A FIELD GUIDE

The AI Tech Lead Path

A field guide & gamified roadmap to leading AI across teams and business units

5 tracks - 27 chapters - 4,660 XP

Contents

Track 01 — Foundations & the role

What the job actually is — and how to shape it.

00. What an AI Tech Lead actually is

Four hats, one mandate, and why "cross" is the hard part.

8 min - 100 XP

It is NOT "the person who is best at prompting." It's a hybrid role with four hats — technical authority, platform/enablement, change agent, and strategist/translator. The job is multiplying yourself through other people, safely, at scale.

Key ideas

1. You'll wear four hats: technical authority (~30%), platform/enablement (~25%), change agent (~25%), strategist/translator (~20%).
2. The hard part of a cross-team/cross-BU role is the word "cross": you lead people who don't report to you — so influence, standards and enablement beat raw technical skill.
3. Your edge (you've actually built the end-to-end AI SDLC) is the credibility that earns the right to influence — but it isn't the job itself.
4. Aim for a one-line mandate: "Raise the AI-engineering capability and safe-adoption of every team I touch, measurably, without becoming a bottleneck."

The four hats

An AI Tech Lead spends time across four modes, and the mix is what makes the role hard.

— Technical authority — set the bar for how AI is built and used; stay hands-on enough to be credible.

— Platform / enablement — build golden paths, tools, templates and guardrails so OTHER teams move fast safely.

— Change agent / influencer — drive adoption across teams and BUs without authority over them.

— Strategist / translator — turn business goals into an AI roadmap and translate AI reality up to execs and across to legal/risk/security.

Why "cross" changes everything

You will ask teams that don't report to you to change how they work. Authority won't do it; influence, standardization and enablement will. Your job is to be a force multiplier, not a hero coder.

Watch

- The Staff Engineer's Path — Tanya Reilly in conversation — LeadDev

Do the work

- Write your own one-line mandate for the role.
- List which of the four hats you're strongest and weakest in today.
- Identify one thing you currently do as a hero coder that you should turn into enablement.

Check yourself

1. What is the single hardest aspect of a cross-team / cross-BU AI Tech Lead role?

- Writing the most code on every team
- **Leading people who don't report to you (influence without authority) (correct)**
- Knowing every model's benchmark scores
- Owning the cloud budget

Why: "Cross" means no direct authority — influence, standardization and enablement matter more than raw technical output.

2. Select the four "hats" of the role.

- **Technical authority (correct)**
- **Platform / enablement (correct)**
- **Change agent / influencer (correct)**
- Sole code owner
- **Strategist / translator (correct)**

Why: Technical authority, platform/enablement, change agent, and strategist/translator — not sole code owner.

3. What is your existing AI-SDLC experience best understood as?

- Proof you should do all the building yourself
- **Credibility that earns you the right to influence others (correct)**
- A reason to stop coding entirely
- Irrelevant to leadership

Why: It's the credibility base — but the role is multiplying others, not doing it all yourself.

01. Shaping the role itself

Do this first — a fuzzy new role with no charter is a trap.

`8 min - 200 XP`

A brand-new cross-BU role with no definition gets judged against everyone's private assumptions. Before you start, get agreement in writing — and draft the charter yourself.

Key ideas

1. Lock six things with your sponsor in writing: mandate & scope, authority (mandate vs advise + budget), success metrics, the operating model (hub-and-spoke), visible exec backing, and protected time.
2. Draft a one-page role charter yourself and bring it to them — don't wait to be handed one.
3. Leading the definition of your own role is itself the first proof you're right for it.
4. Exec air cover makes "influence without authority" work dramatically better across BUs.

Get these six in writing

— Mandate & scope — which teams/BUs; what you own vs influence vs advise.

— Authority — what you can mandate (e.g. the paved road) vs recommend; your budget.

— Success metrics — the 4–6 numbers you'll be judged on (agree them now).

— Operating model — endorse hub-and-spoke so teams give champions' time.

— Backing — explicit, visible exec sponsorship so cross-BU leaders take it seriously.

— Time protection — guard your ~20–30% build time and enablement time from ad-hoc requests.

If you do nothing else

Lock the mandate -> master evals + governance (your authority) -> ship the eval harness & gateway paved road -> scale through champions & the guild -> measure honestly and report up.

Watch

- Negotiating scope & expectations in a new role

Do the work

- Draft a one-page role charter (mandate, scope, authority, metrics, model, backing, time).
- Agree the 4–6 success metrics with your sponsor in writing.
- Secure explicit, visible exec sponsorship for cross-BU work.
- Protect ~20–30% build time and your enablement time on the calendar.

Check yourself

1. Why draft the role charter yourself rather than wait for one?

· Because no one else can write

• Because a fuzzy new role gets judged against private assumptions — and leading its definition proves you fit it (correct)

· To avoid talking to your sponsor

· Because charters are legally required

Why: Define the role or be defined by others' assumptions; drafting it is itself the first demonstration of fit.

2. Select what you should lock in writing with your sponsor.

• Mandate & scope (correct)

• Success metrics (correct)

• Visible exec backing & protected time (correct)

· A promise to never code again

Why: Mandate, metrics, backing and protected time — keep ~20–30% build time, don't give up coding.

3. Why does explicit exec backing matter so much for this role?

- It lets you skip governance
- **Air cover makes influence-without-authority work far better across BUs (correct)**
- It increases your salary automatically
- It removes the need for champions

Why: Loud sponsorship is what makes cross-BU leaders take the collaboration seriously.

Track 02 — Technical craft

The hands-on depth that earns your authority.

02. Prompt & context engineering

Treat the context window as an engineered input, not a wish.

13 min - 160 XP

The most leveraged daily skill: shaping what goes into the model. Prompting is the surface; context engineering — deciding what information, tools and structure enter the window — is the real discipline. Teams will bring you their broken prompts, so own this cold.

Key ideas

1. Context engineering > prompt wording: the biggest wins come from giving the model the right information, examples and tools — not clever phrasing.
2. Use structure: clear roles (system vs user), explicit instructions, delimiters, and ask for structured output (JSON / tool calls) when you'll parse the result.
3. Few-shot examples beat adjectives. Show 2–5 representative examples instead of describing what you want.
4. Mind the window: models degrade with very long context ('lost in the middle') — put the most important instructions at the start and end, and prune irrelevant tokens.
5. Decompose hard tasks into steps/chains; let the model 'think' before answering for reasoning tasks; cache stable prefixes to cut cost and latency.

What actually moves quality

— Right information in context (retrieved facts, schemas, examples) — usually the #1 lever.

— Clear task framing: role, goal, constraints, output format, and what to do when unsure.

— Few-shot examples that match the real distribution of inputs.

— Output contracts: structured output / tool calling so downstream code is reliable.

Engineering practices

— Version prompts like code; never tweak a production prompt without an eval (see the Evaluation chapter).

— Prompt caching for stable system prompts / long shared context to cut cost & latency.

— Decomposition & chaining for complex tasks; reserve 'reasoning' for problems that need it.

— Defend against prompt injection when prompts include untrusted/retrieved text (see AI Security).

Watch

- AI prompt engineering: a deep dive — Anthropic
- Prompting 101 — Code w/ Claude · Anthropic

Do the work

- Rewrite one production prompt with explicit role, constraints and an output contract.
- Add 3–5 few-shot examples drawn from real inputs and measure the difference.
- Identify a stable prefix you can prompt-cache to cut cost/latency.
- Write a one-page prompting style guide for your team's repo.

Check yourself

1. What usually gives the biggest quality improvement?

· More persuasive adjectives in the prompt

• **Putting the right information, examples and tools into the context (correct)**

· A longer system prompt

· Raising the temperature

Why: Context engineering — the right inputs — beats clever wording almost every time.

2. Why place key instructions at the start AND end of a long prompt?

· It looks tidy

• **Models can lose information in the middle of long context ('lost in the middle') (correct)**

· It doubles the token budget

· It disables caching

Why: Long-context attention degrades in the middle; bookend the critical instructions.

3. You'll parse the model's answer in code. What should you do?

· Ask for prose and regex it

• **Request structured output / tool calling with a defined schema (correct)**

· Hope the format is consistent

· Lower max tokens

Why: Define an output contract (JSON/tool calls) so downstream code is reliable.

4. Before changing a production prompt, you should…

· Ship it and watch for complaints

• **Run it against an eval set to catch regressions (correct)**

· Ask one colleague if it 'feels better'

· Add more examples blindly

Why: Prompts are code — gate changes with evals to avoid silent regressions.

03. RAG & knowledge systems

Most 'the AI is wrong' bugs are retrieval bugs.

