



CASE STUDY · DEEP DIVE EXHIBIT · ~8 MIN

Generic Entity Scoring Engine, Native to the CRM

“The hard problem is not adding an agent. It is deciding which agent - and whether any agent - should act.”

ROLE

Independent AI Systems Researcher

TIMELINE

Independent research · 2025–2026

DOMAIN

Agentic Systems & AI Architecture

KULDEEP SINGH · DIGITAL ARCHITECTURE

PART I · THE CASE

What was built, and why

Independent research prototype built on synthetic CRM-like fixtures - no production records, no employer tooling, no client workflows. I prototyped a generic scoring engine that grades the readiness of any record type with the same deterministic algorithm - completeness, urgency, operational risk - then routes qualifying work to one narrow specialist with a fresh context window, and executes every mutating tool behind JSON-schema validation with an audit envelope. The pattern under study is CRM-adjacent (approvals, escalations, provisioning-style flows) but the fixtures are mine and the code is standalone. Nothing here describes any employer's internal systems.

Independent prototype, synthetic data. I built this because single-agent demos kept failing the same way in my experiments: one bloated prompt, thirty tools, and no notion of whether the record was even ready. The engine is deliberately generic - the same scoring algorithm runs unchanged across different fixture entities, so readiness means the same thing everywhere instead of being redefined per object. I wanted that boring machinery - scoring, routing, gating - treated as first-class architecture instead of prompt tricks.

What goes wrong without it: models get invoked on incomplete records, contexts bloat across domains, tools fire with unvalidated arguments, and nobody can reconstruct why an action happened. In record-centric work that is unacceptable.

CONSTRAINTS

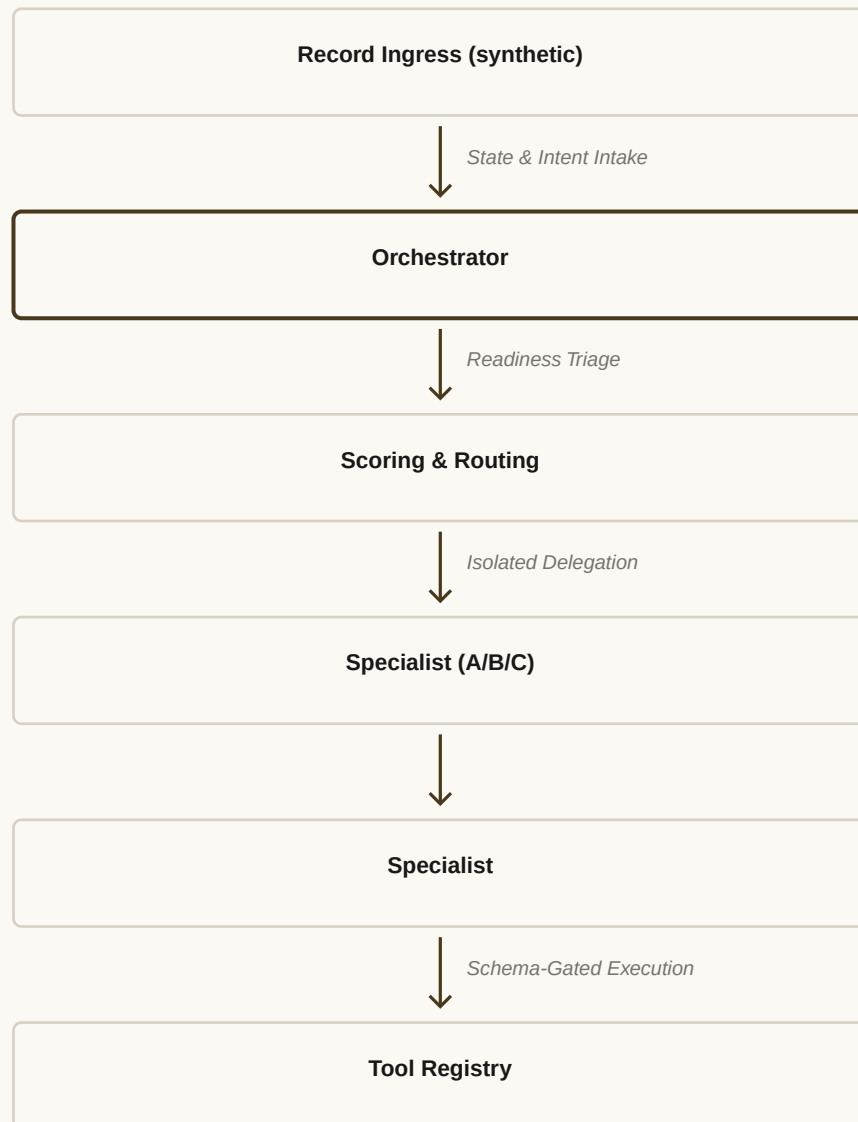
- No model call on unready records - deterministic triage first
- Every mutation behind schema validation and permission checks
- One specialist per task with an isolated context window
- Full reconstruction: scores, routes, and tool I/O all auditable

