



Decision Models: Build an Enterprise Decision Layer Using Decision Models and Large Language Models

A practical guide to typed decision models, calibrated confidence,
and the control plane for agentic AI

October 2, 2026

White Paper

Table of Contents

Introduction	2
The gap between agents and actions	2
What a decision model actually is	4
The 2026 decision model wave	5
Why the decision layer should not be an LLM	7
Placing the decision layer in your architecture	9
Calibrated confidence and escalation routing	10
Governance at sub-millisecond speed	11
Owning the decision layer: build, buy, or self-host	13
Operating a decision layer	14
Evaluation checklist	15

Most enterprise AI programs have quietly acquired a structural problem. The agents are getting better at acting, and the systems around them are not getting better at deciding. A support agent that can draft a reply, query a billing system, issue a refund, and send mail is a genuinely useful thing. The same agent is also a sequence of production database writes wrapped in a language model, and every one of those writes is a decision that something has to make.

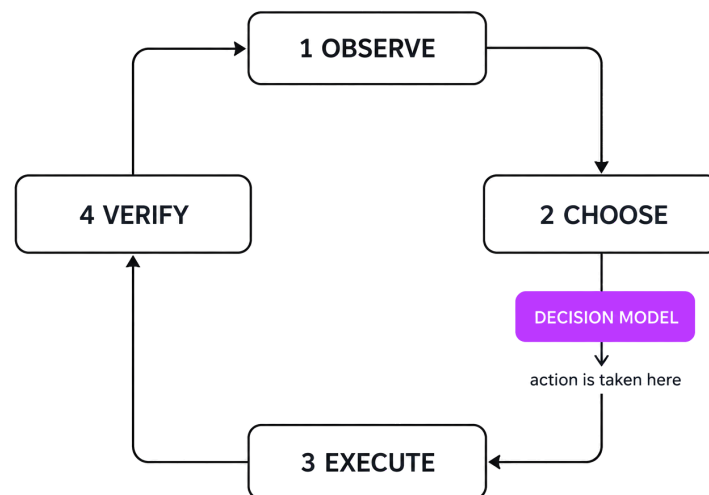
For the last two years the default answer has been to route those decisions back through a large language model. Classification, routing, scoring, verification, policy checks: ask the model, parse the text, act on the answer. That works in a prototype and breaks down in production, for reasons that are more about economics and failure modes than about intelligence.

A different approach has been gathering momentum through 2026. It goes by several names, but the most useful one is a **decision model**: a model that does not generate text at all, and instead answers a set of typed questions about a given state in a single pass. This whitepaper is an attempt to explain what decision models are, where they belong in an enterprise architecture, what the evidence for and against them actually says, and how to evaluate one without believing a benchmark table.

The gap between agents and actions

An autonomous agent is a loop. It observes a state, chooses an action, executes it, and observes the result. The interesting engineering problem is not the choosing. It is the boundary between choosing and executing.

That boundary is where enterprises get nervous, and the nervousness is rational. The action may write to production. It may expose customer data. It may move money. It may be irreversible. Meanwhile the volume of these interactions is rising faster than the ability of humans to review them, which means a line that was policed socially, by everyone knowing that a human pressed the button, quietly stops being policed at all.



The boundary between choosing an action and executing it is where the control problem lives.

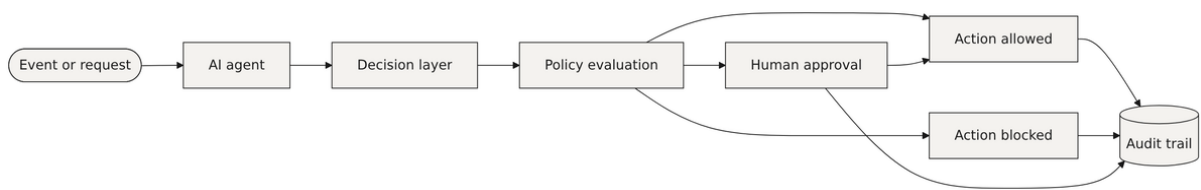
Three things tend to happen next, in this order. First, a team adds a prompt-based guardrail that asks a model whether an action is acceptable. Second, that guardrail becomes the highest-traffic dependency in the critical path. Third, latency and cost force the team to either weaken the guardrail or move it out of the path, which is the same as weakening it.

