Paid trial proposal

Production Change Gate for CoreFlow

Turn one scary pending change into a boring, evidence-backed rollout.

CoreFlow's public engineering thesis is that implementation is increasingly AI-written, while the real bottlenecks are context, verification, and closed-loop execution. My proposed paid trial is a small production-readiness system that ingests the context around a risky change and outputs a rollout recommendation, explicit autonomy boundary, and evidence bundle.

Choose one scenario

Scenario	Risk shape	Trial output
Model or route-policy promotion	Reversible behavioural change with user-facing blast radius	Shadow / canary plan, guardrails, rollback conditions
LoRA or model-adapter rollout	Quality drift, latency change, cost drift	Promotion checklist plus evaluation bundle
GPU saturation or provider failover	Capacity, latency, and cost incident	Failover playbook and degraded-mode rules
Stateful backfill or data migration	Irreversible operational change	Human-approval boundary and staged execution plan

Deliverables

- Working console or CLI aligned to your stack where appropriate.

- Risk classification for reversible vs irreversible changes.

- Context ingestion for repo state, metrics, logs, incidents, and product analytics (real, redacted, or mocked).

- Verification gates: compile / lint, behavioural evals, shadow traffic, canary, rollback requirements.

- Evidence bundle with rollout plan, watch metrics, blocking conditions, and operator notes.

- Short architecture note plus demo walkthrough.

Success criteria

- Recommendations change when critical context or gates are removed.

- Autonomy boundary is explicit: closed-loop candidate / human approval required / block.

- At least one route-promotion case and one incident / failover case are handled end-to-end.

- Output surfaces the metrics that matter here: p95 / p99 latency, error rate, queue depth, cost per generation, session completion, abandonment.

- The artefact is reusable after the trial rather than a throwaway demo.

Indicative 4-day plan

Day	Focus	Output
1	Align on scenario, acceptance criteria, and available context	A crisp trial brief and success definition
2	Build adapters for context ingestion and risk model	Working inputs from repo / metrics / logs / incidents

Day	Focus	Output
3	Implement gates, evidence bundle, and rollout logic	Recommendations plus autonomy boundary
4	Harden, document, and demo	Demo, README, and next-step recommendations

Technical sketch

Lightweight console in TypeScript / Next.js style with a workflow layer for evidence generation. Connectors can be real, redacted, or mocked. The goal is judgement and reusable shape, not pretending to rebuild your stack in miniature.

What CoreFlow gets

A working artefact, an explicit rollout pattern, and a view into how I reason about latency, model behaviour, incidents, and operator boundaries under incomplete information.

Why this is the right trial

It matches your public engineering thesis. It shows how I reason under incomplete information, connect model behaviour to user-facing metrics, and reduce operational risk without slowing iteration to a crawl. Most importantly, it leaves behind a small piece of machinery the team can keep using.

Preferred starting points: route-policy promotion or GPU failover. Both reveal the most about engineering judgement.