14 min - 170 XP

Retrieval-Augmented Generation grounds the model in your data. The hard part isn't calling an LLM — it's getting the right chunks in front of it. Master the retrieval pipeline and you fix the majority of quality complaints.

Key ideas

1. RAG = retrieve relevant context, then generate grounded on it. Quality is bounded by retrieval quality.
2. The pipeline: ingest -> chunk -> embed -> store -> retrieve (often hybrid keyword + vector) -> re-rank -> assemble context -> generate -> cite.
3. Chunking strategy matters enormously: too big wastes context and buries the answer; too small loses meaning. Chunk on structure, keep metadata.
4. Hybrid search (BM25 + vector) + a re-ranker beats pure vector search for most enterprise corpora.
5. Evaluate retrieval separately from generation (recall / precision / faithfulness). Add citations so answers are verifiable — essential in a regulated setting.

Where RAG breaks (and how to fix it)

— Bad chunking -> answer is split across chunks or buried. Fix: structure-aware chunks + overlap + metadata.

— Pure vector misses exact terms (codes, names). Fix: hybrid search + re-ranking.

— Stale/duplicated data -> wrong but confident answers. Fix: ingestion hygiene, dedup, freshness.

— No grounding check -> hallucination. Fix: faithfulness evals + require citations.

For a regulated insurer

— Respect data boundaries: index only data the user is allowed to see; enforce access control at retrieval time.

— Keep PII out of embeddings/logs where possible; know where the vector store lives (data residency).

— Citations + source links make answers auditable and build trust with risk/compliance.

Watch

- What is Retrieval-Augmented Generation (RAG)? — IBM Technology
- Advanced RAG: hybrid search & re-ranking

Do the work

- Diagram your RAG pipeline end-to-end and mark where failures could occur.
- Add hybrid search + a re-ranker to one retrieval flow and measure recall change.
- Build a small retrieval eval set (queries -> expected sources).
- Add citations to generated answers and verify access control at retrieval time.

Check yourself

1. A RAG system gives a confidently wrong answer. Where do you look FIRST?

- The model weights
- **The retrieval step (what context was fetched) (correct)**
- The temperature
- The output parser

Why: RAG quality is bounded by retrieval — inspect what chunks were actually retrieved.

2. Why add keyword (BM25) search alongside vector search?

- It's cheaper only
- **Hybrid search catches exact terms (codes, names) that embeddings miss (correct)**
- It removes the need for a database
- It eliminates chunking

Why: Hybrid + re-ranking outperforms pure vector for most enterprise corpora.

3. Select good practices for RAG at a regulated insurer.

- **Enforce access control at retrieval time (correct)**
- Index everything regardless of permissions
- **Add citations so answers are auditable (correct)**
- **Mind where the vector store lives (data residency) (correct)**

Why: Permission-aware retrieval, citations and data residency matter; never index beyond a user's access.

04. Agents, tools & MCP

When to let the model act — and when not to.

`13 min - 170 XP`

Agents let a model plan, call tools, observe results and loop. They're powerful and over-hyped. Your job is to know when an agent is the right tool, how to build one safely, and how to keep it bounded.

Key ideas

1. The agent loop: plan -> act (call a tool) -> observe -> repeat until done. Tools are how the model affects the world.
2. Prefer the simplest thing that works: a single well-prompted call or a fixed workflow often beats a free-roaming agent. Reach for agents when the path is genuinely dynamic.
3. MCP (Model Context Protocol) is the emerging standard for exposing tools/data to models — learn it; it's how you'll integrate enterprise systems.
4. Bound your agents: limited tool scope, allow-lists, sandboxing, step/time/cost limits, and human-in-the-loop for risky actions.
5. Agents amplify failure modes: prompt injection via tool output, runaway loops, and cost blowups. Observability and guardrails are mandatory.

Workflow vs agent

— Workflow: you orchestrate fixed steps (predictable, testable, cheaper) — default choice.

— Agent: the model decides the steps (flexible, needed for open-ended tasks) — use when the task space is dynamic.

— Many 'agents' should be workflows with one or two tool calls.

Building them safely

— Scope tools tightly; treat every tool as an attack surface.

— Sandbox side-effects; require approval for irreversible/expensive actions.

— Cap steps, time and spend; trace every tool call for debugging and audit.

— Treat tool outputs as untrusted input (indirect prompt injection).

Watch

- How we build effective agents — Barry Zhang, Anthropic
- Model Context Protocol (MCP), explained

Do the work

- Take one 'agent' idea and decide honestly: agent or fixed workflow?
- List the tools an agent would need and the blast radius of each.
- Add step/time/cost limits and tracing to an agent prototype.
- Read about MCP and sketch how you'd expose one internal system as a tool.

Check yourself

1. What is the agent loop?

· Train -> deploy -> monitor

• **Plan -> act (tool) -> observe -> repeat (correct)**

· Chunk -> embed -> retrieve

· Prompt -> cache -> return

Why: Agents plan, call tools, observe results, and iterate until the task is done.

2. When should you prefer a fixed workflow over a free-roaming agent?

· Never — agents are always better

• **When the steps are predictable; it's cheaper, testable and safer (correct)**

· Only when offline

· When you have no tools

Why: Default to the simplest thing; use agents only when the path is genuinely dynamic.

3. Select essential guardrails for agents.

- **Tightly scoped tools / allow-lists (correct)**
- **Step, time and cost limits (correct)**
- **Treat tool outputs as untrusted (injection) (correct)**
- · Unlimited autonomy for speed

Why: Bound scope, resources and trust; unlimited autonomy invites runaway loops and injection.

05. Evaluation & eval-driven development

The keystone skill. Own this and you own quality.

`15 min - 200 XP`

If you can prove an AI system is good and catch when it regresses, you own quality — and quality is your authority across teams. Most teams can't do this. Make evals the heartbeat of every AI feature, including your own AI-SDLC pipeline.

Key ideas

1. Eval-driven development: define how you'll measure success BEFORE building, then iterate against it — like test-driven development for AI.
2. Start with error analysis: read real outputs, label failure modes, and let those define your metrics. Don't start from generic benchmarks.
3. Build a golden dataset of representative inputs with expected behavior; run it in CI so a prompt/model change that regresses fails the build.
4. LLM-as-judge scales evaluation but has pitfalls (bias, inconsistency) — calibrate judges against human labels and prefer pairwise comparisons for subtle quality.
5. Combine offline evals (regression suites) with online signals (A/B tests, guardrail metrics, human feedback) from production.

How to start (this week)

— Collect ~50–100 real inputs covering the important cases and edge cases.

— Read outputs and do error analysis: cluster failures into named categories.

— Turn the top failure categories into checks (assertions, rubrics, or judge prompts).

— Wire it into CI; track scores over time; gate risky changes.

Metrics that fit the task

— RAG: retrieval recall/precision, faithfulness/groundedness, citation accuracy.

— Classification/extraction: precision/recall/F1 against labels.

— Generation/agents: task success rate, rubric scores, pairwise win-rate vs a baseline.

— Always track cost and latency alongside quality.

Watch

- Why AI evals are the hottest new skill — Hamel Husain & Shreya Shankar

- LLM Evals: common mistakes — Shankar & Husain

Do the work

- Collect 50+ real inputs and do error analysis; name your top failure modes.
- Build a golden eval set with expected behavior for one AI feature.
- Add an eval gate to CI so regressions fail the build.
- Calibrate an LLM-as-judge against a sample of human labels.
- Track quality, cost and latency on one dashboard over time.

Check yourself

1. What's the best STARTING point for building evals?

- Generic public benchmarks
- **Error analysis on real outputs to find your actual failure modes (correct)**
- A vendor's leaderboard
- Maximizing the eval set size first

Why: Start from real failures; let them define metrics. Generic benchmarks rarely reflect your task.

2. Why put a golden eval set in CI?

- To slow down deploys
- **So a prompt/model change that regresses quality fails the build automatically (correct)**
- To replace code review
- To increase token spend

Why: CI evals catch silent regressions — the core of eval-driven development.

3. A key pitfall of LLM-as-judge is…

- It's always too strict
- **Bias/inconsistency — it must be calibrated against human labels (correct)**
- It can't be automated
- It only works for code

Why: Judges scale but drift; calibrate them and prefer pairwise comparisons for subtle quality.

4. Why does owning evals give you authority as a lead?

- Evals are flashy in demos
- **Quality is your currency; proving/defending it is something most teams can't do (correct)**
- It avoids ever coding
- It replaces governance

Why: Whoever can measure and defend quality sets the bar — that's leadership leverage.

06. AI system architecture

Compound AI systems are distributed systems.

14 min - 180 XP

Real AI features are systems — retrieval + models + tools + business logic + guardrails + observability. You'll be judged across teams on architecture, not prompts. Learn the reference shape and the failure/degradation patterns.