PART I · ARCHITECTURE

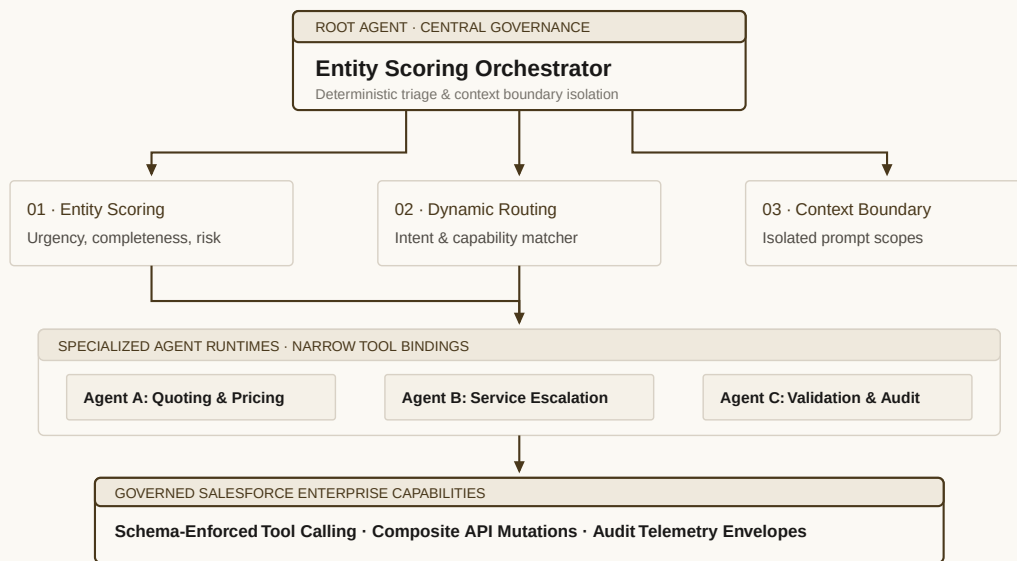
The system as designed

Three stages, deliberately unglamorous: 1) Score - completeness, urgency, and risk computed deterministically over the fixture record. Below threshold, the orchestrator stops and says why. 2) Route - an intent classifier picks exactly one specialist and hands it a fresh, narrow context. 3) Act - the specialist may only call registered tools whose arguments validate against strict schemas; every call and result lands in an audit envelope.

SCORE-FIRST ORCHESTRATOR & SPECIALIST ROUTING



Triage before tokens: score, route to one specialist, and gate every tool call behind schemas.



Full system schematic - Score-First Orchestrator & Specialist Routing.

PART I · DECISIONS

Choices, engineering, tradeoffs

Score-first triage vs. immediate generation

Calling a model on a half-formed record wastes tokens and invites confident nonsense.

CHOSEN

Deterministic scoring gate before any model invocation.

WHY

Cheap arithmetic filters most cases; expensive inference runs only where it can help. My logs showed fewer wasted calls and clearer refusals.

Narrow specialists vs. one mega-prompt

One agent holding quoting, SLA, and billing logic degrades at all three.

CHOSEN

Central router with small, single-domain specialists.

