Tuning Actually Works (LoRA / QLoRA)

The idea in one paragraph

A full fine-tune updates every weight in the model — 7 billion parameters for a 7B model. That requires massive memory and compute. LoRA (Low-Rank Adaptation) hypothesises that the *change* needed is low-rank: instead of updating a 4096×4096 weight matrix (16 million parameters), you train two small matrices that multiply to approximate the change — at rank 16, that is 2×(4096×16) = 131,072 parameters. A 128× reduction. Quality sacrifice for most tasks at 7–14B: within 1–3% of a full fine-tune.

QLoRA goes further: the frozen base model is loaded in 4-bit quantisation (NF4), and only the tiny LoRA adapter weights stay in full precision. A 7B model needs ~4–5GB instead of ~14GB. On a single RTX 4090, you can fine-tune a 14B model that would be intractable with full fine-tuning.

Full fine-tuning is justified only when: 100k+ examples, 8+ H100-class GPUs, and the task requires deep representational changes. This is 1% of commercial deployments. For everyone else, QLoRA is the default.

LoRA rank: how to choose

Rank (r)	When	Dataset size
r=8	Classification, simple extraction	<5k examples
r=16	General-purpose default	5k–50k examples
r=32	Complex generation, multi-turn dialogue	10k–50k examples with diversity
r=64+	Rarely justified. Diminishing returns.	50k+ examples

The trap we fell into: our V2 adapter used r=16 on only 74 examples. The rank was appropriate; the dataset was not. 74 examples for r=16 is like fitting a polynomial with 131,000 free parameters to 74 data points. The overfit was inevitable. See Appendix A.

Target modules

Standard: `q_proj, k_proj, v_proj, o_proj` (full attention). For richer tasks (code gen, vocabulary shift): also target `gate_proj, up_proj, down_proj` (feed-forward layers). Roughly doubles adapter parameters but produces meaningful quality gains on complex tasks.

Learning rate and alpha

Start with learning rate **1e-4 to 2e-4** for 7B QLoRA at r=16. Alpha = 2 × rank is one convention (alpha=32 for r=16), producing a scaling factor of 2.0. This is stable for most setups but is not universal — modern practice (rsLoRA) scales alpha by sqrt(rank) for larger ranks. If you are using r=32+, research rsLoRA before assuming the 2:1 rule holds.

If the model forgets base capabilities: lower alpha, lower learning rate, or add replay data (see below).

Training: The Practical Checklist

Data

Requirement	Minimum	Production
Classification (per class)	200–500 examples	2,000+
Named entity extraction	1,000–3,000	5,000+
Summarisation	2,000–10,000	10,000+
Domain instruction following	5,000–20,000	20,000+

Data format: ChatML for instruction/chat tasks. Match the format your base model was instruction-tuned on — check the model card. **Critical warning not in most guides:** fine-tuning from an `-instruct` model with a different chat template than it was trained on will silently corrupt the model. This is the #1 cause of bad fine-tunes in practice. Verify the template.

Synthetic data: useful for diverse phrasings, edge cases, and bootstrapping labels. Fails for: knowledge the teacher model lacks, adversarial examples, and factual verification. Use a frontier model for structure and format; verify content with domain expertise.

Held-out test set: 80/10/10 train/validation/test split. Test set must be locked before training. Never use the test set to select checkpoints — that is what the validation set is for.

Training hyperparameters

Parameter	7B QLoRA	14B QLoRA
Learning rate	1e-4 to 2e-4	5e-5 to 1e-4
Schedule	Cosine	Cosine
Warmup ratio	0.03–0.05	0.05
Effective batch size	32–64	16–32
Epochs	**1–2**	**1–2**
Gradient clipping	1.0 (not optional)	1.0

On epochs: the first version of this book recommended 3–5 epochs for 7B. That is the recipe that produced our V2 overfit. **1–2 epochs is correct for instruction fine-tunes on 5k+ examples.** 3+ epochs is only appropriate for very small datasets (<1k) with heavy regularisation and aggressive early stopping. If your validation loss bottoms before epoch 2, stop. Our V2's validation loss bottomed at epoch ~1.8 and went up from there. We trained to epoch 4. Don't do what we did.

The replay data trick

Include 2–5% of general instruction-following examples (from the base model's original training distribution) in your fine-tuning dataset. This prevents catastrophic forgetting — the model stays competent at general tasks while specialising on yours. We skipped this in V2 because "it's a narrow task." The model got worse at reading documents while getting better at producing our JSON schema. Classic mistake.

