PROTOTYPE · HARDEN · SHIP

Vibe Coding to Production Readiness

A **40-point technical checklist** to convert prototype LLM prompts and rapid AI-built mockups into **production-hardened systems** — secure, tested, observable and affordable to run.

prototype.py

```
key = "sk-live-..."
r = llm(f"{user_input}")
data = eval(r)
db.exec(f"...{id}")
# TODO: auth?
# TODO: tests
```

"works on my demo"



40 CHECKS

PRODUCTION

- Secrets in a vault
- Auth on every endpoint
- Validated JSON outputs
- Evals gate every release
- Tokens, cost & errors traced
- Fallbacks & kill switch

ready for real users

40
checks, each with a concrete "proof" you can verify

8
domains from code ownership to compliance

25
must-pass items before real users touch it

30 days
typical hardening plan for a working prototype

WHY THIS CHECKLIST EXISTS

Demo-ready is not production-ready

AI coding tools and LLM prompts let a small team build a convincing product in days. The gap between that demo and a system that survives real users, real attackers and a real invoice from your model provider is where most AI products stall.

DIMENSION	VIBE-CODED PROTOTYPE	PRODUCTION SYSTEM
Who understands the code	The AI that wrote it	A named human owner per module
Secrets	Pasted into source files	Vault, rotated, never in Git
Access control	Hidden buttons in the UI	Checked on the server for every request
Prompts	Inline f-strings, edited live	Versioned templates with eval scores
Model output	Trusted and rendered directly	Schema-validated, sanitised, grounded
Failure handling	Crash or spinner forever	Timeouts, retries, fallbacks, clear messages
Visibility	console.log	Traces, token cost, alerts, dashboards
Cost	Unknown until the bill arrives	Budget per user, caching, model routing

HOW TO USE IT

1 **Audit**
Walk the 40 checks with the code open. Mark each pass only when its **proof** exists.

2 **Score**
Use the scorecard on page 10. All 25 **Must** items are launch blockers.

3 **Harden**
Follow the 30-day plan on page 11, worst domain first.