WHY

Small contexts stay sharp. Routing decisions stay visible. Debugging stops being archaeology.

ENGINEERING NOTES

- Built the scorer as pure functions over fixture records - no model involved, fully unit-testable thresholds.
- Implemented the router as intent classification plus capability matching, with explicit abstention when nothing qualifies.
- Registered every tool with a strict JSON schema; validation failures return structured errors, never partial writes.
- Wrapped each run in an audit envelope: input hash, scores, route taken, tool calls, outputs, timestamps.
- Gave each specialist a fresh context assembled from the envelope - never the full conversation history.

PART I · OUTCOMES

Tradeoffs, results, lessons

Routing hops vs. single-shot latency

Gain: Guardrails, visibility, and far fewer runaway actions.

Cost: Extra milliseconds per run for scoring and routing.

Mitigation: Precomputed score vectors and parallel skill-eligibility checks keep the overhead small.

Schema gates vs. freeform tools

Gain: Invalid mutations die at the boundary with a reason attached.

Cost: Each capability needs a written schema and mapping.

Mitigation: A small tool template keeps new capabilities to one file each.

OUTCOMES

- Unready fixtures stop before any model call, with a machine-readable reason
- Specialist contexts stay small and task-shaped across multi-step fixture runs
- Schema gates catch malformed tool arguments before any state change
- Every run reconstructs end to end from its audit envelope

LESSONS

- Score before you generate; most enterprise records are not ready on arrival.
- Small routed contexts outperform large shared ones in every run I measured.

01 · ORIENTATION

An exhibit, not an article

Seven layers, each with something to touch. Start anywhere - the path remembers where you are.

You are not just reading about this architecture. You are going to inspect how it works.

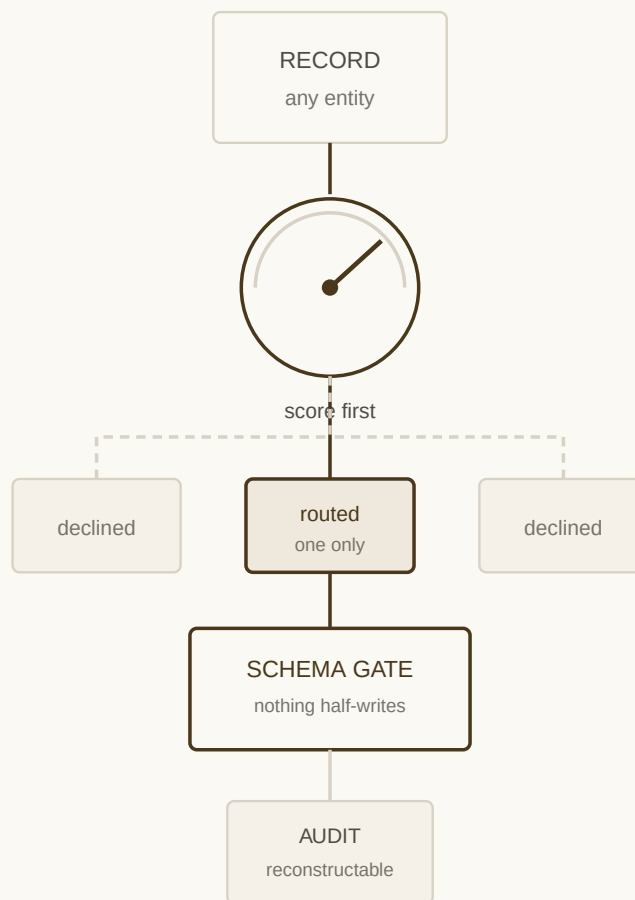


Exhibit plate - the system at a glance.

This exhibit walks through a generic entity scoring engine: one deterministic algorithm that grades any record type, a router that picks exactly one specialist, and schema gates around every mutation. Six short chapters, each with something to touch.

Nothing here is a production system. The fixtures are synthetic, the runs are described rather than executed, and every number comes from the author's own test harness. What is real is the reasoning: why each boundary exists and which failure it absorbs.

