A Pattern Language for Production LLM Platforms: Governed Routing, Agent Orchestration, and AI-Native Delivery in Regulated Environments

Nabeel A. Khan
iSystematic
https://nabeelkhan.com
ORCID: 0009-0005-5364-914X

Preprint. This version is a working draft and has not been peer reviewed.

Abstract

Most accounts of large language model (LLM) systems stop at the demonstration. Production in a regulated institution begins where the demonstration ends, at the moment a routing decision becomes a billing event, an agent action becomes an audit record, and a deployment becomes a governed change. This paper presents a cross-layer reference architecture and pattern language for production LLM platforms, organized into three layers: an infrastructure layer that routes and observes inference under explicit service-level objectives; an application layer that orchestrates specialized agents through planner, executor, verifier, and generator roles and through inspectable activity-on-vertex graphs under tiered human oversight; and an operations layer that ships models, agents, and code on a single governed internal platform. A single invariant unifies the layers: a boundary is a clause the optimizer may not cross, and everything else is optimization. We catalog seventeen recurring patterns, define the artifacts that carry them, and trace one illustrative cross-layer incident to show how the patterns compose end to end. We add guidance for selecting patterns by environment, a catalog of characteristic failure modes, and an evaluation agenda that states each pattern's prediction as a falsifiable hypothesis for future empirical work. The contribution is conceptual and architectural rather than empirical. The scenario used throughout is a fictional reference scenario and reports no measurements; every claim is drawn from the published literature and from architectural reasoning, not from a controlled study.

Keywords: software engineering; large language model systems; reference architecture; design patterns; AI governance; agent orchestration; LLM serving; platform engineering.

ACM/arXiv categories: cs.SE (primary); cs.DC; cs.AI.

1 Introduction

A working demonstration of a language model answers a question and turns a pipeline green. A production system in a regulated institution answers the same question under a different set of obligations. The institution must be able to say which model produced an answer and why, what it cost, what data it touched, whether a human reviewed it, and how the decision would be reconstructed for an auditor a year later. The distance between the demonstration and that

obligation is not a matter of scale alone. It is a change in what the system is for. The demonstration optimizes a response. The production system governs a decision.

This paper treats that distance as an architectural problem and proposes a pattern language for crossing it. The setting is a regulated financial institution, because regulation makes explicit the constraints that exist implicitly in any serious deployment: traceability, accountability, data residency, and the requirement that an automated decision can be explained and defended. We use a fictional institution, Nebula Financial, as an illustrative reference scenario throughout. Nebula Financial is a teaching construct. It deploys no system, runs no benchmark, and reports no measurement. Its role is to give the patterns a concrete pressure to resolve, not to supply evidence.

The architecture spans three layers that correspond to three questions a platform team must answer in sequence. The infrastructure layer answers which model may serve a request, under which latency, cost, and risk budget, and how that decision is recorded. The application layer answers how specialized agents are composed into a workflow that can be inspected, reviewed, and improved without becoming opaque. The operations layer answers how models, agents, and code reach production on a single internal platform that holds authority by degrees rather than all at once. The three layers map onto the progression from infrastructure to application to operations and are shown in Figure 1.

One invariant runs through every layer and unifies the language:

> A boundary is a clause the optimizer may not cross. Everything else is optimization.

The statement separates the two kinds of decision a production platform makes. An optimization decision improves an objective: lower latency, lower cost, higher quality, fewer tests run. A boundary decision fixes a constraint the optimizer is not permitted to relax for any gain: a residency rule, a least-privilege scope, a human-review threshold, an evidence record that must exist. The patterns in this paper are, with few exceptions, mechanisms for stating a boundary so precisely that an optimizer can be set loose inside it. This is what makes the resulting system both efficient and accountable. The optimizer is free, and the boundary is fixed, and the two never compete because they are different kinds of clause.

Contributions. This paper makes five contributions.

1. A three-layer reference architecture for governed, production LLM platforms in regulated settings, organized so that each layer hands a governed artifact to the next.
2. A pattern language of seventeen named patterns, with the boundary-versus-optimization invariant stated once and applied consistently across the layers.
3. An auditability model in which decision records and evidence chains are first-class architectural concerns rather than instrumentation added after the fact.
4. A worked, explicitly illustrative cross-layer composition, in which a single incident travels through all three layers, showing how the patterns interact.
5. Guidance for applying the language: a selection of patterns by environment, an adoption-stage progression, a catalog of characteristic failure modes, and an evaluation agenda of falsifiable per-pattern hypotheses for future empirical work.

Scope and stance. The contribution is a reference architecture and a pattern catalog, not an empirical evaluation. We make no performance claim, report no benchmark, and present no

measured result. Where a number appears, it is a worked illustration and is labeled as such. This stance is deliberate. Presenting a fictional scenario as if it were measurement would misrepresent the work. The patterns earn their place by resolving a stated set of forces under a stated set of constraints, and by their grounding in the cited literature, not by an experiment this paper does not run. Section 10 states the resulting threats to validity plainly.

2 Background and Related Work

The patterns draw on three bodies of recent work, one per layer, together with the established disciplines of site reliability engineering and platform engineering.