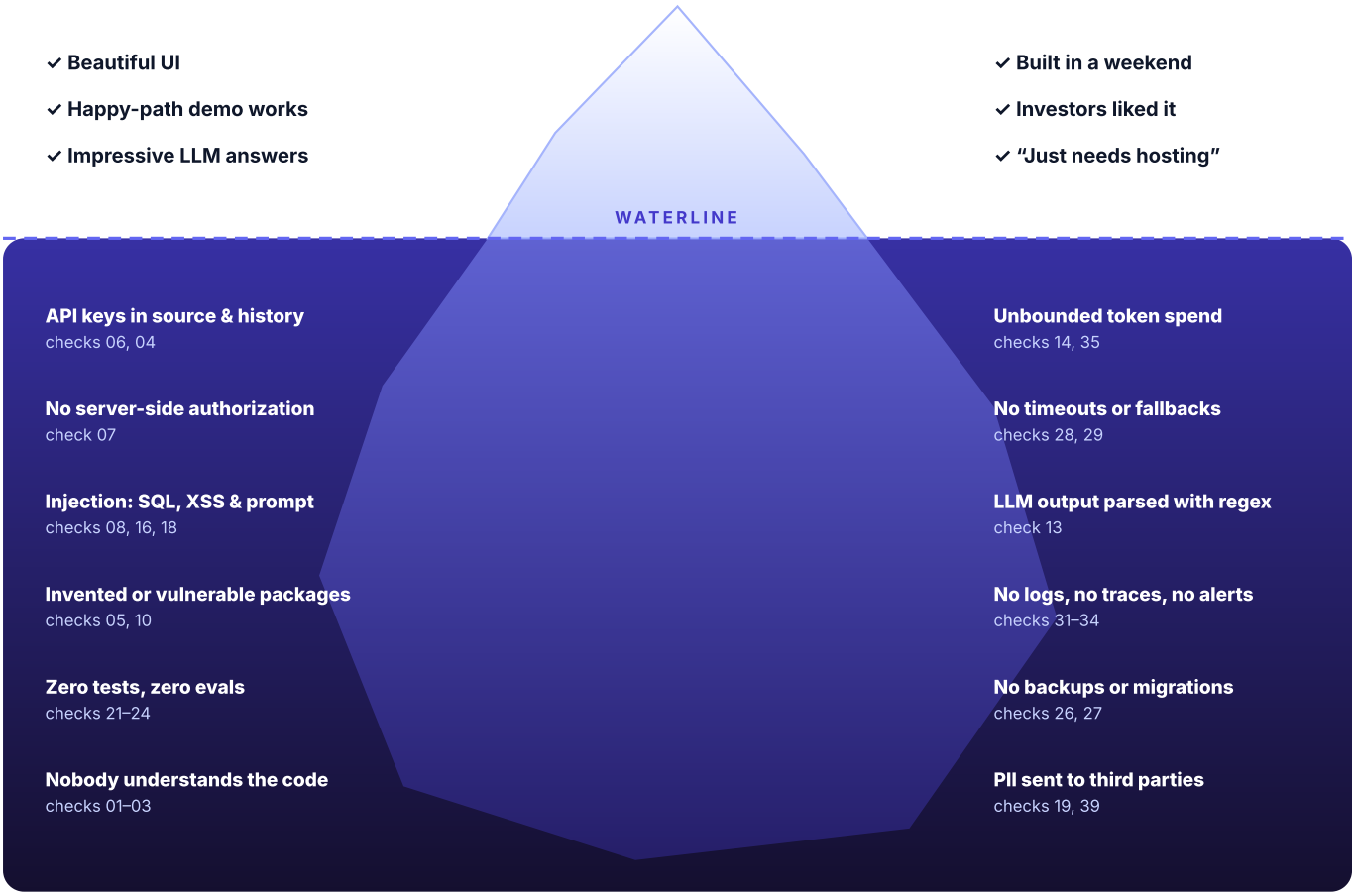
THE 8 DOMAINS

01 Code & architecture
02 Security
03 LLM & prompt engineering
04 AI safety & guardrails
05 Testing & evaluation
06 Data & reliability
07 Observability & cost
08 Delivery & compliance

THE HIDDEN DEBT MAP

What the demo doesn't show you

Everything above the waterline works in a demo. Everything below it is what we find when we audit AI-built prototypes before launch.



Why it happens

AI assistants optimise for code that runs now. Security, failure handling and cost only show up later — under real load and real attackers.

Why it's dangerous

LLM features add new attack paths — prompt injection, data leaks through context, runaway spend — that classic code review misses.

The fix

Keep the prototype's speed, add the missing layers systematically — the next four pages list exactly what, and how to prove it.

Our experience: hardening a working prototype is usually cheaper than rewriting it — but only if you do it **before** launch. Fixing auth, data models and secret leaks after real users arrive costs far more, in money and in trust.

01 Code & architecture

Make AI-written code something your team actually owns.

01 Everything in Git, behind pull requests ☐ MUST

Branch protection on main, at least one human review per PR, no direct pushes.

Proof: repo settings show protected main; PR history exists

02 Every generated file has a human owner ☐ MUST

Someone has read it, understands it and deleted the dead code and duplicate helpers the AI left behind.

Proof: CODEOWNERS file; no unused modules in a dead-code scan

03 Typed, linted and formatted in CI ☐ SHOULD

TypeScript strict / mypy / Ruff / ESLint catch the class of bugs LLMs produce most: wrong shapes and silent nulls.

Proof: CI fails on a type or lint error

04 Configuration lives in the environment ☐ MUST

No hard-coded URLs, keys, model names or feature switches; one build runs in dev, staging and prod.

Proof: grep for http:// and API keys returns nothing

05 Dependencies pinned, real and licensed ☐ MUST

Lockfile committed; every package verified to exist and be the intended one (AI can invent names attackers register); licences reviewed.

Proof: lockfile + licence report in CI

02 Security

Prototypes trust everyone. Production trusts no one.

06 Secrets out of code and Git history ☐ MUST

Keys moved to a secrets manager, rotated, and purged from history — assume anything once committed is leaked.

Proof: secret scanner (e.g. gitleaks) clean on full history

07 Proven authentication, server-side authorization ☐ MUST

Use an established auth provider; check permissions on every endpoint and every object, never only in the UI.

Proof: test: user A cannot read user B's record by changing an ID

08 Validate input, encode output ☐ MUST

Schema-validate every request; parameterised queries; escape anything rendered — including LLM output shown as HTML.

Proof: SQLi / XSS test cases in the suite

09 Edge protection on ☐ SHOULD

HTTPS only, strict CORS, security headers, rate limits per user and IP, request size limits.

Proof: security-headers scan grade A; 429s under abuse test

10 Automated security scanning in CI ☐ SHOULD

Dependency audit, SAST and container scans run on every PR with a fail threshold.

