ENKIDU

MIOVSKI GROUP

AGENTIC AI · PROOF OF CONCEPT

Implementation Journey

From conceptual architecture to as-built system: the challenges we met, how we resolved them, and how the architecture evolved

A trust-assured, multi-agent capability-maturity assessment system, built end to end on a single NVIDIA DGX Spark workstation for Environment and Climate Change Canada. This document traces the implementation from the approved conceptual architecture through the proven Phase 6 vertical slice — the hardware and platform configuration, the model experimentation and substitutions, the plug-and-play container architecture, and, at the centre, the trust harness: chain of evidence, provenance, traceability, and full observability wrapped around every agent inference and behaviour. Prepared as a reference of demonstrated experience to accompany our submission.

Sources: Agentic AI PoC — Architecture Document (conceptual) · As-Built Architecture, Rev A (2026-05-14, Phase 6 complete, Phase 7 in progress) · the project repository

VERSION · FIRST DRAFT FOR REVIEW
16 JULY 2026
PREPARED BY MIOVSKI GROUP
CLASSIFICATION · UNCLASSIFIED

Contents

1. Executive Summary 2

2. Purpose and Reading Guide 2

3. The Starting Point — What the Conceptual Architecture Planned 3

4. Standing Up the Platform — Hardware, Containers, and Build Tooling 4

- **4.1 The environment as designed** 4
- **4.2 Challenge — a dual-Python container the platform did not anticipate** 5
- **4.3 Challenge — the dependency graph that defeated pip** 5
- **4.4 Challenge — container lifecycle and the bind-mount sharp edge** 5

5. The Model Journey — Adaptability Under a Hardware Envelope 6

- **5.1 Attempt one — Llama-3.3-Nemotron-Super-49B-v1.5** 6
- **5.2 Attempt two — Llama 3 8B, and the tool-calling bar** 6
- **5.3 The landing — Qwen3-32B, and engineering to its envelope** 6
- **5.4 The second model — embeddings as a separate small-model service** 7

6. Building the Evidence Base — The Deterministic Substrate 7

- **6.1 Challenge — the organizational model changed underneath the build** 7
- **6.2 Challenge — tagging chunks to the org structure, and honest data quality** 8

7. The MCP Logic Layer — Plug-and-Play by Construction 8

- **7.1 Challenge — debugging across process boundaries** 9
- **7.2 The plug-and-play inventory** 9

8. The Main Objective — The Trust Harness in Practice 10

- **8.1 The keystone — a model proposes, an engine commits** 11
- **8.2 Chain of evidence, provenance, and the audit walk-back** 11
- **8.3 Wrapping inference and behaviour with observability** 12
- **8.4 The proof — the Phase 6 vertical slice** 12

9. The Deviations Ledger — Challenge, Resolution, Architectural Change 12

10. What the Journey Demonstrates 14

11. Glossary 14

1. Executive Summary

Between the approval of the conceptual architecture and the close of the Phase 6 vertical slice, our team designed, configured, built, and proved an end-to-end multi-agent AI system on a single small-footprint workstation — the NVIDIA DGX Spark — for Environment and Climate Change Canada's Business Capability Maturity Assessment proof of concept. The system reads an organization's own documentation and produces deterministic, citation-backed CMMI maturity scores, with every score carrying its full derivation: from the committed decision, back through the model's proposal, to the cited evidence items, to the exact character spans of the exact source documents.

This document tells the implementation story honestly: what was planned, what we encountered when the plan met real hardware, real model behaviour, and real data, how each challenge was resolved, and how those resolutions changed the architecture. None of the changes were regressions; each was a deliberate engineering response, recorded at the time in an as-built deviations ledger. The result is a working system whose central premise — determinism where it matters, reasoning where it helps — survived every change intact, because it was enforced structurally rather than by asking the model nicely.

The journey demonstrates five things in practice: (1) hands-on configuration of small-footprint AI hardware and its full serving, orchestration, and observability stack; (2) adaptability in model selection — three reasoning models evaluated on-device until one met the bar for reliable tool calling and structured output within the hardware envelope; (3) disciplined experimentation with small and mid-sized language models, including the concrete engineering consequences of an 8K-token context window; (4) a containerized, plug-and-play architecture in which every model and service sits behind a standard interface and can be swapped by configuration, not code; and (5) — the main objective of the PoC — a trust harness that wraps every agent inference and behaviour with typed contracts, mandatory validated citations, a deterministic scoring engine, a persistent chain of evidence, and end-to-end observability of the full agent topology.