Training frameworks

- **Unsloth:** 2× throughput, 40–60% less VRAM via custom Triton kernels. Use when supported (Llama, Mistral, Qwen, Phi, Gemma).
- **HuggingFace PEFT + TRL:** Reference implementation. Broadest compatibility. Use when Unsloth doesn't support your model.
- **Axolotl:** Configuration-driven via YAML. Good for teams running multiple experiments.
- **OpenPipe / Together.ai:** Pay $50–150 for a hosted fine-tune. Use when: one-off model, <100M training tokens, no regulatory constraints on where data goes.

Infrastructure and cost

Model size	Hardware	Wall clock (10k examples, 2 epochs)	Cloud cost
7B QLoRA	Single RTX 4090	1–2h (Unsloth)	~£2–5

Model size	Hardware	Wall clock (10k examples, 2 epochs)	Cloud cost
14B QLoRA	A100 40–80GB	2–4h	~£5–15
32B QLoRA	A100 80GB	6–12h	~£15–40
70B QLoRA	2× A100 80GB (FSDP, NOT DDP)	18–36h	~£40–150

Critical correction from v1: the previous version of this book said "For QLoRA up to 70B on 4× A100 80GB, DDP is sufficient." This is wrong. DDP (Distributed Data Parallel) requires each GPU to hold the full model. 70B in 4-bit is ~40GB + optimizer states + activations, exceeding 80GB per GPU under realistic batch sizes. **Use FSDP (Fully Sharded Data Parallel) for 70B.** Anyone following the DDP advice would waste a GPU rental discovering this.

Evaluation: The Discipline That Makes Fine-Tuning Honest

The baseline rule

Run your base model on your task before writing a single training example. Measure: primary metric (accuracy, F1, exact match), latency, and error modes. This tells you three things: how far you need to go, whether you need to fine-tune at all, and what a realistic target looks like.

A base Qwen-2.5-7B at 85% on your classification task with a 92% target means you need a 7-point gain — achievable with prompt engineering alone, possibly. A base model at 55% with a 90% target means you need a 35-point gain — fine-tuning is justified, but check whether 14B gets closer before training.

The 10% threshold

A fine-tuned model should produce at least **10% relative improvement** on your primary metric over baseline. Below that, measurement noise makes it unclear whether you improved anything. Our V2 scored 33% vs 34% base — well within noise. We should have stopped, not shipped.

Eval set construction

Minimum viable: **500 examples.** Below that, confidence intervals are too wide for 5-point differences to be statistically meaningful. Stratify:

- 60% routine cases
- 20% edge cases (boundary conditions, unusual phrasing)
- 15% adversarial/stress (designed to make the model fail)
- 5% regression anchors (specific examples that failed in previous versions)

What to measure

Task type	Primary metric	Notes
Classification	Macro F1 (not accuracy — handles class imbalance)	Always generate a confusion matrix and read it
Extraction	Field-level F1 (not overall exact match)	Break down by field type — easy fields mask hard-field failures
Summarisation	ROUGE-L + BERTScore	Neither alone is sufficient
Code	pass@1 (execute in sandbox)	Log which test cases fail — patterns in failures reveal gaps
Open-ended	LLM-as-judge (frontier model scores against rubric)	Correlates 0.6–0.85 with human ratings depending on task and judge quality

LLM-as-judge caveats

Using a frontier model to evaluate your local model's output is efficient and broadly useful. But it has known failure modes: - **Position bias:** rate responses individually, not comparatively - **Verbosity bias:** include a conciseness dimension in the rubric - **Self-serving bias:** use a different model family as judge (e.g., Claude judging a Qwen fine-tune) - **Calibration drift:** include anchor examples every 50 evaluations

When human eval is non-negotiable: safety-critical domains (medical, legal, financial), subjective quality judgments, and when your automated metrics disagree with each other.

Acceptance criteria in a client SOW

Every Statement of Work should include measurable acceptance:

```
ACCEPTANCE CRITERIA - Model v1.0

Primary task: Customer inquiry classification (8 categories)
Baseline (measured [date] on [model version]):
  - Macro F1: 0.79 (95% CI: 0.76-0.82)
  - TTFT: 620ms on customer hardware spec

Acceptance threshold:
  - Macro F1 >= 0.91 (95% CI half-width <= 0.03)
  - TTFT <= 400ms on customer hardware spec
  - Hallucination rate on safety prompts: < 0.5%

Evaluation set: 500 examples, held-out, not seen during training.
Stratification: 60% routine, 25% edge, 15% safety/adversarial.
```

If the fine-tune does not beat the acceptance threshold, you iterate at your cost. If it still fails after two iterations, refund. This builds trust by eliminating customer risk.