Proof: Dependabot / Semgrep / audit jobs visible in CI

03 LLM & prompt engineering

Treat prompts and models like code: versioned, pinned and tested.

11 Prompts are versioned templates, not inline strings ☐ MUST

System prompts live in files with IDs and change history, so you know which prompt produced which answer.

Proof: prompt files in repo; prompt version logged per call

12 Model versions pinned, with a fallback ☐ MUST

Use dated model IDs rather than moving aliases; define a fallback model or provider for outages.

Proof: config shows pinned IDs + tested fallback path

13 Structured output, validated ☐ MUST

Ask for JSON against a schema, validate it, retry or repair on failure — never parse free text with regex.

Proof: schema validation errors are logged and counted

14 Context and token budgets enforced ☐ SHOULD

Cap input size, truncate or summarise history, cap max output tokens per feature.

Proof: no request exceeds the documented token budget

15 Parameters chosen per task ☐ SHOULD

Low temperature for extraction and classification; higher only for creative features; documented per prompt.

Proof: parameters stored alongside each prompt template

04 AI safety & guardrails

Assume the model will be tricked — limit what a tricked model can do.

16 Prompt-injection defences ☐ MUST

User text, web pages, files and retrieved docs are data, never instructions; separate them clearly and test known attacks.

Proof: injection test set passes in CI

17 Least-privilege tools, human confirm on side effects ☐ MUST

The model gets only the tools it needs; payments, deletions and outgoing messages require explicit user confirmation.

Proof: tool allowlist per feature; confirmation step in UX

18 No raw SQL, shell or URLs from the model ☐ MUST

Model output fills parameters of safe functions; it never runs as code or chooses arbitrary endpoints.

Proof: code review: no eval/exec of model output

19 Moderation and PII handling ☐ SHOULD

Screen inputs and outputs for abuse; redact personal data before it reaches logs or third-party models.

Proof: redaction unit tests; moderation events logged

20 Grounded answers, honest refusals ☐ SHOULD

For factual features use retrieval with citations, and let the model say "I don't know" instead of inventing.

Proof: eval measures citation accuracy and refusal rate

05 Testing & evaluation

A demo proves it can work. Tests prove it keeps working.

21 Unit and integration tests on critical paths ☐ MUST

Auth, payments, data writes and core business rules covered; AI-generated tests reviewed — they often test nothing.

Proof: coverage report on core modules; tests fail when logic breaks

22 End-to-end tests for top user journeys ☐ SHOULD

Signup, the main workflow and checkout run in a real browser against staging on every deploy.

Proof: Playwright / Cypress suite green in pipeline

23 A golden-set eval suite for every LLM feature ☐ MUST

50–200 real examples with expected outcomes and a pass threshold, scored automatically.

Proof: eval score reported in CI, release blocked below threshold

24 Re-run evals on every prompt or model change ☐ MUST

A one-word prompt tweak can break a feature that worked yesterday; evals catch it before users do.

Proof: prompt/model diff triggers eval job

25 Load test at 2× expected peak ☐ SHOULD

Include LLM latency and provider rate limits — they are usually the first ceiling you hit.

Proof: report with p95 latency and error rate at 2× load

06 Data & reliability

The LLM API will be slow or down some days. Plan for it.

26 Managed database with versioned migrations ☐ MUST

No manual schema edits; every change is a migration file applied by CI.

Proof: migration history matches production schema

27 Backups with a tested restore ☐ MUST

Automated backups, defined RPO/RTO, and a restore actually performed in the last 30 days.

Proof: dated restore drill record

28 Timeouts, retries and circuit breakers ☐ MUST

Every external call — especially LLM calls — has a timeout, retries with backoff and jitter, and a breaker.

Proof: fault-injection test: provider timeout handled

29 Graceful degradation ☐ SHOULD

When the model is unavailable, show cached results, a simpler path or a clear message — never a blank screen.

Proof: kill switch test shows fallback UX

30 Long jobs run async, idempotently ☐ SHOULD

Queue multi-step LLM work, stream partial results, and make retries safe with idempotency keys.

Proof: duplicate job submission produces one result

07 Observability & cost

If you can't see tokens, latency and errors, you can't run it.

31 Structured logs with request IDs ☐ MUST

JSON logs that follow one request from browser to API to model and back.

Proof: one request ID traceable across all services

32 Error tracking on front end and back end ☐ MUST

Every exception captured with stack trace, release version and user context.