How to use this exhibit

- Inspect any component to see its purpose, boundary, and the failure it prevents
- Step one request through the system hop by hop
- Answer two design questions before reading how the architect reasoned
- Try to break the system, then see where each failure lands

02 · THE PROBLEM

Start from the failure

Every boundary in this system answers a failure. Meet the failures first.

Before we look at the architecture, consider the failure it exists to prevent.

Single-agent demos fail the same way every time: one bloated prompt holding quoting logic, service policy, and billing rules at once, thirty tools attached, and no notion of whether the record is even ready for automated handling.

The consequences stack up. Models get invoked on incomplete records, burning tokens to produce confident nonsense. Contexts bloat across domains until nothing is sharp. Tools fire with unvalidated arguments against real records. And when someone asks why an action happened, there is nothing to point at.

In record-centric work that is unacceptable. The full narrative lives on the case study page - here we start from that failure and build outward.

THE CONSTRAINTS THIS FAILURE IMPOSES

- No model call on unready records - deterministic triage first
- Every mutation behind schema validation and permission checks
- One specialist per task with an isolated context window
- Full reconstruction: scores, routes, and tool I/O all auditable

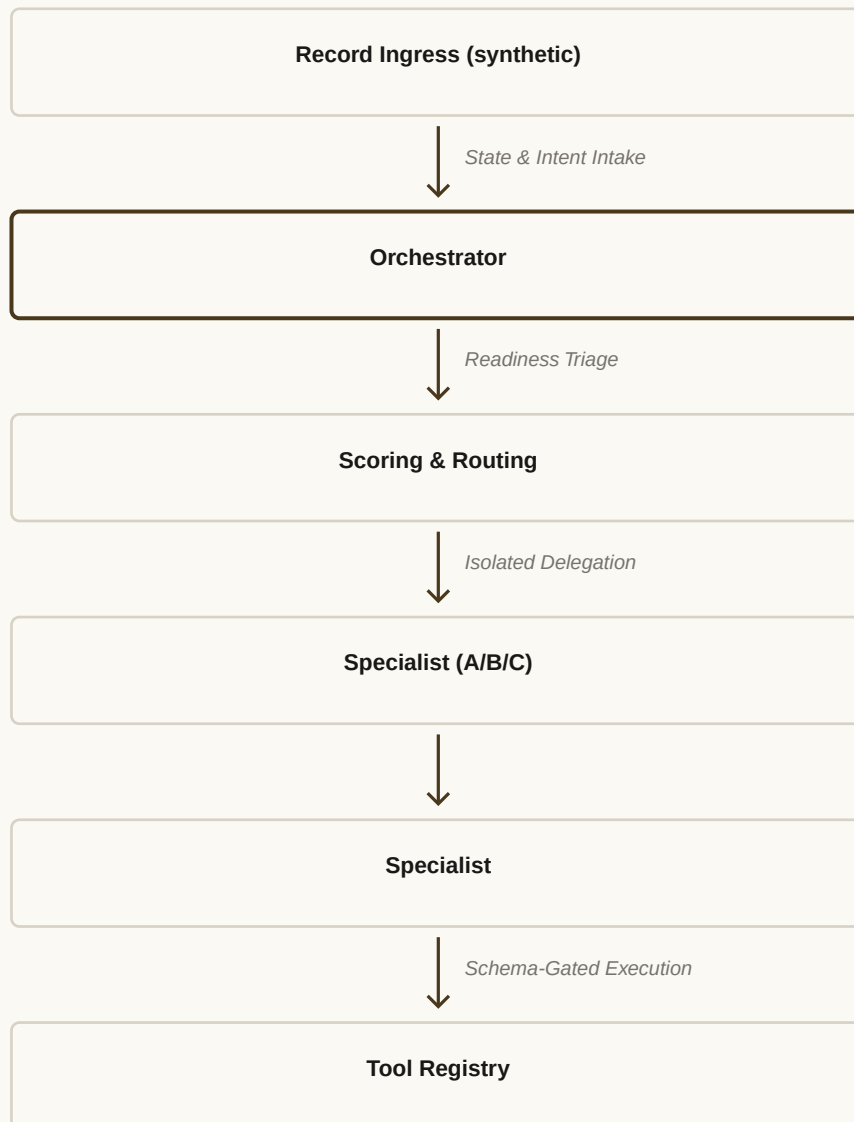
03 · THE ARCHITECTURE

Inspect the system

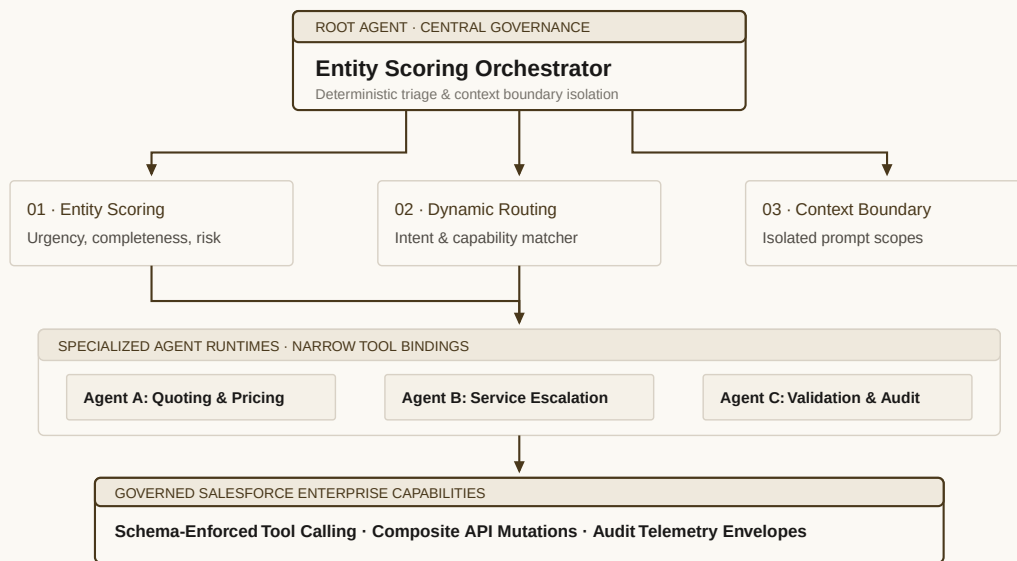
Explore the boundaries that make the architecture work.

Six components. Each one exists because a specific failure demanded it.

SCORE-FIRST ORCHESTRATOR & SPECIALIST ROUTING