Key ideas

1. Think 'compound AI system': orchestration around the model, not the model alone. Most value and most bugs live in the surrounding system.
2. Reference shape: client -> gateway -> orchestration -> retrieval -> model(s) -> guardrails -> observability, with caching and fallbacks throughout.
3. Design for non-determinism and failure: timeouts, retries, fallbacks (cheaper model / cached answer / graceful 'I don't know'), idempotency.
4. Separate concerns so pieces are swappable: model provider, retrieval, prompts and tools should each be replaceable behind interfaces.
5. Bake in data boundaries, PII handling and auditability from the start — far cheaper than retrofitting in a regulated org.

The reference architecture

— Gateway: one governed entry point (routing, auth, rate limits, cost caps, logging, PII redaction).

— Orchestration: prompt assembly, chaining, tool/agent control, retries & fallbacks.

— Retrieval & data: vector/keyword stores, access-controlled, with freshness.

— Guardrails: input/output validation, injection defense, policy checks.

— Observability: tracing, quality/cost/latency metrics, feedback capture.

Cross-team leverage

— Publish ONE reference architecture as the default 'paved road' so teams don't each reinvent it.

— Review 2–3 teams' designs against it; document the diffs and patterns.

— Make the model provider and retrieval swappable — vendor lock-in is a real risk.

Watch

- Building LLM applications for production — Chip Huyen
- The LLM sandwich: the data layer around LLMs — Chip Huyen

Do the work

- Draw the reference architecture for 'how a team should build a production LLM feature here.'
- Add timeouts, retries and a graceful fallback to one AI call path.
- Put the model provider behind an interface so it's swappable.
- Review one team's design against your reference and write the diff.

Check yourself

1. Why call modern AI features 'compound AI systems'?

- They use multiple GPUs
- **Value and bugs live in the system around the model (retrieval, tools, guardrails), not the model alone (correct)**
- They compound interest
- They require multiple models always

Why: The orchestration, retrieval and guardrails are where most engineering (and risk) lives.

2. How should the architecture handle a model/API being slow or down?

- Crash and alert only
- **Timeouts + retries + a fallback (cheaper model / cache / graceful 'I don't know') (correct)**
- Increase max tokens
- Disable logging

Why: Design for non-determinism and failure with graceful degradation.

3. Why put the model provider behind an interface?

- To make it slower
- **Swappability — avoid vendor lock-in and enable model right-sizing (correct)**
- To bypass the gateway
- It's required by GDPR

Why: Keeping providers/retrieval swappable protects against lock-in and lets you change models.

07. LLMOps & production

Ship, watch, roll back — the AI gateway is your paved road.

13 min - 170 XP

Getting an AI feature to a demo is easy; running it reliably is the job. LLMOps is the discipline of deploying, versioning, observing and governing AI in production — and the AI gateway is the single highest-leverage platform thing you can champion.

Key ideas

1. Version everything: prompts, models, retrieval configs and tools — so you can deploy, compare and roll back deliberately.
2. Observe in production: trace every request (tokens, latency, cost, tool calls), watch quality drift, and capture user feedback for your eval sets.
3. Build an AI gateway: one governed path for all LLM calls — routing, model allow-list, rate limits, cost caps, logging, PII redaction, data-residency.
4. Roll out safely: canary/A-B prompt & model changes; have a kill switch and fallbacks.
5. Close the loop: production signals feed evals, which gate the next change. Ops and evals are two halves of one system.

The AI gateway (paved road)

— Every team's LLM call goes through one governed proxy -> consistent logging, cost control and safety.

— Central place to enforce model allow-lists, redact PII, and pin data residency (key for an EU insurer).

— Makes the compliant path the easy path — adoption follows defaults.

Operate it

— Dashboards for quality, cost, latency and abuse; alerts on spikes/drift.

— Prompt/model registry with versions and rollback.

— Feedback capture -> labeled data -> eval sets -> CI gate.

Watch

- Observability for LLMs — Phillip Carter · LLMs in Prod
- Traceability & observability in multi-step LLM systems — Langfuse

Do the work

- Spec an AI gateway: routing, model allow-list, cost caps, logging, PII redaction, residency.
- Add request tracing (tokens, latency, cost, tool calls) to one feature.
- Stand up a prompt/model registry with versioning and rollback.
- Wire production feedback into your eval set.

Check yourself

1. What makes an AI gateway the highest-leverage platform investment?

· It speeds up the model

• **One governed path gives consistent logging, cost control, safety & residency for every team (correct)**

· It removes the need for prompts

· It trains custom models

Why: A single governed entry point is how you standardize and de-risk AI across teams.

2. Why version prompts and models?

· Cosmetic tidiness

• **So you can compare, deploy deliberately and roll back when something regresses (correct)**

· To increase latency

· GDPR requires semantic versioning

Why: Treat prompts/models like deployable artifacts with rollback.

3. How do ops and evals relate?

- They're unrelated
- **Production signals feed eval sets, which gate the next change — one closed loop (correct)**
- Evals replace monitoring
- Ops replaces evals

Why: Production feedback -> evals -> CI gate -> safer change. Two halves of one system.

08. AI security & red-teaming

Non-negotiable at an insurer. Guardrails first, evangelism second.

13 min - 180 XP

AI adds new attack surfaces on top of normal appsec. One PII-in-a-prompt incident can set a whole program back. Know the OWASP LLM risks, defend against prompt injection, and red-team your own systems before someone else does.

Key ideas

1. Learn the OWASP Top 10 for LLM Applications — prompt injection, sensitive-data disclosure, insecure output handling, supply-chain, excessive agency, and more.
2. Prompt injection (direct & indirect): untrusted text — including retrieved docs and tool outputs — can hijack the model. Never trust model output as a command.
3. Protect data: keep PII/secrets out of prompts, logs and training; enforce least-privilege on tools and retrieval; redact at the gateway.
4. Constrain 'agency': the more actions an agent can take, the bigger the blast radius — scope tools, require approval for irreversible actions.
5. Red-team proactively: try to break your own system, document findings, and turn them into standards. Partner with security as an ally, early.

Top risks to internalize

- Indirect prompt injection via RAG content or tool results.
- Sensitive data leakage through prompts, logs, or over-broad retrieval.
- Insecure output handling (model output used in SQL, shell, HTML -> injection).
- Excessive agency / over-permissioned tools.
- Supply chain: untrusted models, datasets and plugins.

Run a red-team exercise

- Define targets (data exfiltration, unauthorized actions, jailbreak bypass of policy).
- Attack: injection payloads in inputs/docs, role-play jailbreaks, tool abuse.
- Record what worked; add guardrails + eval checks; re-test.
- Publish a remediation checklist as a reusable standard.

Watch

- Explained: the OWASP Top 10 for LLM Applications
- Prompt injection: attacks & defenses

Do the work

- Read the full OWASP Top 10 for LLM Applications.
- Run a red-team exercise on one of your AI features; document findings.
- Add output handling defenses (never execute model output unsafely).
- Confirm PII redaction in prompts/logs and least-privilege on tools.
- Turn findings into a reusable AI security checklist with your security team.

Check yourself

1. What is indirect prompt injection?

· A typo in the system prompt

• **Malicious instructions hidden in retrieved docs or tool outputs that hijack the model (correct)**

· A slow API call

· Too many tokens

Why: Untrusted content the model reads (RAG, tool output) can carry injected instructions.

2. Why is 'excessive agency' a security risk?

· Agents are too slow

• **The more actions/permissions an agent has, the larger the blast radius if it's manipulated- (correct)**

· It increases token cost only

· It breaks caching

Why: Over-permissioned tools mean a hijacked agent can do real damage — scope tightly.

3. Select sound AI-security practices.

- **Keep PII/secrets out of prompts and logs (correct)**
- **Treat model output as untrusted before executing it (correct)**
- **Red-team your own system and standardize fixes (correct)**

· Give agents broad permissions for convenience

Why: Least privilege, untrusted-output handling and proactive red-teaming; never broaden agency for convenience.

09. Cost, FinOps & model selection

"What does it cost and what's the ROI?" — have the answer ready.

`11 min - 150 XP`

Execs will ask what AI costs and what it returns. Understanding token economics, right-sizing models, and computing a unit cost turns you from enthusiast into a credible owner of the budget.

Key ideas

1. Token economics: you pay per input + output token. Long context, verbose outputs and chatty agents are the usual cost drivers.
2. Right-size the model: use the smallest model that passes your evals; reserve frontier models for the hard steps. Cascade (cheap first, escalate if needed).
3. Cut cost without losing quality: prompt caching, retrieval instead of stuffing context, shorter outputs, batching, and caching repeated answers.
4. Compute a unit metric: cost per resolved ticket / generated PR / answered question — this is what execs and ROI cases need.
5. Watch hosting trade-offs: API vs self-host vs EU-region managed (Bedrock/Azure OpenAI) balances cost, control and data residency.