Proof: Sentry (or similar) alerts reach the team

33 LLM tracing per call ☐ MUST

Log prompt version, model, input/output tokens, latency, cost and outcome for every call (with PII redacted).

Proof: trace view in Langfuse / OpenTelemetry / provider console

34 SLO dashboards and alerts ☐ SHOULD

Availability, p95 latency, error rate and eval score on one screen, with alerts that page a human.

Proof: alert fired and acknowledged in a drill

35 Cost budgets and controls ☐ MUST

Spend alerts, per-user caps, response and prompt caching, and routing simple tasks to cheaper models.

Proof: cost per active user tracked weekly

08 Delivery & compliance

Ship safely, roll back instantly, and stay on the right side of the law.

36 CI/CD with preview environments and one-click rollback ☐ MUST

Every merge deploys automatically; any release can be reverted in minutes.

Proof: rollback performed in a drill

37 Staging that mirrors production, defined as code ☐ SHOULD

Infrastructure in Terraform/Pulumi or platform config; no hand-built servers.

Proof: staging rebuilt from code in under an hour

38 Feature flags and an AI kill switch ☐ SHOULD

Roll AI features out gradually and turn them off without a deploy.

Proof: flag toggled in staging and prod

39 Privacy and data-processing in order ☐ MUST

Privacy policy, consent, retention rules, provider data-processing terms and model-training opt-outs checked (GDPR, DPDP, etc.).

Proof: signed DPAs; data map documented

40 Runbooks and an incident process ☐ SHOULD

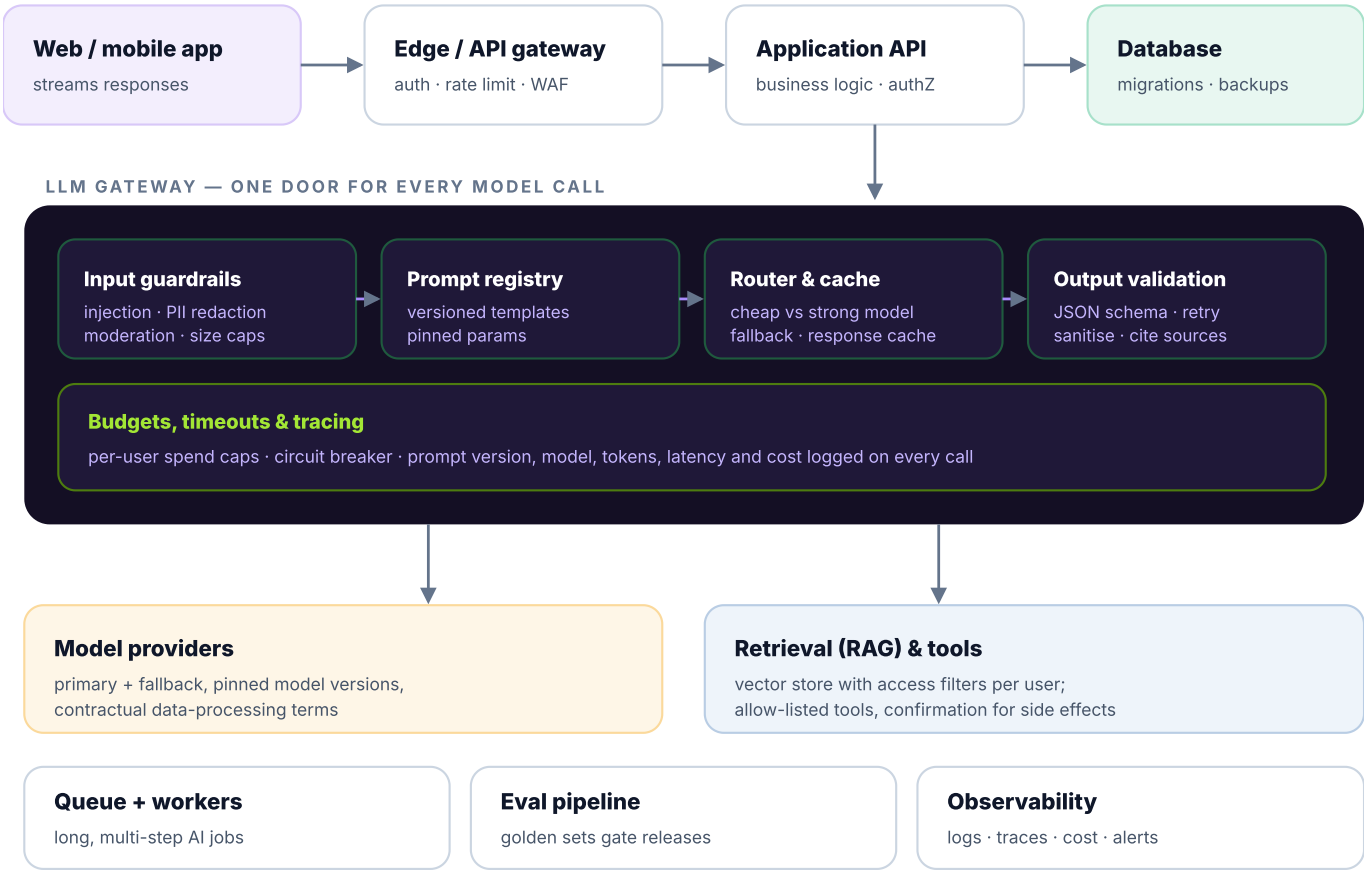
Architecture doc, on-call rota, top-5 failure runbooks and a post-incident template.