The pattern is recognizable enough that it is worth naming the kind of work that keeps landing in that boundary:

- Classification, where a request has to be sorted into one of a known set of categories.
- Routing, where a request has to be sent to one of several handlers, queues, or models.
- Scoring, where a state has to be rated against a rubric, such as a risk tier or a confidence band.
- Ranking, where a set of candidates has to be ordered.
- Verification, where an existing claim or output has to be checked against a rule.
- Policy evaluation, where an action has to be allowed, blocked, or escalated to a human.

Every one of these tasks shares a property that chat completion is bad at accommodating. The set of acceptable answers is **known in advance**, and it is small. A router does not need prose, it needs one of seven labels. A policy check does not need an explanation, it needs one of three verdicts. You are paying a generative model to produce a stream of tokens, and then you are paying again, in latency and in parsing failures, to collapse that stream back down to the single value you already knew you wanted.

If you know the shape of the answer before you ask the question, generation is the wrong primitive.



Decision layer placement between an agent and the actions it attempts

What a decision model actually is

A decision model keeps the language understanding of a large model and discards the generation. It reads the state, reads a set of typed questions, and emits typed answers in a single forward pass. No autoregressive loop, no token stream to parse, no possibility of a malformed response.

The clearest public description of the mechanism comes from a 2026 paper on typed decision models, which makes the point about output constraints precisely: the set of possible outputs is the set of declared options, so a malformed answer is not unlikely, it is unrepresentable, [as the authors of this-that-model-1.0 put it](#). This is the property that separates a decision model from a prompted language model with a JSON schema, and it matters more than any benchmark number. A schema is a request. Constrained decoding is a fairly strong request. A closed output head is a guarantee.

TypeSafe, the company driving most of the current attention, [exposes three primitives in its documentation](#):

- **Choice.** Given a list of up to 255 options, return one selection plus a probability distribution over all of them.
- **Score.** Given a rubric, return a value in a fixed range, such as 1 to 5, with a distribution.
- **Noul.** Given a yes or no question, return a single probability between 0 and 1.

Every answer carries a confidence value. This is the part that experienced practitioners should focus on, because it is what makes the architecture composable rather than merely fast. A model that returns a label is a classifier. A model that returns a label together with a calibrated probability of being right is a component you can build a control plane out of, because the system can now reason about its own uncertainty.

A decision model gives the same kind of output, except the classes are defined at inference time, in the prompt, instead of being fixed when the model was trained.

[A widely circulated practitioner explainer makes the comparison directly](#), and it is the most useful mental model available. Anyone who has deployed a classifier head on top of a transformer already understands the mechanics. The novelty is that the class list becomes a runtime input, which turns a trained artifact into a configurable one.

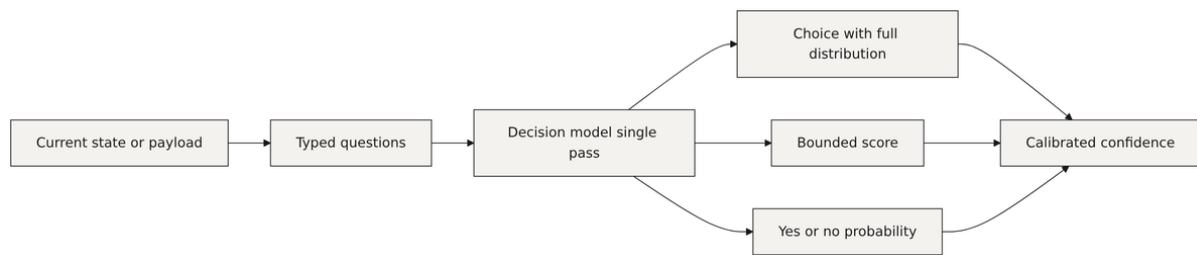
System One and System Two, carefully

The marketing around decision models leans hard on the System One and System Two distinction from behavioural psychology. The academic literature treats that distinction as a useful analogy for human cognition, not as a claim about parallel versus sequential computation in neural networks [a distinction Nature Reviews Psychology drew in November 2025](#). The mapping of System One onto single-pass inference is the vendors' repurposing of the term, and it is worth being precise about that. The engineering distinction is real and consequential. The psychology is a metaphor borrowed for branding.