Triage before tokens: score, route to one specialist, and gate every tool call behind schemas.



Full system schematic - Score-First Orchestrator & Specialist Routing.

Boundary 01 / 06

Entity Ingress

Capture & normalize entity data.

PURPOSE

Normalize an incoming request and its fixture record into the shape the engine reasons about.

BOUNDARY

It transforms and forwards. It never judges readiness and never invokes a model.

FAILURE PREVENTED

Format drift across entity types, where each stage quietly assumes a different record shape.

Boundary 02 / 06

Entity Intelligence

Score, enrich & assess readiness.

PURPOSE

Grade record readiness with pure functions before a single token is spent.

BOUNDARY

Arithmetic only. No model calls, no judgment calls, no side effects - it scores and steps aside.

FAILURE PREVENTED

Confident model output on half-formed records, plus the token spend that comes with it.

Boundary 03 / 06

Intent Router

Decide the right capability.

PURPOSE

Pick exactly one specialist for qualified work - or decline the work explicitly.

BOUNDARY

It routes. It never executes domain work itself and never fans out to several specialists at once.

FAILURE PREVENTED

Silent misrouting, where the wrong agent acts confidently on a task it half-understands.

Boundary 04 / 06

Task-Specialized Agent

Execute with guardrails.

PURPOSE

Do one domain's work with a fresh, tightly scoped context.

BOUNDARY

One domain per specialist. It never sees the full conversation history and never calls tools outside its registry.

FAILURE PREVENTED

Context bloat and cross-domain hallucination, where quoting logic leaks into billing actions.

Boundary 05 / 06

Governed Tool Boundary

Controlled access to enterprise tools.

PURPOSE

Stand between specialist intent and state change, and reject anything malformed.