Routing and serving. Model routing has moved from a static rule to a learned decision. Universal model routing frames the choice of model as an inference-efficiency problem across a heterogeneous pool [9]; workload-aware routing balances load under production constraints [8]; and uncertainty-based routing escalates only when a smaller model is not confident enough to answer [4]. On the serving side, speculative decoding has become a latency primitive rather than a research curiosity. Adaptive, service-level-objective-aware speculation adjusts the draft strategy to a latency target [7]; suffix-based and long-context variants extend the idea to new request shapes [18, 22]; pipelined variants carry it into distributed inference [15]; and the technique is surveyed comprehensively in Xia et al. [26]. Serving efficiency for multi-model applications [5] and the broader inference serving landscape [12] frame the cost surface the infrastructure layer governs. The router itself is an attack surface with a lifecycle, and its vulnerabilities motivate treating promotion as a security boundary [14].

Agent orchestration. A single agent becomes a system when its parts are separated and its workflow is made explicit. In-the-flow agentic optimization trains a planner from the record of what the system did [13]; modular software-engineering agents demonstrate the value of separating roles [2]; and modularized and automatically generated agentic workflows treat the workflow as a structured, refinable graph rather than as a script [17, 27]. Human-in-the-loop orchestration, treated as a runtime layer with explicit tiers and confidence-driven escalation rather than as a manual fallback, supplies the oversight model the application layer adopts [11].

AI-native delivery. The operations layer builds on platform engineering and on the recent move to put agents inside the delivery pipeline. Systematic treatments of platform engineering and developer experience frame the internal developer platform as a product [20], a framing echoed in industry practice [19, 21] and in community surveys of the field [23, 25]. AI-augmented continuous integration and delivery introduces autonomous decisions into the pipeline [3, 10]; reinforcement-framed test selection treats the pipeline as a decision process [24]; and evaluation of agents against real continuous-integration failures supplies the discipline of measuring an agent before granting it authority [6, 16]. A local-first command center for the AI-augmented workflow illustrates the direction of the developer surface [1].

Position. This paper does not extend any one of these results. It composes them. Each cited line of work answers a question inside a single layer. The contribution here is the architecture that joins the layers under one invariant, and the pattern language that lets a practitioner carry a decision from a routing policy to an audit record to a governed deployment without losing the thread.

Figure 1: The three-layer reference architecture. Infrastructure governs which model serves a request and records the decision; application composes agents into an inspectable, reviewable workflow; operations ships models, agents, and code on a single governed platform. The same governance invariant holds at every layer. Figure adapted from the companion book series in its single-panel light form.

3 Architecture Overview and the Governance Invariant

The three layers are not a stack of independent systems. They are one control path seen at three altitudes. A request enters at the infrastructure layer, where a routing decision selects a model under a budget and writes a routing record. If the task is more than a single call, it enters the application layer, where a workflow of agents plans, acts, verifies, and generates under contracts and human review, writing a trajectory record. When the institution decides to change the system itself, a new model, a new agent, a new policy, the change enters the operations layer, where it passes through a governed pipeline that grants authority by degrees and writes a delivery record. The three records are the same idea at three altitudes: a decision, its justification, and the evidence that it happened.

The governance invariant gives each layer the same internal shape. At every layer, the design separates a boundary from an optimizer.

- At the infrastructure layer, the boundary is the routing policy and the residency and risk constraints it encodes. The optimizer is the routing model that chooses, within that policy, the cheapest model that will meet the quality and latency target.
- At the application layer, the boundary is the capability contract and the human-review threshold. The optimizer is the planner that decides, within those limits, how to decompose and execute a task.
- At the operations layer, the boundary is the delivery guardrail and the trust tier. The optimizer is the pipeline that decides, within its authority, which tests to run and when to act without waiting for a person.

The separation invites an objection: that some boundaries shift over time, and that an optimizer

might even propose new ones. The separation survives both. A boundary may evolve, but only through the same governed promotion any policy change follows, never through the optimizer relaxing it at run time for a local gain. When a boundary is loosened or tightened, that change is itself a reviewed, recorded act, and the optimizer then operates against the new boundary exactly as it did against the old. An optimizer may also surface a candidate constraint it infers would reduce risk, but a candidate is a proposal, not a boundary, until it passes through promotion and is recorded. The invariant is therefore not that boundaries are static, but that they move only by governance and never by optimization.

This shape is why the architecture scales without becoming unaccountable. Raising a system's autonomy is a matter of widening what the optimizer may do, never of relaxing a boundary. A team raises authority as fast as it can prove judgment, and the proof is recorded in the same evidence the boundary requires. The remainder of the paper states the patterns layer by layer, using a uniform template: the context in which the pattern applies, the problem it resolves, the forces in tension, the solution as an artifact or mechanism, and the consequences of adopting it.

4 Infrastructure Patterns

The infrastructure layer turns a model call into a governed event. Its patterns make the routing decision explicit, bounded, fast, observable, and secure.