Dimension	Generative model (System Two framing)	Decision model (System One framing)
Output	Token stream, parsed after the fact	Typed answer, closed set
Failure mode	Malformed output, wrong format, hallucinated field	Wrong label, with a stated confidence
Cost shape	Per output token, scales with verbosity	Per input token, flat regardless of questions asked
Latency shape	Grows with reasoning length	Roughly constant per request
Uncertainty	Not exposed by default	First-class return value
Fits	Open-ended synthesis, planning, writing	Classification, routing, scoring, policy

Table: The two primitives and where each belongs.

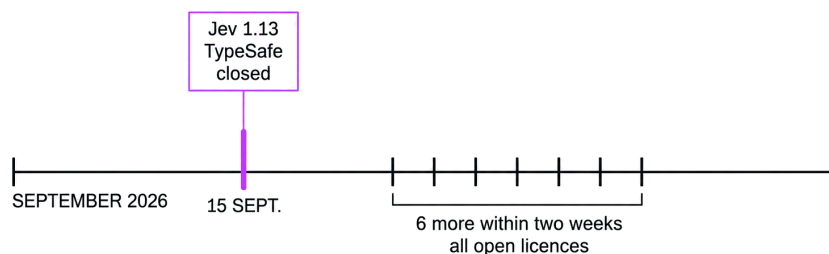
The practical consequence of the table is not that one row is better. It is that an enterprise stack in 2026 needs both, and that most stacks currently use the expensive row for work that belongs in the cheap one.



The typed request and response contract of a decision model

The 2026 decision model wave

Decision models stopped being a niche idea in September 2026. TypeSafe left stealth on 15 September with Jev, its first model, and a \$40M seed round led by DCVC, with founders Diogo Almeida, formerly of OpenAI, Erik Gafni, and Sasha Sheng [announced alongside founders Diogo Almeida, formerly of OpenAI, Erik Gafni, and Sasha Sheng](#). Within two weeks the category had at least six serious entrants.



The category arrived quickly: several independent implementations became visible in the same window.

Model	Maker	Licence	Notes
Jev 1.13	TypeSafe	Closed	Hosted API only. No weights, no parameter count, no self-hosting option.
Strands Decider 2B	AWS	Apache 2.0	Built on a Qwen3.5-2B base with the language-model head removed and replaced by a small pointer head. Runs locally.
Clef and Clef-flash	Cloudflare	Apache 2.0	27B and 9B variants, multimodal, served on Workers AI and published on Hugging Face.
Laya	Convai Innovations	Apache 2.0	Encoder-based, 421M parameters, 100 or more languages.
Kev	Jared Palmer	Open	A family from 0.5B to 27B. Implements TypeSafe's request contract.
AnyJev	Nokia Applied Research	Open	A method, not a model. Adds a decision layer to an existing open LLM.
Jebadiah	Community	Apache 2.0	4B and 9B variants tuned on Apache 2.0 bases.

Table: The decision model landscape as of early October 2026.

The presence of AWS and Cloudflare in that table is the strongest available signal that this is a structural shift rather than one company's positioning. Both shipped open models that implement the same conceptual interface, and AWS adopted TypeSafe's own vocabulary for the answer types. When a hyperscaler copies a startup's interface naming, the interface has usually won.

There is also a live example of the category boundary being drawn too loosely. Google DeepMind's DiffusionGemma, a generative text model that produces 256-token blocks in parallel, appears in some vendor comparison tables alongside genuine decision models [on the Google DeepMind model page](#). It was published in June 2026, before Jev existed. Parallel generation is not the same as typed decision output, and treating the two as the same category is a mistake worth avoiding.

What the numbers do and do not mean

TypeSafe's headline claims are that Jev is dramatically faster and cheaper than a frontier model on decision-shaped tasks. Those claims are vendor-reported, and the vendor's own numbers are not mutually consistent. The launch post describes a range of 40 to 200 times faster, applied to speed. The company homepage states 193.6 times faster and 444.6 times cheaper. The two surfaces describe different quantities.