Delivering the Model

Merging LoRA adapters

```
model = model.merge_and_unload()  # merges adapter into base weights
```

Save the unmerged adapter separately (50–200MB for 7B at r=16). This allows reproducing from any future base model version.

GGUF export pipeline

1. Merge LoRA → save safetensors
2. `python convert_hf_to_gguf.py [model_dir] --outfile model.gguf --outtype bf16`
3. `llama-quantize model.gguf model-q4km.gguf Q4_K_M`
4. **Benchmark the quantised model against your eval set before shipping.** Quantisation can change accuracy — never ship without re-running eval.

Retraining cadence

Quarterly for stable tasks. **Monthly or event-triggered** for tasks tracking current information. Monitor for: user satisfaction drops, LLM-judge score degrading >5% week-over-week, anchor test set dropping >3 absolute points, new input patterns with no training-data neighbours.

Production Monitoring (After You Ship)

What to log

Every inference: timestamp, anonymised input hash, full output, latency breakdown (TTFT + generation), token counts, model version, error code if any. Append-only structured logs.

Detecting drift

- **Output length distribution:** sudden shifts indicate truncation bugs or new refusal patterns
- **Embedding drift:** cosine distance of this week's inputs from the eval set's embedding centroid
- **Anchor prompt check:** 20 known-good prompts run daily; alert if accuracy drops below the acceptance threshold on these
- **Human feedback signal:** if available, track weekly average; alert on 5-point drops

Rollback protocol

Shadow-test new model versions for 48 hours against the incumbent. Then: 5% traffic for 24 hours (not 4 — diurnal patterns need a full day), 20% for 24 hours, 100%. Rollback if primary metric drops significantly (define "significantly" using a confidence interval, not a fixed-point delta — on a 5% traffic slice, noise can easily exceed 2 points).

Priya's Decision

Priya tried prompt engineering on her 8-category triage task. With a well-written system prompt and 3 examples per category, gemma3:4b hit 82% accuracy on 200 test tickets. Her target is 90%.

She has two options: 1. **Fine-tune gemma3:4b** with 2,000+ labelled tickets. QLoRA, r=8, 1–2 epochs. Estimated cost: £3–5 in GPU rental, plus the time to label the data (the expensive part). 2. **Try Qwen-2.5-14B with the same prompt** before fine-tuning anything. The 14B model may hit 90% out of the box, saving weeks of data labelling.

Our recommendation: try option 2 first. A 15-minute test on 200 tickets with a bigger model costs nothing and may eliminate the need to fine-tune at all. If 14B does not hit 90%, fine-tune the model that got closest. If both the 4B and 14B miss, the problem may not be model quality — it may be ambiguous labelling in the training data (the most common silent killer of fine-tuning projects).

The full post-mortem of our V2 fine-tuning failure — including the training config, the loss curves, and every mistake we made — is in Appendix A. Read it before you spend your first GPU-hour. # Chapter 5: Delivery — Documentation, Deployment, and Handoff

Priya's model passes eval. Now she needs it running in production with documentation her team can use without calling her.

This chapter is short because delivery is process, not theory. The checklist matters more than the prose.

The Delivery Package

Every model delivery — whether internal deployment or a client handoff — ships these artifacts:

1. The model file

- GGUF format (Q4_K_M default, other quants on request)
- Filename convention: `{client}-{task}-{base_model}-{version}-q4km.gguf`
- Example: `priya-fintech-triage-gemma3-4b-v1-q4km.gguf`

2. The Modelfile (Ollama)

```
FROM ./priya-fintech-triage-gemma3-4b-v1-q4km.gguf

PARAMETER temperature 0.1
PARAMETER top_p 0.9
PARAMETER num_ctx 4096

SYSTEM """
You are a customer support triage assistant for [Company].
Classify each customer inquiry into exactly one of these categories:
[categories]
Respond with JSON: {"category": "...", "confidence": 0.0-1.0, "reasoning": "..."}
```

"""

One-line install: `ollama create priya-triage -f Modelfile`

3. usage.md

Written for the engineer who inherits this model in six months, not the engineer who built it today.

Contents: - **What this model does** (one sentence) - **What it does NOT do** (one sentence — prevents misuse) - **Prompt format** with 3 copy-paste examples covering: a routine case, an edge case, and a case the model gets wrong - **Sampling parameters** and why they were chosen (temperature 0.1 for classification = deterministic; temperature 0.7 for generation = creative) - **Hardware requirements** (minimum VRAM, tested hardware, expected tok/s) - **Known limitations** (specific failure modes discovered during eval) - **How to update** (where the training data lives, how to retrain, who to ask)

4. benchmarks.md

Metric	Base model	Fine-tuned (this version)	Delta	Eval set
Macro F1	0.79	0.91	+0.12 (+15%)	500 examples, held-out
TTFT	620ms	580ms	-40ms	Same hardware
Throughput	68 tok/s	65 tok/s	-3 tok/s	Negligible

Plus: confusion matrix, per-class breakdown, 5 before/after example pairs (showing the base model's answer and the fine-tuned model's answer on the same input), and a section titled "What this model still gets wrong" with 3–5 real failure examples.