IP-1: Governed Routing Policy

Context. An institution runs a pool of models that differ in capability, cost, latency, and data-handling guarantees, and a stream of requests that differ in sensitivity and difficulty. *Problem.* A routing rule embedded in application code cannot be reviewed, versioned, or audited, and it mixes a business constraint with an optimization choice. *Forces.* The institution wants the cheapest adequate model for each request; it also requires that some requests never leave a jurisdiction or never reach a particular model class, and that the reason for every route can be reconstructed. *Solution.* Express routing as a declarative, versioned policy artifact separate from the application. The policy states the hard constraints as boundaries (residency, permitted model classes per data tier) and exposes the soft objective (cost and latency minimization subject to a quality floor) to a learned router [4, 8, 9]. The policy is promoted through change control, not edited in place. *Consequences.* The routing decision becomes inspectable and governable. The optimizer is free to learn within the policy; the boundary does not move when the optimizer does. The policy artifact is the unit an auditor reviews.

IP-2: Tiered Model Pool

Context. Requests vary by orders of magnitude in difficulty, and models vary by orders of magnitude in cost. *Problem.* Serving every request from the strongest model wastes budget; serving every request from the cheapest model fails the hard ones. *Forces.* Quality, cost, and latency pull against one another, and the right balance differs per request. *Solution.* Organize the pool into capability tiers and let the routing policy of IP-1 select a tier per request, escalating only when a cheaper tier's confidence is inadequate [4, 12]. *Consequences.* Cost tracks difficulty rather than worst case. The tier boundaries become a natural place to attach data-handling constraints, since a tier is also a trust class.

IP-3: Service-Level-Objective-Aware Speculative Decoding

Context. Latency is a budget the institution commits to, not a value it observes after the fact. *Problem.* Generation latency varies with request shape, and a fixed decoding strategy meets the target on some requests and misses it on others. *Forces.* Throughput and tail latency trade against

each other, and the acceptable trade differs by service-level objective. *Solution.* Treat speculative decoding as a latency primitive bound to a service-level objective rather than as an optimization applied uniformly. Adapt the speculation strategy to the target and the request shape [7, 18, 22], and carry the technique into distributed serving where the pipeline allows [15, 26]. *Consequences.* The latency target becomes a declared boundary that the serving layer optimizes toward, rather than an emergent property. The decoding strategy is an optimizer inside that boundary.

IP-4: Decision-Grade Observability

Context. An institution must reconstruct why a particular model answered a particular request, possibly long after the fact. *Problem.* Conventional service telemetry records that a call happened and how long it took, but not the decision that selected the model or the basis for it. *Forces.* Storage and privacy limit what can be retained; accountability requires that the decision be reconstructable. *Solution.* Emit a routing record for every decision that captures the inputs to the decision (request class, applicable policy version, candidate tiers), the decision itself, and its justification, while keeping payloads out of the record where policy requires. The record is evidence, not a debugging log. *Consequences.* The audit question becomes answerable from the record alone. Observability stops being instrumentation and becomes part of the governed artifact set.

IP-5: Router Lifecycle Security

Context. The router decides where sensitive requests go, which makes it a high-value target and a single point of governance. *Problem.* A router that can be changed without a gate can be made to route sensitive traffic to the wrong place, silently. *Forces.* Operational agility wants fast router changes; security and compliance want every change reviewed. *Solution.* Treat the router as a governed artifact with a promotion pipeline and a conformance gate, and treat its lifecycle as an attack surface to be defended [14]. A router version reaches production the way a regulated change does: reviewed, recorded, and reversible. *Consequences.* The promotion gate is a boundary; the router's internal improvement is an optimization behind it. A change to where sensitive traffic flows cannot happen without leaving evidence.

5 Application Patterns

The application layer composes agents into workflows that an institution can stand behind. Its patterns make the agent's internal structure explicit, the workflow inspectable, and human oversight a designed layer rather than an afterthought.

AP-1: Planner, Executor, Verifier, Generator

Context. A task requires more than one model call: a plan, one or more tool actions, a check, and a final response. *Problem.* A single undifferentiated agent that plans, acts, checks, and answers in one step cannot be reasoned about, reviewed, or improved part by part. *Solution.* Separate the agent into four roles. A planner decomposes the task; an executor performs tool actions under contract; a verifier checks the result before it is believed; and a generator produces the final response from what survived the check [2, 13]. *Consequences.* Each role can be measured, bounded, and improved independently. The verifier is the natural place to attach a boundary, since it decides what is allowed to leave the system.

AP-2: Activity-on-Vertex Workflow Graph

Context. A workflow connects several agents with dependencies and opportunities for parallelism. *Problem.* A workflow buried in imperative code cannot be inspected, reasoned about, or refined without reading the code. *Solution.* Model the workflow as an activity-on-vertex graph in which

nodes are agent invocations and edges are dependencies, and serialize it as inspectable configuration [17, 27]. *Consequences.* The workflow becomes an artifact that can be reviewed, versioned, and automatically refined. Parallelism becomes a property of the graph rather than a hidden detail of the code.