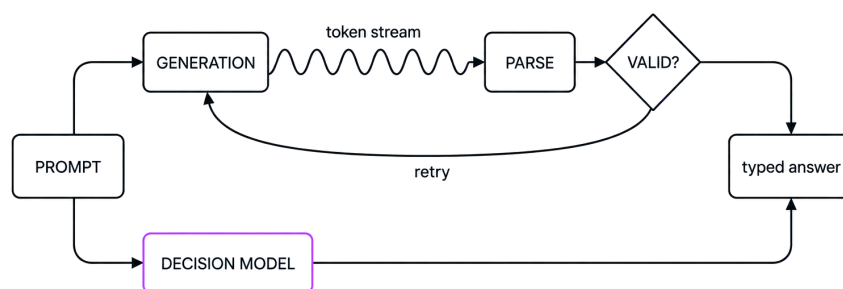
Independent parties have not reproduced the figures. One comparison site states the position plainly: the figures come from TypeSafe, and [decisioneval.dev states plainly that it has not reproduced them](#). That is not an accusation. It is the normal status of a performance claim about a closed, hosted model released three weeks earlier.

Even where independent measurement exists, the results are more careful than the marketing. A reproducible harness that compared Jev against two frontier models on the same task found a per-task cost of roughly eight ten-thousandths of a cent against slightly over one cent, and accuracy of 67.8 percent against 67.9 and 67.8 percent for the two generative baselines. The cost and latency advantage is enormous and appears real. The accuracy advantage is approximately zero, which is a much more interesting and more useful finding than a large one.

The defensible claim for a decision model is not that it is more accurate. It is that it is accurate enough, and that it costs and takes so much less that you can afford to run it on every request.

Why the decision layer should not be an LLM

The temptation to use a large language model as the decision layer comes from a reasonable place. It is the model you already have, it is general, and it requires no additional infrastructure. Three problems appear in production.



Generation produces a stream to be parsed; a decision model returns a typed answer directly.

Latency

Inline guardrails have a hard latency budget, and a large model spends all of it. Published guidance on guardrail placement suggests allocating 20 to 50 milliseconds for evaluation within a response budget under 500 milliseconds, and notes that language-model-based intent classification can cost 200 to 500 milliseconds against sub-millisecond regex. Language-model-as-judge patterns are routinely described as costing hundreds of milliseconds to seconds. Those numbers are fine for a nightly batch job and fatal for an interception point in an interactive agent loop.

Cost

Every request through the decision layer pays the model's price. If the decision layer sits in front of every agent action, the decision layer's traffic is a multiple of the agent's own traffic. One practitioner analysis of in-band harm reduction measured the token overhead at between 50 and 200 percent, describing it as not a rounding error. A layer that doubles your token spend to make decisions is a layer that will be removed.

Structural reliability

The failure mode that is easiest to underestimate is malformed output. A model that is asked to return one of three verdicts will almost always do so, and will occasionally return a sentence instead. In a control plane, an occasional unparseable response is not a small quality dent. It is an availability incident, because the code that consumes the verdict has no branch for the sentence.

A 2026 study of schema compliance in deployed systems put numbers on the gap: in an unconstrained setting, roughly 18 percent of small-model generations contained at least one structural violation, against roughly 6 percent for a larger model, a threefold difference, [reported at KDD 2026 in Schema Guardrails in the Wild](#). Related work finds that structural and syntax failures account for a substantial share of incorrect outputs, often exceeding logic or semantic errors. The correction cost lands entirely on the calling application.