The "still gets wrong" section builds trust. Clients expect failures. Documenting them explicitly — rather than pretending perfection — positions you as someone who understands their own system's limits.

Deployment Checklist

For Ollama (single-user or small team)

- ☐ Install Ollama on target machine
- ☐ Copy GGUF + Modelfile to target
- ☐ Run `ollama create {name} -f Modelfile`
- ☐ Verify: `ollama run {name} "test prompt"` produces expected output
- ☐ Verify: `curl http://localhost:11434/api/generate -d '{"model":"{name}","prompt":"test"}'` returns valid JSON
- ☐ Set up systemd service (Linux) or launchd (macOS) for auto-start on reboot
- ☐ Confirm VRAM allocation with `nvidia-smi` or `ollama ps`
- ☐ Run 20 anchor prompts from eval set; verify all pass

For vLLM (multi-user production)

- ☐ Everything in the Ollama list, adapted for vLLM

☐ Configure `--max-model-len` (context length) and `--gpu-memory-utilization` (default 0.9)
☐ Set up API key authentication on the endpoint
☐ Configure rate limiting per API key
☐ Set up monitoring (Prometheus endpoint → Grafana or equivalent)
☐ Load test: simulate expected concurrent users and verify latency stays below SLA
☐ Configure automatic restart on OOM or crash

For client handoff (external delivery)

☐ Everything above, on client's hardware or rented GPU
☐ 30-minute handoff call: walk through usage.md, run 5 live queries together, answer questions
☐ Confirm client can independently restart the model after a reboot
☐ Confirm client has the unmerged LoRA adapter file (for future retraining)
☐ Send benchmarks.md and the eval set (so client can re-run eval independently)
☐ Confirm data deletion: all client training data removed from Fathom systems within 30 days

---

Documentation Quality Bar

The test: **could a new engineer on Priya's team deploy, operate, and troubleshoot this model using only the delivered documentation, without calling anyone?** If yes, the documentation is sufficient. If no, something is missing.

Common gaps: - How to restart the model after a server reboot (most guides skip this) - What to do when the model produces garbage output (usually a prompt format mismatch) - How to check GPU health and VRAM usage - How to know whether the model has degraded over time (link to the anchor-prompt monitoring section in Chapter 4)

Priya's Deployment

Priya's gemma3:4b triage model passed eval at 91% macro F1 on her RTX 3090. She deploys via Ollama. Her delivery package:

- `priya-fintech-triage-gemma3-4b-v1-q4km.gguf` — 2.5 GB
- `Modelfile` — with her 8 categories and 3 examples each
- `usage.md` — written for her backend engineer who will integrate the API
- `benchmarks.md` — 91% F1, confusion matrix, 5 before/after examples, 3 known failure modes

Total deployment time from "model passes eval" to "API accepting production traffic": 2–3 hours for an engineer who has done it once. First time: half a day.

The model is now running. Priya's API bill dropped from £260/month to £0/month in ongoing cost (electricity only). Her compliance officer has the answer she wanted: "the model runs on the RTX 3090 in the Shoreditch office server rack. Nothing leaves the building."

Thursday's CFO meeting lasted 4 minutes.

The next chapter covers how to price and sell all of this — including the three-way math that justifies what Fathom charges vs. building it yourself vs. hiring a consultancy. # Chapter 6: The Sale — Pricing, Positioning, and the Three-Way Math

This chapter is for two readers: Priya, who needs to justify the decision internally; and anyone building a business delivering fine-tuned local LLMs (which is what we are doing).

The Productised Service Model

A fine-tuned LLM is not a SaaS product. Each customer needs a different base model, different training data, different hardware target, and different evaluation criteria. You cannot ship a single binary that works for everyone.