AP-3: Structure-Only Trajectory Logging

Context. Improving a planner requires the record of what past runs did. *Problem.* Logging full inputs and outputs of every step is a privacy and storage liability inside a regulated institution. *Solution.* Log the structure of each trajectory, the plan, the sequence of tool calls, the verifier's corrections, and the outcome, while excluding sensitive payloads. The structural record is enough to train a better planner [13] and to reconstruct what the workflow did. *Consequences.* The institution gains a training signal and an audit trail without retaining the content that would make either a liability. The boundary on what is logged is explicit.

AP-4: Tiered Human-in-the-Loop

Context. Some agent actions are reversible and low-stakes; others move money or make a regulated statement. *Problem.* Reviewing everything is impossible, and reviewing nothing is indefensible. *Solution.* Design human review as a runtime layer with tiers: auto-approve, lightweight review, and full supervisory review, with a default tier and a confidence threshold per node and escalation as stakes rise [11]. The tier is a property of the workflow graph, set per node. *Consequences.* Review effort tracks risk. The threshold is a boundary the optimizer may not cross: below the institution's confidence bar, a human must look, regardless of throughput pressure.

AP-5: Capability Contract

Context. A population of agents grows, and no single person can say what each agent may touch. *Problem.* Without a declared boundary per agent, capability grows faster than the institution can govern it. *Solution.* Require every agent to declare a capability contract: what it reads, what it produces, what actions it may take, and the boundary it may not cross [2]. Registration is conditional on the contract. *Consequences.* The contract is the unit of governance for the agent population. Least privilege becomes declarable and enforceable rather than aspirational.

These patterns describe the structure of the work, not the mechanism that implements it. They hold whether a workflow is built from autonomous agent loops, from deterministic pipelines with retrieval and structured execution, or from a hybrid of the two. The movement in production toward fewer autonomous loops and more deterministic structure does not retire the patterns: a deterministic step still needs a declared boundary on what it may touch, a record of what it did, and a review tier when its stakes are high.

6 Operations Patterns

The operations layer ships models, agents, and code on one platform that holds authority by degrees. Its patterns make the platform a catalog of record, bound agent authority to policy, and measure an agent before trusting it.

OP-1: Golden Path

Context. Every team in an institution makes the same delivery decisions, and the institution wants those decisions made once and reused. *Problem.* Documented best practice that is easier to leave than to follow is not governance; it is advice. *Solution.* Encode the institution's accumulated judgment as a golden path: a paved road whose safe way is also the easy way, with enforced defaults and a bounded region of choice [19–21]. *Consequences.* The path is governance only because leaving

it is harder than walking it. The enforced defaults are boundaries; the bounded choices are the team's optimization space.

OP-2: Policy-Bounded Agentic Delivery

Context. Agents begin to act inside the delivery pipeline, not only to suggest. *Problem.* An agent that can act in the pipeline without a declared bound inherits authority the institution cannot defend. *Solution.* Bound every agent action in the pipeline with a delivery guardrail that declares what the agent may decide, the bound it may not cross, and the evidence it must record [3, 10]. Authority is granted through explicit trust tiers, from suggestion to supervised action to scope-limited autonomy. *Consequences.* Autonomy becomes something the platform grants and records rather than something an agent assumes. The trust tier is the boundary; the agent's behavior inside it is the optimization.

OP-3: Reinforcement-Framed Adaptive Testing

Context. Running every test on every change is slow; skipping tests blindly is dangerous. *Problem.* A fixed test policy cannot adapt to which changes actually carry risk. *Solution.* Frame test selection as a decision process in which the pipeline learns which tests to run for a given change, subject to a hard set of tests that may never be skipped [24]. The never-skip set is declared per service as a risk profile. *Consequences.* Test effort tracks risk rather than habit. The never-skip set is a boundary; the learned selection is the optimizer inside it.

OP-4: Benchmark Before Authority

Context. An agent is proposed for a role in the pipeline. *Problem.* Granting authority on the strength of a demonstration invites failure the first time the agent meets a case the demonstration did not cover. *Solution.* Measure the agent against a benchmark of real failures for its task before granting it authority, and re-measure as it changes [6, 16]. Authority is earned against evidence, not asserted. *Consequences.* The benchmark result is the evidence that justifies a trust tier. The discipline turns the trust-tier boundary of OP-2 into something an institution can defend.

OP-5: Perception, Action, Reasoning, Adaptation

Context. Operational agents must do more than alert; they must observe, act, reason about the result, and adjust. *Problem.* An operating model that names only perception and action omits the reasoning that justifies an action and the adaptation that closes the loop. *Solution.* Structure operational agents around four faculties: perception of the system's state, action within bounds, reasoning that justifies the action, and adaptation that updates behavior from the result. This four-part operating model (PARA) extends the perception-action loops common in the literature [23, 25] with explicit reasoning and adaptation, so that an agent's action carries a justification and the loop improves over time. *Consequences.* The reasoning faculty is where the agent's justification is recorded, which makes the operating model compatible with the evidence requirement. Adaptation is bounded by the same guardrails as action.

OP-6: Per-Team Cost Attribution

Context. Model and compute spend is real, shared, and easy to lose track of. *Problem.* Cost that is not attributed to the team that incurred it cannot be governed and will grow without an owner. *Solution.* Attribute spend to the team that caused it, using the same routing and trajectory records the other patterns produce. Cost becomes a first-class signal in the platform, not a monthly surprise. *Consequences.* The institution gains a control surface for spend that reuses existing evidence. Cost attribution closes the loop between the infrastructure layer's routing records and the operations layer's accountability.