BOUNDARY

Validation failures return structured errors. Nothing half-writes, ever.

FAILURE PREVENTED

Invalid mutations reaching the record store - wrong types, missing fields, out-of-scope actions.

Boundary 06 / 06

Audit Envelope

Trace, evaluate and learn.

PURPOSE

Make every run reconstructable long after it finishes.

BOUNDARY

It observes and records. It never alters execution or hides fields for convenience.

FAILURE PREVENTED

The unanswerable 'why did it do that' - the failure mode that kills automation programs quietly.

04 · FOLLOW THE FLOW

One request, six hops

Watch a single request earn its way through, hop by hop.

Now let's follow one request through the system, one hop at a time.

01 Entity Ingress

A request arrives with a fixture record. Nothing is assumed about completeness - the envelope just normalizes the shape.

02 Entity Intelligence

Pure functions grade completeness, urgency, and risk. Below threshold the run stops here with a machine-readable reason, and no model is ever invoked.

03 Intent Router

Intent classification plus capability matching selects exactly one specialist. If nothing qualifies, the orchestrator abstains rather than guessing.

04 Task-Specialized Agent

The specialist gets a fresh, narrow context assembled from the envelope - never the full history - and plans its tool calls.

05 Governed Tool Boundary

Each requested call validates arguments against its JSON schema and permission scope. Failures return structured errors; nothing half-writes.

06 Audit Envelope

Input hash, scores, route, tool I/O, and timestamps land in the envelope. The run now reconstructs end to end, days later if needed.

05 · ARCHITECTURE DECISIONS

The choices that hold it up

Two decisions, each with the alternative it beat - then your turn.

You have seen the topology. Here are the two decisions that hold it up - each with the alternative it beat.

Score-first triage vs. immediate generation

Calling a model on a half-formed record wastes tokens and invites confident nonsense.

CHOSEN

Deterministic scoring gate before any model invocation.

WHY

Cheap arithmetic filters most cases; expensive inference runs only where it can help. My logs showed fewer wasted calls and clearer refusals.

ALTERNATIVES CONSIDERED

- Prompt directly on the raw record
- Hardcoded triggers with no model at all

Narrow specialists vs. one mega-prompt

One agent holding quoting, SLA, and billing logic degrades at all three.

CHOSEN

Central router with small, single-domain specialists.

WHY

Small contexts stay sharp. Routing decisions stay visible. Debugging stops being archaeology.

ALTERNATIVES CONSIDERED

- Single prompt with 30+ tools
- Uncoordinated agent swarm

CHALLENGE THE DESIGN

A new request arrives on a half-complete record. What is your first move?

INVOKE THE MODEL IMMEDIATELY

Tempting - the model is the powerful part. But on a half-formed record it spends tokens to produce confident nonsense, and you cannot distinguish its guesses from knowledge.

APPLY HARDCODED RULES, NO MODEL AT ALL

Safe and cheap, until the rules meet a record they were not written for. Pure rules do not degrade gracefully; they just stop covering reality.

SCORE THE RECORD FIRST, THEN DECIDE

This is the architecture's answer. Cheap arithmetic filters most cases and states its reason; expensive inference runs only where it can actually help.

HOW THE ARCHITECT REASONED

The logs settled it: scoring first cut wasted model calls and turned silent guesses into explicit refusals with reasons attached. Triage before tokens.

One agent could hold quoting, service policy, and billing knowledge. Ship it?

SHIP THE SINGLE MEGA-PROMPT

One prompt, thirty tools, three domains. It degrades at all three simultaneously, and debugging means archaeology through one giant context.

LET UNCOORDINATED AGENTS COLLABORATE

Autonomy without a router is just several agents guessing at once. Nobody owns the route, so nobody can explain it later.

ROUTE TO NARROW SPECIALISTS

This is the architecture's answer. Small contexts stay sharp, routing decisions stay visible, and each specialist is boring enough to trust.

HOW THE ARCHITECT REASONED

A central router with single-domain specialists confines each context window to one job. The routing decision is recorded, so precision and observability arrive together.

06 · FAILURE MODES

Let's try to break it

Five failures, five landing places. That is the design working.