But you can productise the *delivery process* — fixed price, fixed timeline, fixed deliverables, repeatable runbook. **The model is bespoke; the process is industrial.**

The three tiers

Tier	What the customer gets	One-off fee (GBP)	Monthly
Audit	Written assessment: which workloads should run on code, cloud API, local base, or fine-tuned model. Decision framework customised to their stack.	£500–£1,000	—
Deployment	Audit + local model set up on their hardware with optimised prompts. RAG pipeline if needed. Working API endpoint.	£2,500–£5,000	—
Fine-tuning	Deployment + LoRA/QLoRA fine-tune on their data. Eval report with acceptance criteria. Delivered GGUF + full package from Chapter 5.	£4,500–£15,000	—

The audit tier exists because most prospects don't actually need fine-tuning. The honest path: audit first, recommend the cheapest intervention that works (often prompt engineering or a

bigger base model), and charge for fine-tuning only when the audit says it's justified. This builds trust and prevents the fabrication trap — selling fine-tuning to someone who doesn't need it.

Anchor on the audit. Every prospect conversation starts with: "Let us look at your workload for £500. We'll tell you what should run locally and what shouldn't — including the cases where cloud API is the right answer. If local is right, we'll quote the next step."

The Three-Way Math (Build vs Buy vs Consultancy)

This is the table Priya needs when her CFO asks "why don't we just do it ourselves?"

Dimension	DIY	Big-4 consultancy	Fathom (or equivalent)
Cash cost	£0 (uses existing engineers)	£45,000–£80,000	£500–£15,000
Hidden cost	160 engineer-hours at £80–120/hr loaded = **£13,000–£19,200**	Included in fee	Included in fee
Timeline	6–10 weeks (first time)	8–16 weeks	1–3 weeks
Rework risk	~30% (first fine-tune often fails — see Appendix A)	~15% (experienced but slow)	~20% (productised but still iterating)
Outcome if it fails	Engineer spent 2 months, morale down, no model, try again	£60k spent, generic model, "lessons learned" deck	£4,500 spent, specific failure documented, iterate or refund
What you own	Everything	Depends on contract (often the consultancy owns IP)	GGUF file, full weights, no licence, no lock-in

The decision tree

```
Does your team have an ML engineer who has done a LoRA fine-tune before?
  YES: Has that engineer got 4+ weeks clear on their calendar?
     YES: Does your eval harness already exist?
        YES → DIY. You probably don't need us.
        NO → DIY but budget 2 extra weeks for eval setup.
     NO → Buy. Your engineer's opportunity cost > our fee.
  NO → Buy. The first time takes 3× longer than the second.
```

The honest position: if a team has an experienced ML engineer with time and infrastructure, they should do it themselves. Our service is for teams that don't — and for the first fine-tune, which is always the slowest and most expensive.

Pricing Philosophy

1. **Fixed SKUs, not custom quotes.** A customer should read the pricing page and know the cost. "Call for a quote" kills conversion for mid-market buyers.
2. **Front-load the margin, not the monthly.** £4,500 once is better than £400/month forever. It forces you to deliver real value and lets the customer feel they own the result.
3. **Pass through infrastructure costs transparently.** Show the GPU rental cost separately from the service fee. Builds trust. Makes DIY unattractive — they're not paying for the GPU, they're paying for the expertise to use it.
4. **Audit first, always.** Never sell fine-tuning without an audit. If the audit shows prompt engineering is sufficient, say so and charge for the audit only. The customer will come back for the next task — and they will trust you.

Finding Customers

The Nervous CTO (primary ICP)

50–500 employees. Regulated industry (legal, healthcare, finance). Using Claude or GPT via API at £1,000+/month. Compliance or finance team has flagged a concern. This person has twelve browser tabs open and no answers.

Five buying triggers (strongest first):

1. **Compliance ultimatum** — legal said "fix this." Easiest close.
2. **CFO noticed the bill** — API spend growing, finance wants answers.
3. **Vendor lock-in fear** — "what if they triple the price?"
4. **Latency or offline requirement** — product ships on-prem.
5. **Competitive moat** — proprietary model no competitor can replicate.

Qualifying in three questions:

1. "Are you paying more than £1K/month to a cloud LLM API?"
2. "Has anyone flagged concern about data leaving your perimeter?"
3. "Do you have a specific repeated task this model would handle?"

If 1 + 2 = yes → strong prospect. If only 2 = yes with compliance pressure → even stronger.

The White-Label Consultancy (highest leverage)

3–20 person AI/ML consultancy with regulated-industry clients. They already have the client relationships. They don't have a productised fine-tuning delivery capability.

The offer: wholesale delivery at £2,250 per T1 against a 10-model-per-year commitment. They resell at £8,000+ under their brand. They handle the client relationship; you handle the engineering. One partnership conversation = 10× your direct sales volume.

Handling Objections

"OpenPipe / Together is cheaper"

"For training cost, yes. But they keep the model on their servers. You pay per-token to access it forever. We hand you the file. Payback in 2–6 months if you're spending over £500/month on cloud API."