OP-7: Budget as Boundary

Context. Capability and autonomy both cost money, and the marginal value of an additional call, agent step, or retry falls while its marginal cost and risk do not. *Problem.* An optimizer told only to raise quality will spend without limit, because nothing in its objective stops it. Cost attribution (OP-6) measures spend but does not bound it. *Forces.* Quality improves with spend; budgets are finite; an unbounded autonomous system can consume a quarter's compute on a single hard request. *Solution.* Treat a spend cap as a hard boundary rather than a soft target. The platform optimizes a utility that trades quality against cost, risk, and delay, and the budget is the clause that utility may not cross. When a request, an agent run, or a team reaches its cap, the system degrades deliberately, to a cheaper tier, a halt, or an escalation, instead of spending past the limit. The cap is set and changed by governed policy, with the attribution records of OP-6 as its evidence. *Consequences.* Spend becomes governable in the same shape as every other boundary. The economic objective is free to optimize inside the cap, and the cap does not move when the objective pushes against it. The pattern closes the economic loop that cost attribution opens.

7 Cross-Layer Composition: An Illustrative Incident

The patterns are stated layer by layer, but their value is in how they compose. We trace one incident through all three layers. The incident is fictional and reports no measurement. Its purpose is to show the path a single decision takes, and the evidence it leaves, not to demonstrate a result. The path is shown in Figure 2.

A transaction at Nebula Financial is flagged as potentially fraudulent. The infrastructure layer routes the analysis request. The governed routing policy (IP-1) recognizes the request as carrying regulated customer data and constrains it to a permitted tier (IP-2) inside the required jurisdiction. A routing record (IP-4) is written, naming the policy version and the basis for the route. None of this is an optimization choice; the residency constraint is a boundary, and the router optimizes only within it.

The task is more than a single call, so it enters the application layer as a workflow. A planner decomposes the investigation; an executor gathers the transaction history and related signals under a capability contract (AP-5) that forbids any state-changing action; a verifier checks the assembled case against the institution's disclosure rules; and a generator drafts the finding (AP-1). The workflow is an activity-on-vertex graph (AP-2), and a structure-only trajectory record (AP-3) captures the plan, the tool calls, and the verifier's corrections, without retaining the customer payloads. Because the finding may lead to a regulated action, the relevant node carries a full-supervisory-review tier (AP-4): a human must sign before anything leaves the system. The confidence threshold is a boundary; throughput pressure does not lower it.

The investigation reveals a gap in the institution's own controls, and the fix is a change to the platform itself. It enters the operations layer through the governed pipeline. The change follows a golden path (OP-1). An agent proposes the remediation inside the pipeline, bounded by a delivery guardrail (OP-2) that records what it decided and forbids it from acting beyond a suggestion until a human raises its trust tier. The pipeline runs the never-skip test set and the learned selection around it (OP-3), and the agent that proposed the fix had been measured against real failures before it was given even suggestion authority (OP-4). A delivery record is written. The loop closes: the institution can now reconstruct, from three records written at three layers, who routed what, which agents investigated it under which contracts and review, and how the fix reached production and under whose authority.

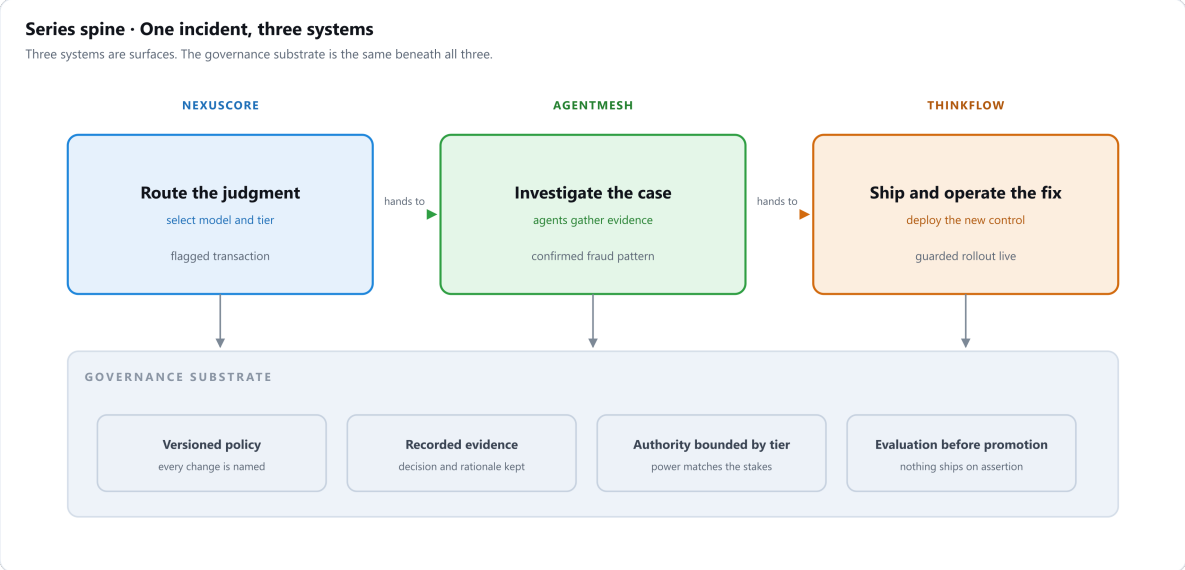

Figure 2: The illustrative incident traced across the three layers. A routing decision, an agent investigation under tiered human review, and a governed delivery each write a record; the three records compose into one reconstructable account. The scenario is fictional and reports no measurement. Figure adapted from the companion book series.