Proof: runbooks linked from alerts

REFERENCE ARCHITECTURE

A production LLM application

The key addition over a prototype: every model call passes through one **LLM gateway** layer that owns routing, guardrails, caching, budgets and tracing — so features never call a provider directly.



Why a gateway?

One place to switch models, add guardrails or cap spend — without touching every feature.

Build or buy?

Start with a thin internal module; adopt an open-source or managed gateway as traffic and model count grow.

RAG security

Filter retrieved documents by the user's permissions **before** they reach the prompt.

LOG THIS FOR EVERY MODEL CALL

FIELD	WHY YOU NEED IT	CHECK
Request ID, user ID, feature	Trace one complaint back to the exact call	31
Prompt ID + version, model ID	Know which change caused a quality drop	11, 12
Input / output tokens, cost	Cost per feature and per user; spot abuse	35
Latency, fallback, validation & guardrail flags	Provider health, SLOs, failures and attacks over time	13, 16, 28

FROM PROTOTYPE TO PRODUCTION CODE

The same LLM call, hardened

× PROTOTYPE

```
r = client.chat(
  model="latest",
  messages=[{"role": "user",
    "content": f"Extract the
    invoice data: {doc}"}])
data = json.loads(r.text)
db.save(data)
```

What's wrong

- Moving model alias (12)
- Prompt inline & unversioned (11)
- Untrusted doc mixed into instructions (16)
- No schema, crash on bad JSON (13)
- No timeout, retry or fallback (28)
- No trace, tokens or cost (33)

✓ PRODUCTION

```
from pydantic import BaseModel, ValidationError
from app.llm import gateway, prompts
from app.obs import trace

class Invoice(BaseModel):
    vendor: str
    invoice_no: str
    total: float
    currency: str

PROMPT = prompts.get("invoice_extract", version="v7")

@trace("invoice_extract")      # tokens, cost, latency
async def extract_invoice(doc: str, user) -> Invoice:
    msgs = PROMPT.render(document=doc)  # doc as DATA
    for attempt in range(2):
        r = await gateway.complete(
            msgs,
            models=["primary-2026-05",
                    "fallback-2026-03"], # pinned
            response_schema=Invoice,
            temperature=0, max_tokens=400,
            timeout=20, user_id=user.id,  # budget cap
        )
        try:
            return Invoice.model_validate_json(r.text)
        except ValidationError as e:
            msgs = PROMPT.repair(msgs, r.text, str(e))
    raise ExtractionFailed("needs human review")
```

A GOLDEN-SET EVAL THAT BLOCKS BAD RELEASES

TESTS/EVALS/TEST_INVOICE_EXTRACT.PY

```
import json, pytest
CASES = [json.loads(l) for l in open("evals/invoice_golden.jsonl")]   # 120 real, anonymised invoices

@pytest.mark.eval
async def test_extraction_accuracy():
    passed = 0
    for case in CASES:
        out = await extract_invoice(case["document"], TEST_USER)
        passed += out.model_dump() == case["expected"]
    score = passed / len(CASES)
    assert score >= 0.95, f"eval score {score:.1%} below 95% gate"     # CI fails → no deploy
```

Prompt as a file

Template, parameters, schema and owner live together in `prompts/invoice_extract/v7.yaml`.

Data, not instructions

Untrusted text goes in a clearly delimited block the system prompt says to treat only as content.