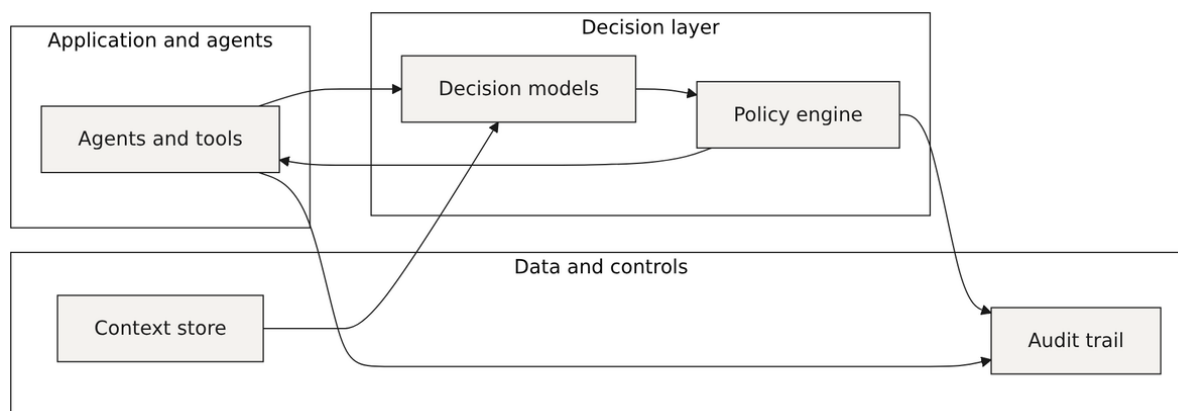
Failure mode	Generative decision layer	Decision model decision layer
Unparseable response	Possible, needs a fallback branch	Not representable
Latency per decision	200 to 500 ms typical	Tens of milliseconds
Cost scaling	Per decision, at frontier prices	Per decision, at small-model prices
Uncertainty available	Only if explicitly requested	Always returned
Behaviour on novel input	Confidently generative	Low confidence, if calibrated

Table: Why the control plane and the reasoning plane want different primitives.

The honest conclusion is not that language models are bad at decisions. They are frequently good at decisions. It is that the correctness of a control plane should not depend on a component whose output format is a probability rather than a guarantee.

Placing the decision layer in your architecture

The most useful way to think about a decision layer is as a thin, synchronous tier between intent and effect. It is not an orchestrator and it is not a policy author. It answers questions, fast, on the critical path.



Reference architecture for an enterprise decision layer

A workable decomposition has four parts:

- **The agent tier**, which holds the reasoning, planning, and tool use. This is where the large models live, and where they belong.
- **The decision tier**, which holds the decision models. It reads a state and a set of typed questions and returns typed answers with confidences.
- **The policy tier**, which holds the rules. It turns an answer into an enforcement outcome, and it is configuration rather than inference.
- **The record tier**, which holds the audit trail. It is append-only and it is the artifact your auditor will actually ask for.

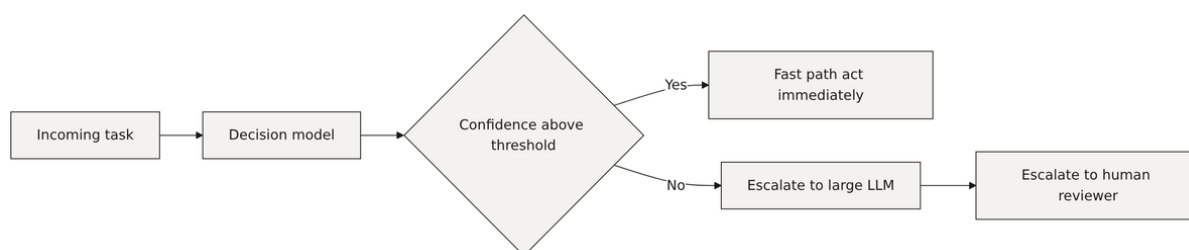
The crucial design decision is that these tiers have different scaling and reliability characteristics. The agent tier is expensive, bursty, and tolerant of a second of latency. The decision tier is cheap, high-volume, and intolerant of more than a few tens of milliseconds. The record tier is write-heavy and must never block the request path. Collapsing them into one service is how teams end up with a governance mechanism that they cannot afford to run on every request.

A second design decision concerns where questions come from. Because the questions are supplied at request time rather than baked into the model, the decision tier can serve many teams from one deployment. A platform team can operate the inference and let domain teams author the question sets, in the same way that a database team operates storage and lets application teams author schemas. That is a materially different operating model from shipping a bespoke classifier per use case.

Calibrated confidence and escalation routing

The confidence value is what turns a fast decision model into an architecture rather than a shortcut. If a model tells you it is 0.98 confident, and that number means something, you can act immediately on confident answers and escalate the rest to something slower and more expensive. This is cascade routing, and it is the reason decision models are interesting even to teams that already have strong classifiers.

Calibration itself is a well studied problem. The canonical reference is Guo et al. on calibration of modern neural networks [Guo et al. published the canonical treatment of this in 2017](#), which is worth flagging as an older source: the finding that deep networks are systematically overconfident has been known for nearly a decade, and it applies to decision models as much as to the classifiers it originally described.