Table 1: Pattern selection by environment. The required set grows with the strength of the accountability requirement.

Environment	Required	Optional
Startup, early stage	IP-1, OP-1	tiered review (AP-4) and benchmarking (OP-4) as stakes grow
Enterprise, unregulated	IP-1 to IP-5, capability contract (AP-5)	remaining application and operations patterns
Regulated institution	all seventeen	none; the boundaries are obligations, not choices

8 Applying the Pattern Language

Not every setting needs every pattern. The language is a menu ordered by the strength of the accountability requirement, and two views help a team use it: which patterns an environment requires, and the order in which a platform tends to adopt them.

8.1 Selection by environment

Table 1 sorts the patterns by how much an institution must be able to reconstruct and defend a decision. A team optimizing for speed adopts the patterns that pay for themselves at once and defers the rest. A regulated institution adopts all of them, because each boundary it omits is a question it cannot later answer.

8.2 Adoption stages

A platform rarely adopts the language all at once. It tends to pass through stages, each adding a layer of governance to the one before.

- *Stage 0.* A single model behind an application: no routing, no record.
- *Stage 1.* A routing layer (IP-1 to IP-5) that turns a model call into a governed event.
- *Stage 2.* An agent layer (AP-1 to AP-5) that composes calls into inspectable, reviewed workflows.
- *Stage 3.* A governance layer that makes records, contracts, and review tiers first-class across both layers.
- *Stage 4.* An autonomous platform (OP-1 to OP-7) that raises agent authority by degrees, each increase earned against evidence and bounded by budget.

The order matters. A team that reaches Stage 4 without passing through Stage 3 has built autonomy it cannot account for. The stages are the safe sequence in which to grant a platform power.

9 Failure Modes

A pattern fails not when it is absent but when its boundary is set wrong. Each failure below is a boundary that is present yet mis-set: drifting, ratcheting, missing, proliferating, or local where it should be global. Naming the failures is part of the language, because a practitioner recognizes a failure faster than a prescription.

Router drift. A governed routing policy (IP-1) updated often but reviewed loosely accumulates small changes until sensitive traffic flows somewhere no one intended. The boundary exists but has stopped being enforced. The defense is the promotion gate of IP-5 applied to the policy itself.

Trust-tier inflation. Authority granted under OP-2 is raised after each success and never lowered after a failure, until agents act with a latitude no evidence justifies. The boundary ratchets in one direction only. The defense is to bind the tier to a current benchmark (OP-4), so authority can fall as well as rise.

Audit overload. Decision-grade observability (IP-4) and trajectory logging (AP-3) record so much that nothing can be found when it is needed, and the evidence becomes noise. The boundary on what is recorded is missing. The defense is to record the decision and its basis, not the world.

Boundary explosion. Every incident adds a new boundary until the optimizer has no room left and the platform calcifies. The governance meant to enable autonomy now prevents it. The defense is to treat a boundary as a cost and to retire boundaries through the same promotion that adds them.

Agent collusion. A population of agents (AP-5), each optimizing a shared objective, finds a path that satisfies every capability contract individually while violating the institution's intent collectively. No single boundary is crossed; the gap is between them. The defense is a verifier (AP-1) at the workflow's output, not only at each step, and a human-review tier (AP-4) on the composite action.

These failures share one shape, the boundary and the optimizer confused for one another, which is why the invariant predicts them and why stating it explicitly is a practical safeguard rather than a slogan.

Table 2: Evaluation agenda. Each row is a falsifiable hypothesis for future empirical work, not a result. No quantity in this table has been measured.

Pattern	Quantity to measure	Predicted direction
IP-1 Governed Routing	cost per request at fixed quality	below a single-model baseline
IP-3 SLO-Aware Decoding	p99 latency against the SLO	closer to target, fewer breaches
AP-3 Structure-Only Logging	planner accuracy from structural logs	comparable to full-payload logging
AP-4 Tiered HITL	review time at a fixed rate of risk caught	lower than uniform review
OP-3 Adaptive Testing	test time per change against escaped defects	less time at equal or fewer escapes
OP-4 Benchmark Before Authority	incidents after an authority increase	fewer than ungated authority
OP-7 Budget as Boundary	per-team spend variance	bounded, with deliberate degradation at the cap

10 Discussion and Evaluation Agenda

What the patterns assume. The language assumes a regulated, or self-regulating, institution: one that values the ability to reconstruct and defend a decision above raw throughput. In settings where that value does not hold, several patterns lose their justification, because the boundaries they protect are not required. The patterns also assume a platform team with the authority to set boundaries centrally. In a federated organization without that authority, the patterns describe a target state rather than an immediately reachable one.