Where the money goes

— Oversized models for easy tasks; frontier where a small model would pass evals.

— Bloated context (stuffing instead of retrieving) and long, unbounded outputs.

— Agents looping without step/cost caps.

— No caching of stable prompts or repeated queries.

Build the ROI case

— Define a unit of value (a ticket, a PR, a case) and measure cost per unit.

— Compare to the human/time cost it replaces or accelerates — conservatively.

— Set budgets and alerts at the gateway; track cost alongside quality.

Watch

- Reducing LLM cost: token & model strategies
- Choosing the right model (small vs frontier)

Do the work

- Instrument cost per request for one AI feature.
- Try a smaller model behind your eval set; keep it if it passes.
- Add prompt caching and output-length limits where safe.
- Compute one unit-cost metric (e.g. cost per resolved ticket).

Check yourself

1. What's the disciplined approach to model selection?

· Always use the most powerful model

• **Use the smallest model that passes your evals; escalate only the hard steps (correct)**

· Pick by brand

· Whatever is newest

Why: Right-size against evals and cascade — frontier only where needed.

2. Which metric best supports an ROI conversation with execs?

- Total tokens used
- **Cost per unit of value (per ticket / PR / case) (correct)**
- Number of prompts written
- Model parameter count

Why: Unit economics (cost per outcome) is what ROI cases are built on.

3. Select legitimate ways to cut cost without hurting quality.

- **Prompt caching of stable prefixes (correct)**
- **Retrieval instead of stuffing huge context (correct)**
- **Bounding output length (correct)**
- Removing all evals

Why: Caching, retrieval and output limits cut cost; never remove your evals.

Track 03 — Leadership & influence

Leading and scaling people you don't manage.

10. Soft skills — the real job

Influence, translation, evangelism with substance, coaching at scale.

`12 min  -  150 XP`

For a cross-team/cross-BU role these outweigh the hard skills. Rank them this high in your own development plan.

Key ideas

1. Influence without authority is the #1 skill: build coalitions, empower a champion inside each team, trade value, and make the right thing the EASY thing.
2. Translate fluently across three audiences — engineers (precise, show code/evals), product (outcomes/risk/cost), execs & risk (one-page, decision-oriented).
3. Evangelize with substance: every demo carries a number and a way to verify quality, so you're never the hype person who gets found out.
4. Coach and teach at scale — your success metric is OTHER people's capability, not your own output.
5. Handle fear honestly: bring skeptics in as reviewers/red-teamers so they become owners, not blockers.

Influence without authority

You'll ask teams that don't report to you to change how they work. Find and empower champions, trade value ("I'll save your team 2 days -> you pilot my approach"), make the compliant/high-quality path the easy default, and let others take the credit.

Communication & translation

— To engineers: precise, technical, credible — show code/evals, not slides.

— To product/business: outcomes, risk, cost, timelines — no jargon.

— To execs/risk/legal: one page, decision-oriented, options with tradeoffs.

Coaching, facilitation, change

Run workshops, office hours and a community of practice; facilitate design forums neutrally and drive to documented decisions; manage change by acknowledging fear and never overselling.

Watch

- Allan Cohen on Growing Influence Without Authority — Babson / Box
- Leading without authority — Mary Meaney Haynes · TEDxBasel

Do the work

- Pick one team, get a measurable win WITH them, and let them present it.
- Take one AI initiative and write the same update three ways (engineer / product / exec).
- Identify one skeptic and invite them in as a reviewer or red-teamer.
- Read Influence Without Authority (Cohen & Bradford) or Switch (Heath).

Check yourself

1. What is the #1 soft skill for a cross-team AI Tech Lead?

· Public speaking

• **Influence without authority (correct)**

· Writing documentation

· Time management

Why: You change how teams work without managing them — influence is the core lever.

2. When communicating with executives and risk officers, the best format is…

· A long technical deep-dive with code

• **A one-page, decision-oriented summary with options and tradeoffs (correct)**

· A live coding demo

· A Slack thread of links

Why: Execs need decisions framed with risk and options, not implementation detail.

3. What's the right move with a vocal AI skeptic?

· Ignore them and route around

· Overrule them with authority

• **Enroll them as a reviewer / red-teamer so they become an owner (correct)**

· Escalate to their manager

Why: Skeptics ignored become blockers; skeptics enrolled become advocates.

4. "Evangelism with substance" means…

- Hyping AI in every meeting
- **Always attaching a number and a way to verify quality to your demos (correct)**
- Only presenting to executives
- Avoiding demos entirely

Why: Demos backed by real evals build durable trust; hype gets found out.

11. Stakeholder & executive communication

Translate AI reality up, down and sideways.

10 min - 150 XP

Much of the role is translation: turning AI reality into language engineers, product, execs and risk can each act on. The same update, framed three ways, is a core skill — and writing well is your highest-leverage scaling tool.

Key ideas

1. Match the audience: engineers want precision and evidence; product wants outcomes, cost and timelines; execs/risk want a one-page decision with options and trade-offs.
2. Lead with the answer (BLUF — bottom line up front), then support it. Execs read the first two lines.
3. Quantify honestly and conservatively; attach a number and a way to verify. Trust is your currency.
4. Written artifacts scale you: a crisp one-pager or decision memo travels to rooms you're not in.
5. Manage up with no surprises: regular, short status to your sponsor; surface risks early, never bury them.

The three registers

— To engineers: show code/evals, be precise, invite critique.

— To product/business: outcomes, risk, cost, timeline — no jargon.

— To execs/risk/legal: one page; decision, options, trade-offs, recommendation.

Make it land

— Use the inverted pyramid: conclusion first, detail below.

— Pair every claim with evidence and a confidence level.

— Tell a before/after story with one memorable number.

Watch

- Communicating with executives (for technical leaders)
- Writing a one-page decision memo

Do the work

- Write the same AI update three ways: engineer Slack post, product one-pager, exec slide.

- Draft a one-page decision memo (recommendation + options + trade-offs) for a real choice.
- Set a recurring, short, no-surprises update to your sponsor.

Check yourself

1. An exec asks about an AI initiative. Best format?

- A 20-slide technical deep dive
- **One page: recommendation, options, trade-offs, with the bottom line up front (correct)**
- A live debugging session
- A link to the repo

Why: Execs want a decision framed with options and trade-offs, conclusion first.

2. Why are written artifacts so valuable for this role?

- They look professional
- **A crisp one-pager scales you — it reaches rooms and people you're not in (correct)**
- They replace meetings entirely
- They avoid accountability

Why: Good writing is leverage; it carries your thinking across the org without you present.

3. What does 'manage up with no surprises' mean?

- Only share good news
- **Give regular short updates and surface risks early, never bury them (correct)**
- Escalate everything
- Wait for quarterly reviews

Why: Sponsors trust leads who flag risks early and keep them informed.

12. Coaching, enablement & running a guild

Your success metric is other people's capability.

`11 min - 160 XP`

You scale by making others better, not by doing the work yourself. That means coaching champions, building enablement people actually use, and running a community of practice that compounds knowledge across teams and BUs.

Key ideas

1. Teach to scale: coach champions and pair-program rather than taking the keyboard. The win is when teams ship AI features without you in the room.
2. Run a community of practice (a guild): a biweekly cross-BU forum to share patterns, demos, problems and decisions. It's your main scaling lever.
3. Enablement that sticks: short docs, office hours, templates and runnable examples beat long courses no one finishes.

4. Champion model: recruit one enthusiastic engineer per team, give them tools, air-cover and recognition; they multiply your reach.

5. Facilitate well: surface dissent, keep it neutral, and drive every session to a documented decision or takeaway.

Coaching over doing

— Ask questions and pair, rather than solving it for them.

— Create a champion onboarding kit so the role is repeatable.

— Recognize champions publicly — recognition sustains volunteers.

Running the guild

— Light, regular cadence (biweekly) with a simple format: demo + pattern + decision.

— Rotate presenters across teams/BUs so knowledge spreads.

— Capture patterns and decisions somewhere durable and searchable.

Watch

- Running an engineering community of practice
- Coaching engineers (instead of doing it for them)

Do the work

- Recruit and name your first AI champion in a pilot team.
- Run the first biweekly guild (demo + pattern + decision).
- Write a champion onboarding kit.
- Start a searchable home for patterns & decisions.

Check yourself

1. What's the strongest sign your enablement is working?

· You write the most AI code

• **Teams ship AI features without you in the room (correct)**

· You attend every standup

· You own the biggest budget

Why: Independence is the goal — capability multiplied through others.