Confidence based escalation routing between a decision model and a larger model

The evidence on whether cascading works is genuinely contested, and anyone evaluating a decision layer should know both sides.

The case for. A September 2026 study from Carnegie Mellon reports that accepting a decision model's confident verdicts and escalating the remainder to a frontier model produced a result 0.9 points more accurate than the frontier model alone on 1,610 held-out pairs, at 41 percent of that model's fee, [according to the JEV-as-a-Judge study](#). The same work reports that where a verdict can be read directly off the text, the decision model lands within three points of the frontier model at 0.36 percent of the fee, and that it falls behind where the verdict has to be derived, as in mathematics, code, or logic.

The case against. A second 2026 paper reaches close to the opposite conclusion. [On rubric judging, deferring a decision model's uncertain verdicts to a language model gains almost nothing](#), because the language model repeated 96.0 percent of the decision model's most confident errors, against an independence baseline of 50.3 percent. The mechanism is the important part: cascading only helps when the two models make **independent** errors. When they are wrong in the same places, the second opinion is not a second opinion.

Finding	Direction	Source
Confident-accept plus escalate beat frontier alone by 0.9 points at 41 percent of the fee	Supports cascading	CMU, arXiv 2609.26550
Deferral recovers at most 1.5 to 2.0 points where errors correlate	Against cascading	arXiv 2609.29769
Confidence-aware routing cut inference cost 31 percent while reducing calibration error	Supports cascading	arXiv 2605.18796
Confidence-based deferral can be suboptimal in theory and in practice	Against cascading	NeurIPS 2023

Table: The cascade routing debate, both sides.

The practical guidance that follows is not to pick a side. It is to **measure the error correlation before you build a cascade**. If your decision model and your escalation target make errors on the same inputs, escalation buys you almost nothing and you should spend the effort on a better first-stage question set instead. There is also direct evidence that confident answers are not automatically correct ones: one security analysis of a decision model found its confidence saturated at 1.0 on a majority of items, with several of those answers wrong. High stated confidence is a claim about the model's training distribution, not a fact about the world.

Governance at sub-millisecond speed

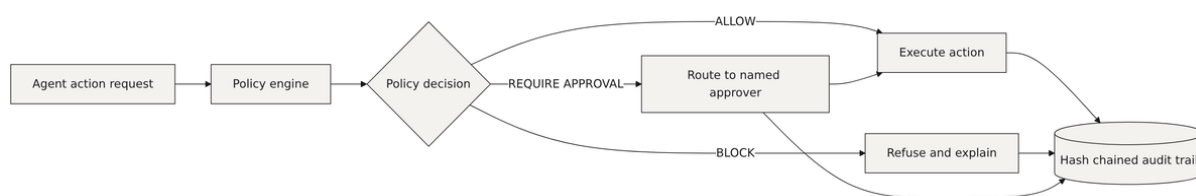
Governance is the use case where the latency argument stops being about cost and starts being about whether the control can exist at all.

An AI governance layer does a small number of things. It evaluates a proposed action against policy. It produces an enforcement decision, typically allow, block, or require human approval. It records what happened, immutably, so that the decision can be reconstructed later. It may pause an action in a queue until a named human approves it.

Every one of those operations is a decision. And because the governance layer sits in front of every agent action by definition, its cost and latency are multiplied by the entire agent fleet's traffic. This is the specific reason a governance layer that calls a frontier model for each policy evaluation is self-defeating. It becomes the most expensive and slowest component in the stack, and the first thing teams carve out under pressure.

The regulatory backdrop strengthens the argument without settling it. Article 14 of the EU AI Act requires that high-risk systems can be effectively overseen by natural persons, with oversight commensurate to the risks, level of autonomy, and context of use [as set out in Article 14 of the EU AI Act](#). Human oversight that depends on a 400 millisecond model call per action does not scale to the volume it was written to cover. Note also that the high-risk obligations were rescheduled: Regulation (EU) 2026/1744 moved the stand-alone Annex III deadline to 2 December 2027, while the Article 50 transparency duties applied from 2 August 2026. Teams working to older dates should re-baseline.