"Ollama is free"

"Yes — it's the runtime. We fine-tune the model that goes into Ollama. They are not the same thing. We deliver what Ollama *runs*."

"We have engineers who can do this"

"Great — how long will it take them? The first fine-tune typically takes a senior ML engineer 4–8 weeks including eval setup. We deliver in 10 business days because we've built the runbook. And if we fail, you're out £4,500, not two months of an engineer's salary."

"But GPT-4 is smarter"

"On general tasks, yes. On your specific task, our fine-tuned local model achieves [X%] vs [Y%] for GPT-4, at a fraction of the per-query cost, with zero data leaving your premises. For your use case, 'smarter on average' and 'better for your workflow' are different things."

"We'll just use RAG"

"RAG retrieves facts. Fine-tuning changes how the model behaves. Both are often needed — but they solve different problems. The question is: does your model not *know* something, or does it not *behave right* when it does?"

Priya's Internal Justification

Priya's CEO asks: "Why did you spend time on this instead of features?"

Her answer (one paragraph, email-ready):

> "We were paying £260/month to Anthropic for customer-support triage, growing 15% month-on-month toward £1,000/month by year-end. Compliance flagged that customer data was leaving the building via the API. I deployed a local model on existing hardware (a used RTX 3090 we bought for £800). It handles triage at 91% accuracy — above our threshold. Monthly API cost is now zero. The compliance concern is resolved. Total investment: £800 hardware + one week of engineering time. Payback: 3 months on API savings alone, immediate on compliance."

That is the paragraph. No jargon, no architecture diagrams, no AI hype. Cost in, cost out, risk resolved, timeline. A CEO reads this in ten seconds and moves on to the next email.

What This Book Cannot Sell You

We are not selling local LLMs as a universal solution. Cloud APIs are better for most teams below 1M tokens/day. Prompt engineering is sufficient for most tasks where people reach for fine-tuning. RAG is the right answer more often than fine-tuning.

What we are selling is the *decision*. The frameworks in this book — the six economic rules, the size-tier table, the three questions before fine-tuning, the eval methodology, the three-way build-vs-buy math — are designed to give you a defensible answer for your specific situation. That answer might be "stay on Claude." If it is, we saved you from buying a GPU you didn't need.

If the answer is "go local," we can help. Start with the audit.

This is the final chapter. Appendix A contains the full post-mortem of our failed V2 fine-tune. Appendix B contains the procurement FAQ for compliance and legal teams. Both are written for different readers than the chapters — read them only if your role calls for them. # Appendix A: The V2 Overfit — A Full Post-Mortem

This appendix documents our first real fine-tuning attempt. It failed. We are publishing it because the failure taught us more than any successful fine-tune would have, and because most fine-tuning content online only shows the happy path.

The Setup

Objective: Fine-tune Qwen-2.5-7B to classify and route agent messages in our internal comms system.

Base model: Qwen-2.5-7B-Instruct (the model that scores 8/10 and runs at 106 tok/s on our RTX 2070)

Hardware: Alienware desktop, RTX 2070 8GB VRAM, CUDA 12.2

Framework: QVAC LoRA (QLoRA variant, 4-bit NF4 base)

Adapter configuration: - Rank: 16 - Alpha: 32 (2× rank) - Target modules: q_proj, k_proj, v_proj, o_proj - Adapter size: 4.3 MB

Dataset: 74 synthetic examples (yes, seventy-four)

Training: 4 epochs, cosine schedule, learning rate 2e-4

Training time: 13 seconds per epoch. Total: ~52 seconds.

Cost: Approximately zero (own hardware).

The Result

Metric	Base model (no fine-tune)	V2 adapter
Eval accuracy	**34%**	**33%**

Metric	Base model (no fine-tune)	V2 adapter
Net improvement	—	**-1% (net negative)**

We spent a week preparing this fine-tune. The training itself took 52 seconds. The model got worse.

What Went Wrong

1. The dataset was catastrophically small

74 examples for a LoRA adapter with 131,072 trainable parameters. That is 1,771 parameters per example. For comparison, a reasonable ratio is 1–10 parameters per example. We were overfitting by approximately 180×.

The book's Chapter 4 says the minimum is 500 examples. When we wrote that chapter, we had not yet attempted a fine-tune. We wrote the rule from industry wisdom, then immediately violated it because we believed "our task is narrow, so fewer examples should work." It did not work.

Lesson: the 500-example minimum is not a guideline. It is a floor below which LoRA mathematically overfits on instruction tasks. We proved it empirically.

2. We trained too many epochs

4 epochs on 74 examples means the model saw each example ~4 times. The training loss curve shows classic U-shape: loss decreased until approximately step 24 (epoch ~1.3), then validation loss began increasing while training loss continued to decrease. The model memorised the training set and lost generalisation.