2. What's the main purpose of the guild / community of practice?

· A status meeting

• **A cross-BU forum to share patterns, demos and decisions — your scaling lever (correct)**

· A place to assign tickets

· A replacement for governance

Why: The guild spreads knowledge across teams and BUs without you being everywhere.

3. Which enablement is most likely to actually be used?

- A 40-hour course
- **Short docs, office hours, templates and runnable examples (correct)**
- A one-time all-hands
- A long PDF

Why: Small, practical, on-demand enablement beats long courses people don't finish.

13. The operating model across teams & BUs

Hub-and-spoke: centralize standards, decentralize the building.

9 min - 150 XP

Don't try to be personally present in every team — you'll become a bottleneck (the classic failure). Run a federated "Center of Enablement" model.

Key ideas

1. Hub (you + maybe 1–2 people): build the paved road (gateway, templates, eval harness, reference architecture), set standards, own governance, run the guild.
2. Spokes (AI champions): one upskilled, empowered engineer per team does the day-to-day. This is how you scale across BUs without cloning yourself.
3. A cross-BU community of practice (biweekly guild) is your main scaling lever.
4. Paved road > policing: make the compliant, high-quality path the EASIEST path — people follow defaults, not documents.
5. Operating principle: centralize the standards and platform; decentralize the building.

Hub-and-spoke (Center of Enablement)

A small hub builds and owns the paved road and standards; embedded AI champions in each team build day-to-day and multiply your reach. A biweekly guild shares patterns, demos and decisions across BUs.

Why this beats centralizing everything

— Avoids the #1 failure mode: becoming the bottleneck on every AI feature.

— Champions give you eyes, hands and trust inside teams you don't manage.

— Defaults (paved road) drive behavior far more reliably than policy documents.

Watch

- Team Topologies: enabling teams & the platform model

Do the work

- Sketch your hub-and-spoke: who's in the hub, which teams get champions.
- Recruit and name your first AI champion in a pilot team.
- Schedule a recurring biweekly guild / community-of-practice meeting.

- List one "paved road" default you can ship so the right way is the easy way.

Check yourself

1. What's the core operating principle of the model?

- Centralize everything in one team
- Decentralize standards, centralize building
- **Centralize the standards & platform, decentralize the building (correct)**
- Let every team do whatever they want

Why: Hub owns standards + paved road; spokes (champions) do the building.

2. What role do "AI champions" play?

- They approve every PR centrally
- **They are embedded, upskilled engineers who multiply your reach inside teams (correct)**
- They replace the governance process
- They manage the cloud budget

Why: Champions are spokes — they let you scale across BUs without being in every room.

3. Why prefer a "paved road" over policing?

- Policing is illegal
- **People follow easy defaults far more reliably than policy documents (correct)**
- Paved roads need no maintenance
- It avoids ever talking to security

Why: Make the compliant, high-quality path the easiest path and adoption follows.

14. Hiring & growing AI capability

Build the bench, don't hoard the skill.

`9 min - 140 XP`

An AI Tech Lead grows the org's capability, not just their own. That means knowing what to hire vs upskill, what 'good' looks like in AI engineers, and how to build a durable bench so the program outlives you.

Key ideas

1. Upskill first: most AI capability is grown from strong existing engineers, not hired in. Hire to fill specific gaps (e.g. ML/eval depth, data engineering).
2. Screen for the right things: problem framing, evals/measurement instinct, debugging non-determinism, and judgment about when NOT to use AI — over trivia.
3. Create growth paths: champion -> embedded AI engineer -> mentor. Make the role attractive and recognized.
4. Avoid the hero trap and the single-point-of-failure: document, pair, and spread knowledge so capability is resilient.

5. Plan succession from the start: a second hub person and a champion bench mean the program survives turnover.

Buy vs build (people edition)

— Build: upskill curious, strong engineers via the guild, projects and pairing.

— Buy: hire for specific depth you lack (evals/ML, data, security) — usually a few key roles.

— Beware hiring 'prompt experts' with no engineering depth.

What 'good' looks like

— Frames problems and measures them (evals) before building.

— Comfortable with ambiguity and non-determinism; ships and iterates.

— Knows the limits — picks the simplest tool, including 'not AI'.

Watch

- Hiring AI engineers: what to screen for
- Upskilling a team for AI

Do the work

- List capability gaps; mark each as 'upskill' or 'hire'.
- Define a simple AI-engineer competency rubric (framing, evals, judgment).
- Sketch a growth path: champion -> embedded AI engineer -> mentor.
- Name a potential second hub person (succession).

Check yourself

1. What's the default way to build AI capability in an org?

· Hire all new specialists

• **Upskill strong existing engineers, hiring only for specific gaps (correct)**

· Outsource everything

· Wait for the model to improve

Why: Most capability is grown internally; hire selectively for depth you lack.

2. Which is the best signal in an AI engineer?

· Memorizes model benchmark numbers

• **Frames problems and measures them with evals; knows when NOT to use AI (correct)**

· Writes the longest prompts

· Prefers the newest model always

Why: Measurement instinct and judgment beat trivia and tool-chasing.

3. Why plan succession and spread knowledge early?

- To reduce your importance
- **So the program is resilient and survives turnover — avoid single points of failure (correct)**
- It's an HR rule
- To slow hiring

Why: A bench and a second hub person keep the program alive beyond any one person.

Track 04 — Strategy, governance & value

Choosing the right work and running it safely.

15. Finding high-value use cases

The hardest skill: deciding what to build at all.

11 min - 160 XP

Most AI effort is wasted on the wrong problems. A lead's product sense — spotting use cases with real value, feasibility and acceptable risk, and saying no to the rest — is what separates impact from theatre.

Key ideas

1. Score use cases on three axes: value (business impact) × feasibility (can AI do it reliably?) × risk (regulatory/safety/reputational). You want high value, proven feasibility, manageable risk.
2. Start where errors are cheap and volume is high (drafting, triage, search, summarization) before high-stakes automated decisions.
3. Prefer 'assist the human' over 'replace the human' early — it captures value at far lower risk, crucial in insurance.
4. Say no well: a clear, criteria-based no protects credibility and capacity. Maintain a visible prioritized backlog.
5. Build vs buy: buy commodity capabilities, build where you have proprietary data or real differentiation.

A simple prioritization

— Value: revenue, cost, risk-reduction, or experience — quantify roughly.

— Feasibility: is the task within reliable LLM ability? Can you eval it?

— Risk: regulatory tier, data sensitivity, blast radius of errors.

— Sequence: quick credible wins first, then bigger bets.

Patterns that tend to pay off

— Drafting & summarization (tickets, docs, comms).

— Search & Q&A over internal knowledge (grounded RAG).

— Triage/classification & routing with a human check.

— Developer productivity (your own proven area).

Watch

- Identifying high-value AI use cases
- AI product strategy: build vs buy

Do the work

- List candidate use cases and score each on value × feasibility × risk.
- Pick one high-value, low-risk quick win to pilot with a team.
- Make a 'no / not yet' list with the reasons, and share it.
- Decide build vs buy for one capability and justify it.

Check yourself

1. What three axes should you score AI use cases on?

· Hype × novelty × budget
• **Value × feasibility × risk (correct)**
· Model size × latency × cost
· Team size × deadline × scope

Why: Value, feasibility and risk together tell you what's worth building.

2. Where should you usually start, especially at an insurer?

· High-stakes fully-automated decisions
• **High-volume tasks where errors are cheap and a human stays in the loop (correct)**
· The most technically impressive idea
· Whatever a vendor is selling

Why: Assist-the-human, low-stakes, high-volume tasks capture value at low risk first.

3. Why is saying 'no' well part of the job?

· To seem tough
• **A criteria-based no protects credibility and capacity for the bets that matter (correct)**
· To avoid building anything
· To slow other teams down

Why: Disciplined prioritization (and visible reasons) is how you create real impact.

16. Governance, risk & compliance

Your superpower at a regulated insurer (EU AI Act · DORA · GDPR).

`14 min - 200 XP`

This is where you leapfrog a "normal" AI lead. At an insurer the constraint isn't can we build it — it's can we run it safely under regulation. Few people speak both AI and risk. Master this and you're indispensable across every BU.

Key ideas

1. Know the EU AI Act risk tiers (prohibited / high-risk / limited / minimal). Insurance pricing & risk assessment for life/health can be high-risk.
2. Know DORA (ICT & third-party/vendor risk, resilience testing — your AI vendors are third parties) and GDPR (lawful basis, data minimization, Art. 22 automated decisions, never leak PII into prompts/logs).
3. Leverage existing model risk management: insurers already validate actuarial models — frame AI/LLM governance as an extension of frameworks the company already trusts.
4. Use NIST AI RMF and ISO/IEC 42001 as the skeleton for internal AI governance.
5. Build an AI use-case intake & risk-triage template — it puts you at the center of every BU's workflow.