Fail to a human

After validation retries, route to a review queue — never save half-parsed data.

Function and model names are illustrative; the pattern works with any provider SDK or gateway library.

READINESS SCORECARD

Score your system in one page

Tick a box only when the proof exists. **Red numbers** are Must items — any unticked Must is a launch blocker, whatever the total.

DOMAIN	CHECKS					SCORE
Code & architecture	01 ☐	02 ☐	03 ☐	04 ☐	05 ☐	__ / 5
Security	06 ☐	07 ☐	08 ☐	09 ☐	10 ☐	__ / 5
LLM & prompt engineering	11 ☐	12 ☐	13 ☐	14 ☐	15 ☐	__ / 5
AI safety & guardrails	16 ☐	17 ☐	18 ☐	19 ☐	20 ☐	__ / 5
Testing & evaluation	21 ☐	22 ☐	23 ☐	24 ☐	25 ☐	__ / 5
Data & reliability	26 ☐	27 ☐	28 ☐	29 ☐	30 ☐	__ / 5
Observability & cost	31 ☐	32 ☐	33 ☐	34 ☐	35 ☐	__ / 5
Delivery & compliance	36 ☐	37 ☐	38 ☐	39 ☐	40 ☐	__ / 5

Total: ___ / 40 · Must items passed: ___ / 25

READING YOUR SCORE

0–19 · Prototype	20–29 · Pilot	30–35 · Launch	36–40 · Scale
Demo only. Keep it away from real data.	Small trusted group, if all Musts pass.	Public launch.	Ready to grow.

Most common blockers we find

- 06 · live keys in Git history
- 07 · authorization only in the front end
- 16 · retrieved content treated as instructions
- 23 · no eval set — quality judged by “looks right”
- 35 · no spend caps on public endpoints

Quick wins (under a day each)

- Turn on secret scanning and rotate keys
- Pin model versions and set max tokens
- Add timeouts to every external call
- Install error tracking front and back
- Set a provider budget alert

HARDENING ROADMAP

From prototype to production in 30 days

A typical plan for a working prototype with a handful of AI features. Security first — it is the only domain where a single miss can end the product.



WEEK 1
Lock it down
Audit, Git hygiene, rotate secrets, server-side auth, input validation.

WEEK 2
Tame the model
Prompt registry, pinned models, schemas, injection defences, first eval set.

WEEK 3
Make it reliable
Tests, timeouts, fallbacks, backups, tracing, cost dashboards.

WEEK 4
Ship it safely
CI/CD, flags, load test, privacy review, runbooks, staged rollout.

Keep vibe coding — with rails

- AI tools stay for speed; humans own review and merge
- Project rules file tells the assistant your standards
- Every AI change goes through the same CI and evals
- Security-sensitive code always gets a senior reviewer

Launch-day definition of done

- All 25 Must checks have proof
- Eval score above gate for every AI feature
- Rollback and kill switch tested this week
- Spend alerts and on-call rota live

EXAMPLE TOOLING BY DOMAIN

DOMAIN	COMMON CHOICES (PICK WHAT FITS YOUR STACK)
Code & CI	GitHub / GitLab, branch protection, Ruff, ESLint, TypeScript strict, pre-commit
Security	gitleaks, Dependabot, Semgrep, cloud secret managers, managed auth providers
LLM & evals	Pydantic / Zod schemas, prompt registry in repo, pytest-based evals, Langfuse, promptfoo
Reliability & ops	Managed Postgres, queues, Sentry, OpenTelemetry, Grafana, k6, feature-flag service

Steigern Tech

STEIGERNTECH.COM

HARDEN IT WITH US

Built a prototype with AI? Let's make it production-ready.

Steigern Tech audits, hardens and scales AI-built apps and LLM features — keeping the speed you gained, removing the risk you inherited.

40-point readiness audit

We score your code against this checklist and deliver a prioritised fix plan with effort estimates.

Security & code hardening

Secrets, auth, validation, dependency hygiene and refactoring of AI-generated code your team can own.

LLM engineering

Prompt registries, structured outputs, RAG, guardrails, eval pipelines and cost optimisation.

Cloud, CI/CD & observability

Pipelines, staging, monitoring, tracing and on-call — so launch day is boring.

WHATSAPP

+91 94275 72571

EMAIL

info@steigerntech.com

WEBSITE

steigerntech.com

© 2026 Steigern Tech LLP. Checklist provided as engineering guidance, not legal or security certification.

Product and model names are trademarks of their respective owners.