We shipped the epoch-4 checkpoint because training loss was lowest. **We should have selected the epoch ~1.3 checkpoint based on validation loss.**

Lesson: always select checkpoints on validation loss, never training loss. Implement early stopping. For instruction fine-tunes on small datasets, 1–2 epochs is usually correct. 3+ epochs requires aggressive regularisation and a much larger dataset.

3. No replay data

We included zero general instruction-following examples. The adapter learned to produce our routing JSON format — but at the cost of basic document-reading capability. The model became better at formatting our schema and worse at understanding the messages it was supposed to classify.

This is textbook **catastrophic forgetting.** The standard prevention is including 2–5% of general instruction-following data in the fine-tuning dataset. We skipped it because "it's a narrow task."

Lesson: always include replay data, regardless of task narrowness. The cost is trivial (50 extra examples from the base model's instruction set). The protection is significant.

4. The eval set was not hard enough

Base model accuracy was 34% — meaning the task is genuinely difficult for this model at this size. Our 10% relative improvement threshold requires hitting ~37.5%. We did not calculate this number before training. We optimised blind, checked the result after, and discovered we were below baseline.

Lesson: calculate your target number before training. If the gap between baseline and target is larger than what LoRA typically achieves (5–15 points on classification), either use more data, increase model size, or reconsider whether fine-tuning is the right approach.

5. Prompt engineering was already better

After the V2 failure, Werner built an auto-improved prompt engineering pipeline. It scored **100% on the same classification tasks** — without any fine-tuning, without any training data, without any GPU time. The base Qwen-2.5-7B with a well-structured system prompt outperformed every fine-tuning configuration we tried.

Lesson: this is why Chapter 4 says "prompt engineering first." We wrote that rule, ignored it, and the experiment proved it right.

The V3 Plan (What We're Doing Differently)

Dimension	V2 (what we did)	V3 (what we're planning)
Dataset size	74 examples	200+ minimum, targeting 500
Epochs	4	1–2, validation-loss-selected, early stopping
Rank	16	8 (smaller dataset → lower rank)
Replay data	None	3% general instruction-following
Checkpoint selection	Training loss (wrong)	Validation loss (correct)
Eval timing	After training completed	Continuous during training (checkpoint every 10 steps, eval every checkpoint)
Base model	Qwen-2.5-7B	Also trying gemma3:4b (scored higher on our quality test) and Qwen-2.5-14B (to test whether the task needs a bigger base)

V3 has not been completed at the time of writing. When it is, this appendix will be updated with the results — whether they succeed or fail.

What You Should Take From This

1. **Do not skip prompt engineering.** It is free, fast, and sometimes better than fine-tuning.

2. **500 examples is a real floor, not a suggestion.** Below it, LoRA overfits on instruction tasks.
3. **1–2 epochs for instruction fine-tunes.** More is not better unless you have massive, diverse data.
4. **Select checkpoints on validation loss.** Never training loss.
5. **Include replay data.** 2–5% of general examples prevents catastrophic forgetting at near-zero cost.
6. **Know your target number before training.** If you don't know what success looks like, you can't recognise failure.

We are publishing this because every fine-tuning guide we read showed a success story. Nobody showed the fail. The fail is where the learning lives.

V2 training config, loss curves, and evaluation scripts are available in our internal repository. Contact us if you want to see the raw data — we will share it. # Appendix B: Procurement FAQ

This appendix is written for compliance officers, procurement leads, and legal teams who have received this book from an engineer. It answers the questions you are about to ask, in the order you are about to ask them.

Data Handling

Where does our training data go? To the GPU where the fine-tuning runs — either your hardware (on-premises) or a rented GPU (Vast.ai, RunPod, Hetzner). Training data is never uploaded to a third-party cloud LLM provider (no OpenAI, no Anthropic, no Google).

Who has access to our data during the engagement? The delivery engineer running the fine-tune. No one else. Data is accessed via SSH to the training machine. No web interface, no shared storage, no multi-tenant infrastructure.

What happens to our data after delivery? Fathom permanently deletes all client training data within 30 days of model delivery. We will provide written confirmation of deletion on request.

Is our data used to train other models or improve Fathom's own systems? No. Client data is used exclusively for the contracted fine-tune. It is not mixed with other clients' data, used for Fathom's internal model development, or retained for any purpose after delivery.

Intellectual Property

Who owns the fine-tuned model weights? You do. The GGUF file is delivered to you. You may modify it, retrain it, deploy it, distribute it, or delete it. Fathom retains no licence, no usage rights, and no ongoing claim.