2. Purpose and Reading Guide

This Implementation Journey is a companion to two source documents: the original Agentic AI PoC Architecture Document, which set the intent (a seven-agent topology, deterministic scoring, MCP-based tool integration), and the As-Built Architecture dossier (Rev A), which reconciles that intent against the code on disk at the close of Phase 6. Where those documents describe the system, this one describes the path: the sequence of build phases, the friction encountered, the decisions taken, and the architectural consequences of each.

It is written as a reference of demonstrated experience. Every challenge and resolution described here is drawn from the as-built record and the project repository — not reconstructed after the

fact. Deviations from the original architecture are presented deliberately and completely, because the ability to encounter friction on constrained hardware, diagnose it, and resolve it without compromising the trust guarantees is precisely the experience this document evidences.

Status note for the reviewer: *this draft describes the build through the Phase 6 vertical slice and the Phase 7 work in progress, per the As-Built Rev A baseline. If subsequent phases (evaluation harness, remaining agents, full-model widening) should be reflected in the final version, that status can be added to Sections 8 and 9 on your direction.*

3. The Starting Point – What the Conceptual Architecture Planned

The conceptual architecture defined the problem — assess business-capability maturity from a heterogeneous corpus of interview transcripts, process documents, incident reports, and audit findings — and a design premise that carries the whole system: determinism where it matters, reasoning where it helps. Scoring formulas, rubrics, the capability model, and the organizational structure are code-enforced ground truth; the language model interprets, extracts, and narrates, but never performs arithmetic and never assigns a final score. Everything runs on local infrastructure: source documents contain sensitive organizational information, so nothing is transmitted to an external API.

The planned stack, as specified before the build began:

LAYER	PLANNED SELECTION	PLANNED RATIONALE
Hardware	NVIDIA DGX Spark (GB10 Grace Blackwell, 128 GB unified CPU/GPU memory)	Data sovereignty on one on-premise workstation; capacity for a ~49B model with headroom for embeddings, vectors, and observability; pre-integrated NVIDIA stack
Reasoning model	Llama-3.3-Nemotron-Super-49B-v1.5 via NIM	Trained for structured tool calling; strong reasoning; deployable on-device; OpenAI-compatible API keeps the runtime swappable
Agentic framework	NVIDIA NeMo Agent Toolkit (NAT)	Framework-agnostic composition, native MCP client, built-in evaluation and tracing hooks
Vector memory	Chroma	Embedded/lightweight; right-sized for thousands of chunks; rich metadata filtering for capability/type/weight constraints

LAYER	PLANNED SELECTION	PLANNED RATIONALE
Observability	Arize Phoenix (OpenTelemetry / OpenInference)	Open-source, single container, all trace data stays local; RAG-specific groundedness evaluators
Tool integration	Model Context Protocol (MCP) servers, incl. custom Capability Model, Rubric/Scoring, and Report Builder servers	Deterministic logic in code behind narrow, typed tool signatures — never in prompts
Orchestration pattern	Supervisor/worker with seven agents (Supervisor, Retrieval, Evidence Extraction, Scoring, Critique, Aggregator, Narrative)	Predictable, cost-bounded, traceable; separation of duties between extraction, scoring, and narrative
Relational store	SQLite (upgradeable to PostgreSQL)	Capability model, rubrics, run history, and scores are relational knowledge, distinct from vector memory

TABLE 1 · THE PLANNED STACK. NEARLY ALL OF IT SURVIVED CONTACT WITH REALITY; WHERE IT DID NOT, SECTIONS 4–7 EXPLAIN EXACTLY WHAT CHANGED AND WHY.

4. Standing Up the Platform – Hardware, Containers, and Build Tooling

The first stretch of the journey was pure platform engineering: turning a fresh DGX Spark into a reproducible, containerized development and serving environment. Three challenges arrived immediately, and each resolution became a permanent, documented part of the build.

4.1 The environment as designed

The project runs as an NVIDIA AI Workbench project: Workbench manages a development container that mounts the repository, and a Compose stack runs the model-serving and observability services alongside it. In-container, services resolve by name: the NIM reasoning endpoint on port 8000, a local embedding service on port 8002, Phoenix on 6006 (UI) with OTLP ingest on 4317/4318, and Chroma and SQLite running in-process against on-disk state. The DGX Spark's 128 GB of unified CPU/GPU memory holds the reasoning model, the embedding model, the vector store, and the observability stack concurrently, with headroom — and because the hardware is dedicated, inference latency is predictable, which matters for multi-agent orchestration.

4.2 Challenge – a dual-Python container the platform did not anticipate

CHALLENGE · The Workbench base image ships system Python 3.10, but the NeMo Agent Toolkit requires Python 3.11+, and the Workbench UI does not permit swapping the base image. Meanwhile JupyterLab and Workbench's own configuration packs depend on the system Python staying exactly where it is.