Now let's try to break it. Each failure below lands somewhere specific - that is the design working.

The half-complete record

WHAT GOES WRONG

A request arrives below the readiness threshold - missing fields, stale state, ambiguous intent.

WHERE IT LANDS

The scorer stops the run before any model call and returns a machine-readable reason. Cost of the failure: arithmetic.

The task that spans three domains

WHAT GOES WRONG

A request touches quoting, service policy, and billing in one breath - the shape that bloats a shared context.

WHERE IT LANDS

The router isolates it to one specialist with a fresh, narrow context. The other domains never enter the window.

The wrong specialist

WHAT GOES WRONG

Classification confidence is low, or the chosen specialist cannot cover the task.

WHERE IT LANDS

The orchestrator abstains with a stated reason, and the audit envelope records the route that was considered - misroutes become visible, reviewable events.

The malformed tool call

WHAT GOES WRONG

A specialist emits tool arguments with wrong types, missing fields, or out-of-scope actions.

WHERE IT LANDS

The schema gate rejects the call with a structured error before any state change. Nothing half-writes, ever.

The unanswerable 'why did it do that'

WHAT GOES WRONG

Days later, someone must reconstruct a run - for debugging, review, or accountability.

WHERE IT LANDS

The audit envelope replays input hash, scores, route, and tool I/O in order. Reconstruction is a read operation, not an investigation.

STANDING COMPROMISES

Routing hops vs. single-shot latency

Gain: Guardrails, visibility, and far fewer runaway actions.

Cost: Extra milliseconds per run for scoring and routing.

Mitigation: Precomputed score vectors and parallel skill-eligibility checks keep the overhead small.

Schema gates vs. freeform tools

Gain: Invalid mutations die at the boundary with a reason attached.

Cost: Each capability needs a written schema and mapping.

Mitigation: A small tool template keeps new capabilities to one file each.

07 · REFLECTION

What remains true

Limits, stated plainly - and what changes next.

You have seen the topology, the decisions, and the failure boundaries. Two honest notes before you leave.

This engine is strongest where the world is legible: records with definable completeness, tasks with classifiable intent, tools with writable schemas. Where those hold, the boring machinery outperforms clever prompting every time.

Its limits are stated, not hidden. Thresholds are hand-tuned on fixtures, runs are single-request rather than conversational, and the router classifies rather than retrieves. Each limit is a named next experiment, not a surprise.

WHAT I WOULD CHANGE TODAY

- Learn score thresholds from labeled fixture outcomes instead of hand-tuning them
- Retrieve over capability documentation so the tool registry scales past hand-written entries
- Extend the envelope across multi-turn runs without letting history leak back into specialist contexts

VERIFIED OUTCOMES

- Unready fixtures stop before any model call, with a machine-readable reason
- Specialist contexts stay small and task-shaped across multi-step fixture runs
- Schema gates catch malformed tool arguments before any state change
- Every run reconstructs end to end from its audit envelope

More in the Deep Dive Series

[01 · Generic Entity Scoring Engine, Native to the CRM](#)

[02 · BrewCMS: Open-Source Content Control Plane & Agent Operating System](#)

[03 · Local Evaluation of Salesforce RCA - CML](#)

[04 · TM Forum Open API Intelligence Suite & Spectra AI Copilot](#)

[05 · WebAssembly in Salesforce LWC: 500K Record Engine & Net-Zero DB](#)

[06 · Multi-Site Quoting Without Leaving the CRM](#)

Continue the exploration

[Interactive exhibit: victorkuldeep.com/case-studies/scoring-intelligence-agent/deep-dive](https://victorkuldeep.com/case-studies/scoring-intelligence-agent/deep-dive)

[Long-form essays: victorkuldeep.com/thinking](https://victorkuldeep.com/thinking)

[Ask AXIOM: victorkuldeep.com/ask](https://victorkuldeep.com/ask)

[Connect: victorkuldeep.com/connect](https://victorkuldeep.com/connect)



Kuldeep Singh

Principal Technical Architect · Salesforce Industries · Revenue Cloud (RCA) · Agentic AI