The regulatory landscape (working knowledge)

— EU AI Act — risk tiers, transparency obligations, timelines; insurance is explicitly in scope for parts of it.

— DORA — applies to financial entities incl. insurers; ICT risk, third-party (vendor) risk, operational resilience testing.

— GDPR — lawful basis, purpose limitation, DPIAs, automated-decision rights (Art. 22), and strict PII handling.

— Standards — NIST AI RMF, ISO/IEC 42001 (AI management systems), ISO 23894 (AI risk).

Your unfair advantage: existing model risk governance

Insurers already have actuarial model governance and model inventories. Position AI/LLM governance as an extension of frameworks the company already trusts — this makes you credible to risk officers fast.

Make it practical

Build an intake template (~10 questions) that auto-classifies risk tier and routes to the right approvals, and co-author governance lightly WITH Legal, Privacy/DPO, Security and Model Risk so it's theirs too.

Watch

- The EU's AI Act Explained
- Explained: The OWASP Top 10 for LLM Applications

Do the work

- Build an AI use-case intake & risk-triage template (~10 questions -> risk tier -> approvals).
- Identify one named contact each in Legal, Privacy/DPO, Security and Model Risk.
- Read the EU AI Act risk-tier summary and note which of your use cases could be high-risk.
- Map how AI governance can extend your company's existing model-risk framework.

Check yourself

1. Why is governance described as a "superpower" specifically at an insurer?

- Because regulation is optional there
- **Because the constraint is safe operation under regulation, and few people speak both AI and risk (correct)**
- Because insurers don't use AI
- Because it removes the need for evals

Why: The bottleneck is running AI safely under regulation; bridging AI + risk makes you indispensable across BUs.

2. What's the smartest way to make AI governance credible to risk officers quickly?

- Invent a brand-new framework from scratch
- **Frame it as an extension of the existing model-risk governance they already trust (correct)**
- Avoid risk officers until launch
- Copy a startup's playbook verbatim

Why: Insurers already validate actuarial models — extend that trusted framework rather than replacing it.

3. Select regulations/standards directly relevant to AI at an EU insurer.

- **EU AI Act (correct)**
- **DORA (correct)**
- **GDPR (correct)**
- **NIST AI RMF / ISO 42001 (correct)**

Why: All four matter: EU AI Act (risk tiers), DORA (ICT/vendor resilience), GDPR (data), NIST/ISO (frameworks).

4. What single artifact puts you at the center of every BU's AI workflow?

- A model leaderboard
- **An AI use-case intake & risk-triage template (correct)**
- A prompt cheat sheet
- A cost dashboard

Why: Intake/triage routes every proposal through you and saves every BU pain.

17. Responsible AI: fairness & ethics

Especially load-bearing in insurance pricing & underwriting.

`12 min - 170 XP`

At an insurer, fairness isn't optional ethics theatre — biased models can mean discriminatory pricing or underwriting, legal exposure, and real harm. Responsible AI is part of your technical authority: know the failure modes and how to measure and mitigate them.

Key ideas

1. Bias enters through data (historical discrimination, unrepresentative samples), labels, and proxies (a feature that stands in for a protected attribute).
2. There are multiple, mathematically conflicting definitions of fairness — you usually can't satisfy all at once, so the choice is contextual and must be made with legal/risk.

3. Measure it: test outcomes across protected groups (e.g. demographic parity, equal error rates) and document the trade-offs you chose.

4. Mitigate across the lifecycle: better data, feature review for proxies, constraints/post-processing, human oversight, and ongoing monitoring for drift.

5. Transparency & contestability: be able to explain decisions and offer recourse — aligned with GDPR Art. 22 and the EU AI Act for high-risk uses.

Why insurance is special

— Pricing/underwriting can be 'high-risk' under the EU AI Act and is closely regulated.

— Proxies are a classic trap: postcode, name, or behavior can encode protected attributes.

— Decisions affect people materially — fairness, explanation and recourse matter.

What a lead does about it

— Bake fairness checks into evals; review features for proxies with the business.

— Document the fairness definition chosen and the trade-off, with legal/risk sign-off.

— Keep a human in the loop for consequential decisions; monitor for drift over time.

Watch

- AI bias and fairness — MIT 6.S191
- Algorithmic bias and fairness — Crash Course AI

Do the work

- Add fairness checks (outcomes across groups) to one model's eval suite.
- Review a feature set for proxies of protected attributes.
- Document the fairness definition & trade-off with legal/risk.
- Ensure consequential decisions have human oversight and a recourse path.

Check yourself

1. What is a 'proxy' in the fairness context?

· A caching server

• **A feature that stands in for a protected attribute (e.g. postcode encoding ethnicity) (correct)**

· A fallback model

· A type of embedding

Why: Proxies let bias sneak in even without using a protected attribute directly.

2. Why can't you simply 'satisfy fairness' with one metric?

- Fairness can't be measured
- **There are multiple, mathematically conflicting definitions — the choice is contextual (correct)**
- Regulators forbid metrics
- It's purely subjective

Why: Different fairness definitions conflict; you choose contextually and document the trade-off.

3. Why is this chapter especially load-bearing at an insurer?

- Insurers don't use models
- **Pricing/underwriting can be high-risk and regulated; biased models cause real harm and legal exposure (correct)**
- It only affects marketing
- Fairness is optional there

Why: Consequential, regulated decisions make fairness, explanation and recourse essential.

18. Measuring success

If you can't measure it, you can't lead it — or defend the role.

9 min - 150 XP

Track both org/value metrics and your-own-growth metrics. Be conservative and honest — execs trust honest numbers, and vanity metrics destroy the trust your role depends on.

Key ideas

1. Org/value metrics: adoption (active usage, not seats), DORA metrics before/after, eval scores & AI-incident rate, value (time saved, cost per unit), and % of use cases through risk triage.
2. Your-own-growth metrics: # champions trained & active, # teams shipping AI features WITHOUT you in the room, reusable artifacts shipped, cross-BU reach.
3. Measure outcomes (cycle time, defect rate, satisfaction), not vanity (lines of AI-generated code).
4. Conservative, honest numbers build the trust that vanity metrics destroy.

Org / value metrics

— Adoption: teams/engineers actively on the paved road.

— Throughput: DORA (lead time, deploy frequency, change-fail rate, MTTR) before vs after; ticket cycle time.

— Quality: eval scores over time; AI-feature incident rate; % AI features with evals in CI.

— Value: time saved, cost per unit (per PR/ticket/case), business outcomes per use case.

— Risk: % use cases through triage; # red-team findings closed.

Your own growth

The clearest signal you're succeeding: teams ship AI features without you in the room, and the number of active champions grows.

Watch

- Why AI evals are the hottest new skill for product builders — Hamel Husain & Shreya Shankar

Do the work

- Define the 4–6 numbers you'll be judged on and baseline them now.
- Capture DORA metrics for a pilot team before any AI rollout.
- Pick one honest unit value metric (e.g. cost/time per resolved ticket).
- Set up a simple metrics dashboard (adoption, DORA, cost, eval scores).

Check yourself

1. Which is a vanity metric to avoid leaning on?

- Cycle time
- **Lines of AI-generated code (correct)**
- Change-fail rate
- Eval scores over time

Why: Lines of generated code says nothing about value; measure outcomes instead.

2. What's the clearest signal that YOU are succeeding as the lead?

- You personally write more AI code each quarter
- **Teams ship AI features without you in the room; active champions grow (correct)**
- You attend more meetings
- You own a bigger cloud budget

Why: Your success is other people's capability — independence and a growing champion network.

3. Why prefer conservative, honest numbers to optimistic ones?

- They're easier to compute
- **They build the trust the role depends on; vanity numbers destroy it (correct)**
- Regulators require pessimism
- They make the AI look worse

Why: Trust is your currency across BUs; honest metrics protect it.

19. The phased plan (now -> 12 months)

Listen -> quick win -> paved road -> scale -> strategic.

`11 min - 200 XP`

A sequence you can actually execute, from before the role starts to a durable, strategic position within a year.

Key ideas

1. Phase 0 (now -> start): lock the mandate, audit your gaps, productize one tool with evals, start key relationships.
2. First 30 days: listen & map (don't propose yet), publish what you're hearing, land one low-risk quick win WITH a team, stand up the guild.
3. 30–90 days: ship the eval harness, POC the AI gateway, publish reference architecture + risk-triage template, recruit first champions, baseline metrics.
4. 3–6 months: scale to 3–5 teams across >=2 BUs via champions, launch enablement, formalize governance v1, first red-team, first quarterly impact report.
5. 6–12 months: cross-BU roadmap, the model runs without you in every room, governance v2, build your brand, plan succession.