RESOLUTION · The container build installs Python 3.12 alongside 3.10 and routes all project code through a virtual environment at /opt/venv, leaving system Python untouched for the platform tooling. The arrangement is scripted in the container's post-build step, so it is reproducible on any rebuild rather than a hand-crafted machine state.

ARCHITECTURAL IMPACT · A deliberate dual-runtime container design, encoded in the build script — and a pattern reused wherever the platform's defaults and the toolchain's requirements diverge: satisfy both, in code, reproducibly.

4.3 Challenge – the dependency graph that defeated pip

CHALLENGE · The combined dependency graph — Phoenix, LangChain, ChromaDB, OpenTelemetry, MCP, sentence-transformers, and nvidia-nat together — defeated pip's backtracking resolver, which failed with resolution-too-deep after roughly eight minutes of resolution attempts. On an aarch64 platform there was a second trap: installing sentence-transformers naively pulls the CPU-only PyTorch wheel from PyPI, silently discarding the GPU.

RESOLUTION · The build bootstraps uv, which resolves the same graph in seconds, and routes every project-dependency install through it. PyTorch is installed first, explicitly from the CUDA-13 aarch64 wheel index, so sentence-transformers picks up the GPU wheel rather than the CPU default.

ARCHITECTURAL IMPACT · Build tooling changed from pip to uv — recorded as deviation №3 in the as-built ledger — and the install order became an architectural fact of the build script, not tribal knowledge.

4.4 Challenge – container lifecycle and the bind-mount sharp edge

CHALLENGE · The original architecture wanted NIM and Phoenix managed as Workbench Apps, sharing the project's network namespace and lifecycle. Getting the Compose stack right surfaced a sharp edge: Workbench Compose does not inherit the user's HOME environment, so relative or variable-based bind-mount paths silently expand to empty strings.

RESOLUTION · Both services run as a documented Compose stack with absolute bind-mount paths, GPU reservations, shared-memory sizing, health checks (including a generous start period for first-time NIM model loading), and restart policies. For the PoC they run as host-side containers; promoting them to managed Workbench Apps is a tracked, consciously deferred operator-runbook

task — the current arrangement works but does not survive host reboots cleanly, and saying so plainly is part of the as-built discipline.

ARCHITECTURAL IMPACT · Infrastructure-as-code for the full serving and observability stack, with its known limitation recorded in the deferred-work ledger rather than discovered by the next operator.

5. The Model Journey – Adaptability Under a Hardware Envelope

The single most instructive stretch of the journey was reasoning-model selection. The plan named one model; the build evaluated three on-device before committing. The sequence, and what each step taught, is the clearest demonstration in this document of adapting models to get the best result on constrained hardware.

5.1 Attempt one – Llama-3.3-Nemotron-Super-49B-v1.5

The architecture's choice. In practice, pulling and serving the Nemotron NIM container failed on the DGX Spark — the packaged container would not serve on this platform. Rather than fight the packaging, the team treated the model as what the architecture always said it was: a replaceable component behind an OpenAI-compatible API.

5.2 Attempt two – Llama 3 8B, and the tool-calling bar

A smaller Llama 3 8B NIM served correctly — proof the serving path worked — but it did not support OpenAI-style tool/function calling reliably enough. That is a disqualifying failure in this architecture, because every agent depends on structured tool calls and schema-valid JSON output. The experiment established the selection bar precisely: not benchmark scores, but reliable structured-output and tool-calling behaviour on this hardware, in this serving stack, for these agents.

5.3 The landing – Qwen3-32B, and engineering to its envelope

Qwen3-32B via NIM met the bar: solid tool-calling and structured-output behaviour with a comfortable GPU fit inside the Spark's unified memory. It came with a real trade-off — a maximum context length of 8,192 tokens, far below the Nemotron's 32K+ — and the team engineered to that envelope rather than pretending it away:

- **Dynamic token budgeting.** The shared LLM helper computes max_tokens dynamically against the remaining context budget on every call, so a long prompt can never silently truncate the response.

- **Retrieval sized to the window.** Retrieval pulls 5–10 chunks per query rather than the 20+ a 32K-context model would take — a deliberate quality-density decision, not a limitation discovered in production.
- **Thinking-mode discipline.** Qwen3 ships with 'thinking mode' enabled by default; for structured-output tasks it burns tokens internally and multiplies latency with no quality benefit. The NAT workflow configuration disables it via chat_template_kwargs; applying the same flag in the plain-Python LLM helper is a tracked pre-flight item — an example of the deferred-work ledger catching a small inconsistency before it becomes a mystery.