Policy-as-code is the emerging implementation pattern, and it is worth being precise about its maturity. Toolkits such as Microsoft's agent governance policy engine implement hash-chained approval records, and community specifications describe tamper-evident audit trails. There is, as of this writing, **no ratified standard** for the format of a governance receipt. Treat it as emerging practice rather than as compliance you can certify against.



Governance decision flow from policy evaluation to a hash chained audit trail

A governance layer is also the natural consumer of a decision model, and the shape of the integration is worth describing concretely, because it is where the two ideas meet. A layer of this kind evaluates a proposed action against policy, produces one of the enforcement outcomes, and writes a hash-chained record of the outcome. Compass, Shakudo's governance agent for AI agents, is one example of this pattern: it sits between an agent and the actions it attempts, exposes policy as configuration, pauses high-risk actions for human approval, and maintains a chained audit log. It does not change how the agent reasons. It only governs what the agent is

permitted to do.

The design reason a governance layer wants a decision model rather than a generative one is the same reason a firewall does not want a person behind it. The evaluation happens on every request, the possible answers are few and known in advance, and the record has to be complete. Any component with a 2 percent chance of returning prose is not a component you can put in that position.

The constraint is not hypothetical, and the organizations running into it first are the large ones. Loblaw, Canada's largest retailer, described the same split from the operator's side: what its teams needed was the freedom to choose the best tool for each job **and** an oversight layer strong enough for a company of that scale. That is the argument of this section stated as a requirement rather than as a design preference — the freedom and the control have to arrive together, or the deployment stalls. [Read the Loblaw Digital case study](#).

We needed a platform that gives our teams the freedom to use the best tools available while maintaining the governance and oversight that a company of our scale requires. Shakudo delivers exactly that.

Charu Pujari

Senior Vice President, Engineering and AI, Loblaw

[Read the Loblaw Digital case study](#)

Owning the decision layer: build, buy, or self-host

The category's most consequential split is not accuracy. It is licensing and deployment topology.

The flagship commercial decision model is a closed hosted API. Its vendor has published no weights, no parameter count, and no self-hosting option. For an organization with ordinary data handling requirements that is a normal buy decision. For a regulated bank, an air-gapped defense environment, or a public sector body, it is disqualifying, because it means every routing decision about internal content leaves the perimeter. As one industry write-up puts it, those environments need the pattern rather than the dependency.

This is where the open entrants matter, and the 2026 timing is not a coincidence. Within days of the category becoming visible, AWS, Cloudflare, Convai, and several independent developers published permissively licensed alternatives, and the size range is wide enough to cover a laptop and a data center. The decision layer is one of the few 2026 AI categories where a fully self-hosted stack is available on day one.

Option	Cost profile	Data residency	Operating burden	Fits
Closed hosted API	Usage based, lowest unit cost	Data leaves the perimeter	None	Teams without residency constraints
Vendor-hosted open model	Usage based, higher unit cost	Data leaves the perimeter	Low	Teams wanting licence flexibility
Self-hosted open model	Infrastructure and staff	Full control	High, ongoing	Regulated, air-gapped, sovereign
Open method on existing LLM	Marginal, reuses current estate	Wherever your LLM runs	Moderate	Teams with a large self-hosted estate

Table: Build, buy, and self-host, without the usual framing.

Two cautions belong with the table. First, open licence and open weights are different things. A model published under a permissive licence with weights on Hugging Face is not automatically OSI open source, and at least one 2026 entrant is open-weight while its hosted path costs several times the closed alternative. Read the licence, not the announcement. Second, there is a naming hazard: several unrelated projects share the name of the category's flagship model, and at least one published model card describes that flagship as open-weights, which is false. Verify the artifact you are downloading.

The self-hosting case is not primarily about cost at small scale. It is about owning the question set and the audit record. Running an open decision model inside your own boundary means that the routing decisions about internal content are made inside that boundary, which is the property regulated readers actually need.

Residency is not an abstract concern either, and the clearest evidence is from an operator who chose it deliberately. Whitecap Resources, an oil and gas producer, runs its estate on-premises for a specific reason: to keep control of where its data ends up and which legal framework governs it. Read that against the paragraph above and the point sharpens — the argument for owning the decision layer is not cost at this scale, it is jurisdiction.

[Read the Whitecap Resources case study.](#)

We're an on-prem company because we didn't want our data exposed to laws that we had no control over. Put it in the cloud and it may be subject to a different country's legal framework and exposure to being opened at any time. The company made a decision to keep as much on-prem as possible, because then we had control over where that information ended up.