Phase 0 — before the role starts

— Define the role with your sponsor in writing (mandate, scope, metrics).

— Audit gaps; pick your 3 weakest high-impact areas (likely evals, AI-systems architecture, governance).

— Productize one tool: eval harness + 2-page architecture doc + a cost number.

— Coffee with one person each in Security, Privacy, Model Risk, Platform, and 2–3 leads in other BUs — listen.

Phases 1–4 — listen, build, scale, strategize

Sequence the work: listen & quick win -> build the paved road -> scale through champions & the guild -> become strategic and durable. Never skip straight to scaling.

Watch

- The first 90 days in a new (leadership) role

Do the work

- Phase 0: get a written mandate (scope + metrics) from your sponsor.
- Phase 1: run a listening tour and publish an AI landscape map.
- Phase 1: deliver one low-risk quick win WITH a willing team.
- Phase 2: ship the eval harness and POC the AI gateway.
- Phase 3: scale to 3–5 teams across >=2 BUs via champions.

Check yourself

1. What should you do in your FIRST 30 days?

· Mandate a new platform org-wide

• **Listen & map (don't propose yet), then land one low-risk quick win (correct)**

· Rewrite every team's codebase

· Hire a large team

Why: Listen first, build trust, prove value with a quick win before proposing big changes.

2. What's the right order of the phases?

- Scale -> listen -> quick win -> governance
- **Listen/quick win -> build paved road -> scale via champions -> strategic (correct)**
- Governance -> scale -> listen -> build
- Build everything first, then listen

Why: Listen & quick win -> paved road -> scale -> strategic. Never jump straight to scaling.

3. What should happen BEFORE the role officially starts (Phase 0)?

- **Lock a written mandate and start key relationships (correct)**
- Announce a company-wide AI policy
- Fire the skeptics
- Nothing — wait for day one

Why: Shape the mandate and seed relationships early so you start from strength.

20. Failure modes to avoid

The traps that quietly sink AI leads.

`8 min - 150 XP`

Knowing the anti-patterns is half the battle. These are the ways the role most often goes wrong.

Key ideas

1. Becoming the bottleneck — if every AI feature needs you, you've failed. Multiply through champions and paved roads.
2. Hype without substance — demos with no evals get found out. Always attach a number and a way to verify quality.
3. Ignoring governance until it bites — one PII-in-a-prompt incident can set the whole program back. Guardrails first.
4. Going non-technical too fast, or becoming the "AI police" who only says no (teams then route around you via shadow AI).
5. Boiling the ocean, selling with vanity metrics, and ignoring the human/fear dimension.

The big traps

— Bottleneck: centralizing all building instead of enabling teams.

— Hype: evangelizing without evals; you lose credibility when it breaks.

— Governance-last: in a regulated insurer, a data leak sets you back months.

— Losing your technical edge: standards stop landing when you can't build.

— AI police: only saying no drives shadow AI; be the easy yes path.

— Ocean-boiling: trying to transform every BU at once instead of sequencing.

— Vanity metrics & ignored fear: erode trust and create blockers.

Watch

- AI adoption pitfalls: hype vs reality in the enterprise

Do the work

- Audit your plans: where might YOU become the bottleneck? Remove it.
- Ensure every demo you give carries a number + a way to verify quality.
- Confirm guardrails exist before evangelizing a new capability.
- Check you're the "easy yes" path, not the AI police.

Check yourself

1. What is described as the #1 failure mode?

- Spending too little on cloud
- **Becoming the bottleneck (correct)**
- Reading too many books
- Hiring champions

Why: If every AI feature needs you, you've failed — multiply through champions and paved roads.

2. What happens if you become the "AI police" who only says no?

- Teams comply perfectly
- **Teams route around you with shadow AI (correct)**
- Governance improves automatically
- Adoption accelerates

Why: Only saying no drives shadow AI; be the easy, compliant yes path instead.

3. Select genuine failure modes from the list.

- **Hype without substance (no evals) (correct)**
- **Ignoring governance until it bites (correct)**
- Pairing every concept with building
- **Selling with vanity metrics (correct)**

Why: Pairing concepts with building is good practice — the others are traps.

21. Cadence & artifacts

The rhythm you run, and the portfolio of leverage you build.

`9 min - 150 XP`

A sustainable operating rhythm plus a set of reusable artifacts — each artifact scales you so you don't have to be everywhere.

Key ideas

1. Weekly: build something hands-on, touch base with champions + one new stakeholder, run office hours, post a short public update.
2. Biweekly: run the guild (demo + pattern + decision). Monthly: review your plan, check metrics, write one internal piece. Quarterly: impact report, red-team/governance review, roadmap refresh.
3. Keystone artifact: a reusable, CI-integrated eval harness.
4. Other artifacts: AI gateway/paved road, reference architecture, risk-triage template, prompt/agent library, security checklist, enablement curriculum, metrics dashboard, champion kit, governance handbook.

Cadence

— Weekly: hands-on build · champion 1:1s + one new stakeholder · office hours · short public update.

— Biweekly: guild / community of practice.

— Monthly: review the plan, check metrics, publish one internal write-up.

— Quarterly: honest impact report, red-team / governance review, roadmap refresh.

Artifacts to build (your leverage)

— Eval harness (keystone) · AI gateway / paved road · reference architecture.

— Use-case intake & risk-triage template · prompt/agent pattern library.

— AI security & red-team checklist · enablement curriculum · metrics dashboard.

— Champion onboarding kit · AI governance handbook v1.

Watch

- Running an engineering community of practice / guild

Do the work

- Set your weekly cadence (build, 1:1s, office hours, public update).
- Run the first biweekly guild meeting.
- Ship the eval harness as a shared internal artifact.
- Create a champion onboarding kit.
- Produce your first quarterly impact report.

Check yourself

1. Which is the "keystone" artifact?

· A prompt cheat sheet

• **A reusable, CI-integrated eval harness (correct)**

· A slide deck

· A Slack channel

Why: The eval harness is the keystone — it underpins quality and your authority.

2. What's the right primary cadence for the cross-BU community of practice (guild)?

- Daily
- **Biweekly (correct)**
- Once a year
- Never — async only

Why: A biweekly guild is the main scaling lever for sharing patterns and decisions.

3. Why build reusable artifacts at all?

- To look busy
- **Each one scales you so you don't have to be everywhere (correct)**
- To replace champions
- To avoid writing documentation

Why: Artifacts are leverage — they let teams move without you in the room.

Track 05 — Capstones & resources

Build the artifacts. Then keep learning.

22. Capstone: build an eval harness

Ship the keystone artifact — and your calling card.

`20 min - 250 XP`

Time to build. Create a reusable, CI-integrated eval harness for one real AI feature. This is the single most valuable artifact you can own as a lead: it makes quality measurable and becomes the standard other teams adopt.

Key ideas

1. Done means: a golden dataset, automated scoring, a CI gate that fails on regression, and a score-over-time view — reusable by other teams.
2. Ground it in error analysis on real outputs, not generic benchmarks.
3. Include cost & latency, not just quality.
4. Make it a template others can copy: clear README, swappable dataset and scorers.

Build steps

- Pick one AI feature and collect 50–100 representative inputs (include edge cases).
- Do error analysis; define 3–6 checks (assertions, rubric, or calibrated LLM-judge).
- Implement a runner that scores the dataset and outputs a report (quality, cost, latency).
- Add a CI job that fails the build when scores drop below a threshold.
- Write a README so another team can point it at their feature in under an hour.

Stretch goals

— Pairwise comparison vs a baseline; track win-rate.

— Pull a sample of production traffic into the dataset automatically.

Watch

- Why AI evals are the hottest new skill — Husain & Shankar
- Building an eval harness (promptfoo / Braintrust)

Do the work

- Collected 50+ real inputs with edge cases.
- Defined 3–6 checks from error analysis.
- Runner produces a quality/cost/latency report.
- CI gate fails on regression.
- README lets another team reuse it quickly.

Check yourself

1. What makes this harness valuable beyond your own feature?

· It's written in a trendy language

• **It's reusable — other teams adopt it, spreading measurable quality (correct)**

· It deletes old prompts

· It hides costs

Why: A reusable harness becomes the org standard — leverage, not a one-off.

2. What should the checks be grounded in?

· Public leaderboards

• **Error analysis of real outputs (correct)**

· The vendor's marketing

· Random sampling of the internet

Why: Real failure modes define the checks that matter for your task.

23. Capstone: a governed RAG feature

Grounded, access-aware, cited — production-shaped.