Who owns the LoRA adapter? You do. The adapter is delivered alongside the merged model. You may apply it to future base model versions.

Can Fathom describe the engagement publicly? Only the engagement type (e.g., "8-category classification, 7B model") — never the client name, the training data, or the specific outputs. If

we want to use the engagement in a named case study, we will request separate written permission. You may decline with no consequence.

What about the base model's licence? The base model (e.g., Qwen-2.5-7B, gemma3:4b) is open source with its own licence. We will confirm the specific licence before selection: - **Apache 2.0** (Qwen, Mistral): fully commercial, no restrictions - **Llama Community License** (Llama 3.x): commercial use permitted, restricts services >700M MAUs - **Gemma Terms** (Gemma): commercial use with additional restrictions — review terms specifically

We select Apache 2.0 models by default to avoid licence friction.

Security

What security certifications does Fathom hold? We are pre-SOC 2 and pre-ISO 27001 at the time of writing. We are a small team in the early stages of the business. Our security posture:

- All data in transit: TLS 1.3
- All data at rest on training machines: full-disk encryption
- Access control: SSH key-based, no shared credentials, no web consoles
- Logging: all training machine access logged with timestamps
- Personnel: all delivery engineers are UK-based

We will complete a security questionnaire on request. If SOC 2 or ISO 27001 certification is a procurement gate for your organisation, we cannot satisfy it today. We will be transparent about this rather than claim something we have not earned.

What about the GPU rental provider's security? When using Vast.ai or RunPod marketplace GPUs, the infrastructure is shared (multi-tenant). For clients with strict data residency or isolation requirements, we use: - **Hetzner dedicated servers** (single-tenant, German datacenters, GDPR-native) — our default for regulated-industry clients - **Client's own hardware** (we SSH in, do the work, leave) — maximum isolation

We will specify the training infrastructure in the SOW before work begins.

Compliance

GDPR: is a Data Processing Agreement (DPA) required? Yes, if client data contains personal data of EU/UK data subjects. We will sign a standard DPA. Template provided on request.

GDPR: what is the legal basis for processing? Legitimate interest (Article 6(1)(f)) or contractual necessity (Article 6(1)(b)), depending on the nature of the training data. The DPIA obligation sits with the data controller (you, the client). We will provide information required for your DPIA on request.

HIPAA: can Fathom sign a BAA? Not at this stage. If HIPAA compliance is required, we recommend: (a) de-identify all training data before sending it to us, or (b) use on-premises training on your own hardware, with Fathom providing remote engineering support only (no data leaves your facility).

FCA / PRA: regulatory considerations for financial services? Fine-tuned local models processing financial data should be documented under your operational resilience framework. Key considerations: model risk management (SR 11-7 equivalent), third-party risk assessment (Fathom as a service provider), and ongoing monitoring obligations. We will cooperate with your compliance team's vendor assessment process.

Commercial

Payment terms? 50% on contract signing, 50% on delivery. Invoice sent on signing. Payment by bank transfer or card.

What if the fine-tune does not meet the agreed acceptance criteria? Two revision rounds are included. Each revision uses updated training data or adjusted hyperparameters. If the model still does not meet the acceptance criteria after two revisions, we offer a full refund. The acceptance criteria are defined in the SOW before work begins — not after results are in.

Is there a minimum commitment? No. Each T1 delivery is a standalone engagement. No subscription, no recurring fee, no lock-in. Monthly infrastructure management (T2/T3) can be cancelled with 30 days' notice.

What is Fathom's professional indemnity (PI) insurance level? [To be confirmed — flagged as a gap in our current setup. We will state the PI level in the SOW once insurance is in place. If your procurement requires a minimum PI level, tell us before we quote and we will confirm whether we meet it.]

What is Fathom's legal entity? [To be confirmed — flagged as pending. The SOW will name the contracting entity with registration details.]

Gaps We Have Not Yet Closed (Honesty Section)

This section exists because we would rather tell you what we do not have than let you discover it during vendor assessment.

Gap	Status	Timeline
SOC 2 certification	Not started	No committed date
ISO 27001 certification	Not started	No committed date
Professional indemnity insurance	In progress	Expected before first external client
Business continuity plan	Draft stage	Expected Q2 2026
Formal escrow arrangement	Not in place	Will implement on request for engagements >£10,000

We include this table because procurement teams find gaps anyway. Finding them in our own documentation — rather than in the answers we give under pressure during a vendor assessment call — is better for both sides.

This appendix will be updated as gaps close. Version-dated at foot of page. If the date is more than 90 days old, ask us for the current version.

Last updated: April 2026