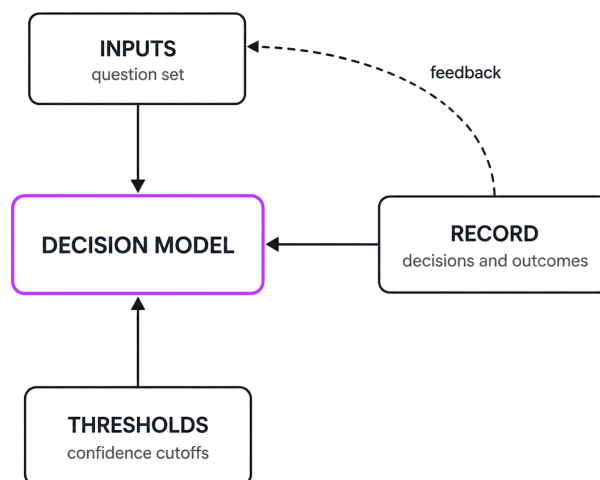
James Wakelin

Director of Business Intelligence @ Whitecap

[Read the Whitecap Resources case study](#)

Operating a decision layer

A decision model is a component with a lifecycle, not a service call. Treating it as the latter is the most common way these deployments decay, because the model is cheap enough to run everywhere and therefore easy to forget.



Operating a decision layer means watching the inputs, the thresholds and the record, not just the model.

Four failure modes recur.

- **Question drift.** The question set is configuration, so it changes without a model release. A question added in March changes the input distribution in a way no version number records.
- **Threshold rot.** The escalation threshold is tuned once against a sample and never revisited as traffic and inputs change.
- **Silent miscalibration.** Stated confidence stops matching observed accuracy, usually because inputs drifted

away from the distribution the calibration was measured on.

- **Unaudited expansion.** New teams adopt the decision tier without anyone owning the aggregate record.

A minimal operating routine that addresses all four:

- ☐ Version the question set as an artifact, with the same review as code.
- ☐ Re-measure calibration on a held-out sample on a fixed schedule, not only at release.
- ☐ Track the escalation rate as a first-class metric, and alarm on sustained movement in either direction.
- ☐ Re-check error correlation between the decision tier and the escalation target before relying on the cascade.
- ☐ Keep the audit record append-only and complete, including decisions that were allowed.
- ☐ Re-baseline against the current regulatory schedule rather than the schedule in last year's design document.

The metric that matters most is not accuracy. It is the **escalation rate**, because it is the only number that tells you simultaneously whether the decision model is competent for the traffic it is seeing and whether you are actually saving the money you built the layer for. A decision model whose escalation rate has crept from 5 percent to 40 percent is still accurate and no longer cheap.

Evaluation checklist

Use this to assess a decision layer, or a decision model as a component, before it carries production traffic.

- ☐ Are the answers genuinely typed and closed, so that a malformed response is unrepresentable?
- ☐ Does every answer come with a confidence value, and has that value been independently calibrated?
- ☐ Is the cost and latency profile measured on your traffic, not taken from a vendor homepage?
- ☐ If you are cascading, have you measured whether the errors are independent?
- ☐ Does the licensing permit your deployment topology, and is it a licence or merely open weights?
- ☐ Can the decision layer be evaluated inline, within the latency budget of the agent it governs?
- ☐ Is the question set versioned, reviewable, and owned by a named team?
- ☐ Is there an append-only record of allowed decisions, not only blocked ones?
- ☐ Is the escalation threshold re-measured on a schedule, with an owner?
- ☐ Does the design hold up against the current regulator timeline, and against the audit trail format you can actually produce?

The decision layer is not a new kind of intelligence. It is a recognition that the enterprise AI stack has two very different jobs, and that they were being served by one component. Reasoning and deciding have different cost structures, different latency budgets, different failure modes, and different correctness requirements. Separating them is unglamorous work. It is also the difference between an agent platform that can be governed and one that merely can be demonstrated.

Ready to Get Started?

Shakudo enables enterprise teams to deploy AI infrastructure with complete data sovereignty and privacy.

shakudo.io

info@shakudo.io

Book a demo: shakudo.io/sign-up