`22 min - 250 XP`

Build a small but production-shaped RAG feature over internal-style data: grounded answers with citations, access control at retrieval time, and an eval for retrieval quality. This exercises architecture, retrieval and governance together.

Key ideas

1. Done means: hybrid retrieval + grounded generation, citations on every answer, permission-aware retrieval, and a retrieval eval set.
2. Route the LLM call through a gateway-style wrapper (logging, cost, PII redaction) even if minimal.

3. Add graceful failure: if confidence/retrieval is poor, say 'I don't know' rather than hallucinate.
4. Document data residency and what data is indexed vs excluded.

Build steps

— Ingest a small corpus; chunk on structure with metadata (incl. an access tag).

— Implement hybrid retrieval (keyword + vector) + a re-ranker.

— Generate grounded answers WITH citations; refuse when retrieval is weak.

— Enforce access control at retrieval time (filter by the user's permissions).

— Add a retrieval eval (queries -> expected sources) and a faithfulness check.

Stretch goals

— Wrap calls in a minimal gateway (log tokens/cost, redact PII).

— Add a cost-per-answer metric.

Watch

- What is RAG? — IBM Technology
- Production RAG: citations & access control

Do the work

- Corpus ingested with structure-aware chunks + metadata.
- Hybrid retrieval + re-ranker working.
- Answers include citations and refuse when unsure.
- Access control enforced at retrieval time.
- Retrieval + faithfulness eval in place.

Check yourself

1. Why enforce access control at RETRIEVAL time?

· To speed up embeddings

• **So users can never be shown content they aren't permitted to see (correct)**

· To reduce token cost

· It's optional decoration

Why: Permission-aware retrieval prevents leaking data through the answer.

2. What should the feature do when retrieval is weak?

· Guess confidently

• **Gracefully say 'I don't know' rather than hallucinate (correct)**

· Return raw chunks

· Switch to a bigger model silently

Why: Graceful refusal beats confident hallucination, especially in a regulated setting.

24. Capstone: red-team & report

Break your own system, then standardize the fix.

18 min - 230 XP

Attack one of your AI features the way an adversary would, document what worked, and turn it into a reusable remediation checklist. This is how you earn security's trust and build a real guardrail standard.

Key ideas

1. Done means: a written report of attempted attacks, what succeeded, severity, and concrete remediations — plus a reusable checklist.
2. Cover the OWASP LLM basics: prompt injection (direct & indirect), data exfiltration, insecure output handling, excessive agency.
3. Test indirect injection via retrieved content / tool outputs, not just the user box.
4. Convert each finding into a guardrail and an eval check, then re-test.

Run the exercise

- Define targets: data exfiltration, unauthorized actions, policy bypass.
- Try direct & indirect prompt injection, jailbreaks, and tool abuse.
- Attempt to leak system prompt / hidden data / another user's data.
- Record severity and reproduction steps for each success.
- Add guardrails + eval checks; re-test to confirm closure.

Write it up

- Findings table: attack, result, severity, remediation, status.
- A reusable AI red-team checklist for future features.
- Share with security to co-own the standard.

Watch

- OWASP Top 10 for LLM Applications
- How to red-team an LLM application

Do the work

- Defined attack targets and scope.
- Tested direct + indirect injection and tool abuse.
- Documented findings with severity & repro steps.
- Added guardrails + eval checks and re-tested.
- Produced a reusable red-team checklist, shared with security.

Check yourself

1. Beyond the user input box, where must you test for prompt injection?

- Nowhere else
- **In retrieved content and tool outputs (indirect injection) (correct)**
- Only in the system prompt
- Only in the model weights

Why: Indirect injection rides in on documents and tool results the model reads.

2. What turns a red-team into lasting value?

- A one-time scary demo
- **Converting findings into guardrails, eval checks and a reusable checklist (correct)**
- Keeping results secret
- Blaming the model vendor

Why: Standardized, re-tested fixes — not theatre — are the deliverable.

25. Capstone: role charter & 90-day plan

Define the role before it defines you.

16 min - 220 XP

Pull the whole path together into the artifact that sets you up to succeed: a one-page role charter plus a 90-day plan and a metrics baseline. Bring it to your sponsor — leading the definition of your own role is the first proof you're right for it.

Key ideas

1. Done means: a one-page charter (mandate, scope, authority, metrics, operating model, backing, protected time) + a 30/60/90 plan + a baseline of 4–6 metrics.
2. Lock authority explicitly: what you can mandate (e.g. the paved road) vs advise, and your budget.
3. Agree 4–6 success metrics with your sponsor now, before you're judged on someone else's assumptions.
4. Sequence the 90 days: listen & quick win -> paved road -> scale via champions — don't jump to scaling.
5. Secure visible exec backing; it's what makes influence-without-authority work across BUs.

Write the charter (one page)

— Mandate & scope: which teams/BUs; own vs influence vs advise.

— Authority & budget: what you can mandate vs recommend.

— Success metrics: the 4–6 numbers you'll be judged on.

— Operating model: hub-and-spoke + champions endorsed.

— Backing & protected time: exec sponsorship; ~20–30% build time guarded.

Plan & baseline

— 30 days: listen, map, one quick win, stand up the guild.

— 60 days: eval harness + gateway POC, recruit champions, baseline metrics.

— 90 days: scale to 3–5 teams across >=2 BUs; first impact report.

— Baseline now: adoption, DORA, cost, eval scores — so you can show deltas.

Watch

- The first 90 days in a new role
- Negotiating role scope & expectations

Do the work

- Drafted a one-page role charter.
- Agreed 4–6 success metrics with your sponsor (in writing).
- Wrote a 30/60/90-day plan.
- Captured a baseline of your key metrics.
- Secured visible exec backing and protected build time.

Check yourself

1. Why draft the charter yourself instead of waiting for one?

· To avoid your sponsor

• **A fuzzy new role gets judged on private assumptions; defining it proves you fit it (correct)**

· Charters are legally required

· To delay starting

Why: Define the role or be defined by others' assumptions — and leading that is itself the proof.

2. What's the right 90-day sequence?

· Scale first, then listen

• **Listen & quick win -> paved road -> scale via champions (correct)**

· Governance only

· Hire a big team immediately

Why: Earn trust with a quick win, build the paved road, then scale through champions.

3. Why baseline metrics on day one?

· Busywork

• **So you can show before/after deltas and defend the role's impact later (correct)**

· Regulators require it

· To slow the rollout

Why: Without a baseline you can't prove the impact you create.

26. Learning resources

Depth over breadth — pair every concept with building something.

7 min - 100 XP

Curated, not exhaustive. Pick ~1 book/month, but pair every concept with shipping an artifact — you learn this role by building, not just reading.

Key ideas

1. Leadership: The Staff Engineer's Path (Reilly) first; Staff Engineer (Larson) + lethain.com; Influence Without Authority; Team Topologies; Switch / Made to Stick.
2. AI engineering & evals: AI Engineering (Chip Huyen); Designing ML Systems; Anthropic's "Building effective agents"; Hamel Husain on evals; try promptfoo / Braintrust / LangSmith.
3. Security & responsible AI: OWASP Top 10 for LLM apps; NIST AI RMF; ISO/IEC 42001.
4. Governance: EU AI Act + insurance guidance; DORA overview; your own company's model-risk & privacy policies.
5. Architecture: Designing Data-Intensive Applications (Kleppmann); Fundamentals of Software Architecture.

Read with intent

Consume ~1 book/month, but always pair a concept with building something — the eval harness, the gateway POC, a reference implementation.

Follow practitioners

— Simon Willison (simonwillison.net), Latent Space, Eugene Yan, Chip Huyen, the Anthropic engineering blog.

— Will Larson (lethain.com) for staff-level leadership and org design.

— Hamel Husain for practical, opinionated LLM evals.

Watch

- The Staff Engineer Mindset with Tanya Reilly — O'Reilly

Do the work

- Start The Staff Engineer's Path (Reilly).
- Start AI Engineering (Chip Huyen).
- Subscribe to 2–3 practitioner blogs/newsletters.
- Try one eval framework hands-on (promptfoo / Braintrust / LangSmith).

Check yourself

1. What's the recommended way to actually consume these resources?

· Read everything before doing anything

• **Pick ~1 book/month and pair every concept with building an artifact (correct)**

· Only watch videos

· Memorize benchmark scores

Why: You learn the role by building; reading alone doesn't transfer.

2. Which book is recommended FIRST for technical leadership without management authority?

- Designing Data-Intensive Applications
- **The Staff Engineer's Path (Reilly) (correct)**
- AI Engineering
- Switch

Why: Reilly's book is the canonical starting point for staff-level, influence-based leadership.