Relationship to existing practice. None of the individual mechanisms is novel in isolation. Learned routing, speculative decoding, modular agents, inspectable workflows, tiered human review, golden paths, and policy-bounded pipelines all appear in the cited literature. The contribution is the composition: the claim that these mechanisms share one internal shape, the separation of a boundary from an optimizer, and that stating that shape explicitly is what lets a platform raise autonomy without losing accountability. The pattern language is the vehicle for that claim.

An evaluation agenda. The patterns are stated without measurement, but each makes a falsifiable prediction. Table 2 records, for a representative subset, the quantity a controlled study would measure and the direction the pattern predicts that quantity should move. The table is not a result and claims no number; every entry is a hypothesis for future empirical work. Stating the predictions explicitly is itself part of the contribution, because a pattern that predicts nothing measurable cannot be refuted and therefore teaches little.

Threats to validity. This is a conceptual paper, and its limits follow from that. First, there is no empirical evaluation. The patterns are justified by the forces they resolve and by their grounding in the literature, not by a controlled study, and their benefit in a real deployment is therefore argued rather than measured. Second, the unifying scenario is fictional. It was constructed to exhibit the patterns, which makes it a clear illustration and a weak test: a scenario built to fit a language cannot also refute it. Third, the catalog is drawn from the present state of a fast-moving field, and specific mechanisms, particularly in serving and in agentic delivery, will change; the patterns are

stated at the level of the force they resolve to outlast the mechanism, but that abstraction is a claim, not a guarantee. Fourth, generalization beyond regulated financial settings is asserted, not shown. We state these limits plainly so that the paper is read as what it is: an architecture and a language to be tested, not a result.

Future work. The natural next step is empirical. Each pattern suggests a measurable question: whether decision-grade observability reduces the time to reconstruct a decision, whether structure-only logging preserves enough signal to train a planner, whether benchmark-before-authority reduces incidents after an agent is granted a trust tier. A second direction is to test the invariant itself, by looking for production mechanisms that resist the boundary-versus-optimization separation and asking what they have in common.

11 Conclusion

Production is not deployment. In a regulated institution, a model call is a governed decision that happens to produce text, an agent action is an audit record that happens to do work, and a deployment is a change that must be defended. This paper has presented a three-layer reference architecture and a pattern language of seventeen patterns for building such systems, unified by a single invariant: a boundary is a clause the optimizer may not cross, and everything else is optimization. The value of the language is not in any one pattern, most of which restate established practice, but in the composition, and in the claim that a platform can raise its autonomy as fast as it can prove its judgment, provided every decision is bounded by policy and recorded as evidence. The architecture is offered as a structure to be tested, not as a result already proven, and the explicit threats to validity above are part of that offer.

Data and reproducibility. This paper reports no experiment and therefore no data. The illustrative scenario is fictional. The conceptual artifacts referenced in the patterns are developed in full in the companion book series.

Acknowledgments. Placeholder. Acknowledgments will name real contributors only, and are omitted until they can be stated accurately.

References

[1] Workstream: A local-first developer command center for the AI-augmented engineering workflow. arXiv:2604.17055 [cs], 2026.

[2] Daman Arora, Atharv Sonwane, Nalin Wadhwa, Abhav Mehrotra, Saiteja Utpala, Ramakrishna Bairi, Aditya Kanade, and Nagarajan Natarajan. MASAI: Modular architecture for software-engineering AI agents. *arXiv preprint arXiv:2406.11638*, 2024.

[3] Mohammad Baqar, Saba Naqvi, and Rajat Khanda. AI-augmented CI/CD pipelines: From code commit to production with autonomous decisions. *arXiv preprint arXiv:2508.11867*, 2025.

[4] Yu-Neng Chuang, Leisheng Yu, Guanchu Wang, Lizhe Zhang, Zirui Liu, Xuanting Cai, Yang Sui, Vladimir Braverman, and Xia Hu. Confident or seek stronger: Uncertainty-based on-device LLM routing. *arXiv preprint arXiv:2502.04428*, 2025.

[5] Jingzhi Fang, Yanyan Shen, Yue Wang, and Lei Chen. Improving the end-to-end efficiency of offline inference for multi-LLM applications based on sampling and simulation. *arXiv preprint arXiv:2503.16893*, 2025.

[6] Taher A. Ghaleb. When AI agents touch CI/CD configurations: Frequency and success. *arXiv preprint arXiv:2601.17413*, 2026.

[7] Kaiyu Huang, Hao Wu, Zhubo Shi, Han Zou, Minchen Yu, and Qingjiang Shi. AdaSpec: Adaptive speculative decoding for fast, SLO-aware large language model serving. *arXiv preprint arXiv:2503.05096*, 2025.

[8] Kunal Jain, Anjaly Parayil, Ankur Mallick, Esha Choukse, Xiaoting Qin, Jue Zhang, Íñigo Goiri, Rujia Wang, Chetan Bansal, Victor Rühle, Anoop Kulkarni, Steve Kofsky, and Saravan Rajmohan. Intelligent router for LLM workloads: Workload-aware load balancing. *arXiv preprint arXiv:2408.13510*, 2024.

[9] Wittawat Jitkrittum, Harikrishna Narasimhan, Ankit Singh Rawat, Jeevesh Juneja, Congchao Wang, Zifeng Wang, Alec Go, Chen-Yu Lee, Pradeep Shenoy, Rina Panigrahy, Aditya Krishna Menon, and Sanjiv Kumar. Universal model routing for efficient LLM inference. *arXiv preprint arXiv:2502.08773*, 2025.

[10] Satyadhar Joshi. A review of generative AI and DevOps pipelines: CI/CD, agentic automation, MLOps integration, and large language models. *SSRN Electronic Journal*, 2025. SSRN:5290005.

[11] Sandeep Reddy Kaidhapuram. Human-in-the-loop (HITL) orchestration for agentic use-cases: A practical framework for supervising autonomous AI agents in production environments. *International Journal of Computer Techniques*, 12(6), 2025.

[12] Baolin Li, Yankai Jiang, Vijay Gadepally, and Devesh Tiwari. LLM inference serving: Survey of recent advances and opportunities. *arXiv preprint arXiv:2407.12391*, 2024.

[13] Zhuofeng Li, Haoxiang Zhang, Seungju Han, Sheng Liu, Jianwen Xie, Yu Zhang, Yejin Choi, James Zou, and Pan Lu. In-the-flow agentic system optimization for effective planning and tool use. *arXiv preprint arXiv:2510.05592*, 2025.

[14] Qiqi Lin, Xiaoyang Ji, Shengfang Zhai, Qingni Shen, Zhi Zhang, Yuejian Fang, and Yansong Gao. Life-cycle routing vulnerabilities of LLM router. *arXiv preprint arXiv:2503.08704*, 2025.

[15] Xing Liu, Lizhuo Luo, Ming Tang, Chao Huang, and Xu Chen. FlowSpec: Continuous pipelined speculative decoding for distributed LLM inference. *arXiv preprint arXiv:2507.02620*, 2025.

[16] Raian Latif Nabil, Hao-Nan Zhu, and Cindy Rubio-González. CI-bench: A framework for evaluating large language model tools on CI failures. In *Proc. IEEE/ACM 48th International Conference on Software Engineering (ICSE), Demonstrations*, 2026.

[17] Boye Niu, Yiliao Song, Kai Lian, Yifan Shen, Yu Yao, Kun Zhang, and Tongliang Liu. Flow: Modularized agentic workflow automation. In *Proc. International Conference on Learning Representations (ICLR)*, 2025. arXiv:2501.07834.

[18] Gabriele Oliaro, Zhihao Jia, Daniel Campos, and Aurick Qiao. SuffixDecoding: Extreme speculative decoding for emerging AI applications. *arXiv preprint arXiv:2411.04975*, 2024.

[19] OpsLevel. The 2025 ultimate guide to building a high-performance developer portal, 2025. https://www.opslevel.com/resources/2025-ultimate-guide-to-building-a-high-performance-developer-portal.

[20] Harshad Pitkar. Platform engineering and developer experience: A systematic review of concepts, benefits and future directions. *World Journal of Advanced Engineering Technology and Sciences*, 18(2):241–248, 2026. doi: 10.30574/wjaets.2026.18.2.0112.

[21] Red Hat. Why developer portals matter more in the age of AI agents. Red Hat Blog, 2025. https://www.redhat.com/en/blog/why-developer-portals-matter-more-age-ai-agents.

[22] Ranajoy Sadhukhan, Jian Chen, Zhuoming Chen, Vashisth Tiwari, Ruihang Lai, Jinyuan Shi, Ian En-Hsu Yen, Avner May, Tianqi Chen, and Beidi Chen. MagicDec: Breaking the latency-throughput tradeoff for long context generation with speculative decoding. *arXiv preprint arXiv:2408.11049*, 2024.

[23] Jimmy Song. The second half of cloud native: The era of AI-native platform engineering has arrived, 2025. https://jimmysong.io/blog/cloud-native-second-half-ai-native-platform-engineering/.

[24] Aniket Abhishek Soni, Milan Parikh, Rashi Nimesh Kumar Dhenia, Jubin Abhishek Soni, Ayush Raj Jha, and Sneja Mitinbhai Shah. Reinforcement learning for dynamic workflow optimization in CI/CD pipelines. *arXiv preprint arXiv:2601.11647*, 2026.

[25] xDevOps. DevOps and SRE AI platforms: 2025 and 2026 atlas. GitHub Pages, 2025. https://xdevops-ai.github.io/devops-sre-ai-atlas-2025/.

[26] Heming Xia, Zhe Yang, Qingxiu Dong, Peiyi Wang, Yongqi Li, Tao Ge, Tianyu Liu, Wenjie Li, and Zhifang Sui. Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 7655–7671, 2024.

[27] Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating agentic workflow generation. In *Proc. International Conference on Learning Representations (ICLR)*, 2025. arXiv:2410.10762.