The strategic lesson the journey proves: because every agent talks to the model through the standard OpenAI client against a NIM endpoint, three model changes cost zero agent-code changes. Model selection was configuration plus measurement — exactly the plug-and-play property the architecture promised.

5.4 The second model – embeddings as a separate small-model service

The architecture's supporting decision — never use the reasoning model for embeddings — was implemented as its own small-model service: BAAI/bge-large-en-v1.5 (1,024-dimensional vectors) behind a compact OpenAI-compatible FastAPI shim on port 8002. Chunking defaults (1,500 target characters with 200 overlap) were tuned to the embedder's 512-token sequence limit. As with the reasoning model, swapping the embedding model is an environment-variable change. Two models, two workloads, two right-sized services — the small-language-model discipline in miniature.

6. Building the Evidence Base – The Deterministic Substrate

Before any agent ran, the build stood up the deterministic substrate: the reference-data seed flow and the document ingestion pipeline. Both are pure data processing — no model calls, no agent orchestration — and both are idempotent by design: re-running either against unchanged inputs produces no new rows. This is what guarantees the agents reason over a stable, reproducible evidence base, and it is the first layer of the chain of evidence: every chunk carries a content-derived document id, an ordinal, character offsets, and organizational metadata, so a citation resolves to an exact span of an exact source.

6.1 Challenge – the organizational model changed underneath the build

CHALLENGE · Early phases seeded the organizational structure from a maturity-template spreadsheet using dotted identifiers (A.1, A.2). Mid-Phase 6, the canonical source of truth changed to programs.xlsx — the real financial-centre hierarchy, with six-digit codes, 15 branches, 94 directorates, and 344 divisions, laid out in a sparse hierarchical Excel format where cells inherit from the row above.

RESOLUTION · A dedicated loader parses the sparse layout with forward-fill and first-occurrence deduplication (a directorate can legitimately appear under several program contexts); the schema gained Division as a first-class level and denormalized program-context columns so a prompt can say "this directorate works on X under Y" without a join. Seeding is split so the org tables can be reset and re-seeded without ever touching the independently-sourced capability taxonomy.

ARCHITECTURAL IMPACT · The unit of assessment became the real organizational identifier scheme. One consequence was consciously deferred: the Report Builder, written against dotted ids, needs updating to the six-digit scheme before report generation is wired in — tracked in the ledger with the exact coupling documented, because the Phase 6 slice ends at a committed decision and never exercises it.

6.2 Challenge – tagging chunks to the org structure, and honest data quality

CHALLENGE · Per-directorate scoring requires knowing which directorate each chunk belongs to — but source documents do not announce their org structure uniformly. One interview document had clean directorate-name section headers; another used interviewee-name headers and never named directorates at all.

RESOLUTION · A new tag stage was inserted into the ingestion pipeline — between load and chunk — that segments text into sections by canonical org-structure name matches (using the Capability Model's own name-lookup tools), chunks each section separately, and stamps every chunk with its section's directorate, branch, and division metadata, with a branch-level fallback where no directorate matches. The pipeline is seven stages as built: discover → load → document → tag → chunk → embed → persist. Ingestion of the two interview corpora produced 685 chunks; 112 tagged cleanly to the target directorate — enough to run and prove the slice — and the second document's zero-directorate result was recorded as a named data-quality gap with two candidate fixes (a header-inserting pre-pass, or a curated interviewee→directorate override map), scheduled for a later phase.

ARCHITECTURAL IMPACT · The pipeline gained a stage; chunk identifiers deliberately reflect the document's structure under the current organizational model; and a data-quality limitation was measured, quantified, and tracked instead of papered over — the same honesty discipline the scoring engine applies to evidence.

7. The MCP Logic Layer – Plug-and-Play by Construction

Three custom Model Context Protocol servers hold the code-enforced ground truth: the Capability Model MCP (pure lookups over the taxonomy and org structure — no LLM, no mutation from agents), the Rubric/Scoring MCP (the CMMI rubric projected onto a capability, plus the

deterministic scoring engine), and the Report Builder MCP (deliverable assembly from committed decisions). Each server is a thin FastMCP wrapper over an importable module — a design choice that paid off directly when transport friction arrived.

7.1 Challenge – debugging across process boundaries

CHALLENGE · The original design has agents reach MCP tools over the stdio subprocess transport. During slice development, multi-process debugging — several spawned servers, interleaved failures — slowed diagnosis exactly when fast iteration mattered most.

RESOLUTION · Both paths were kept. The Phase 6 agent slice imports the servers' underlying modules directly, in-process — one process, one stack trace — while a NAT workflow definition spawns all three servers as stdio subprocesses exactly as designed and points a ReAct agent at the full tool set against the local Qwen3-32B NIM. The transport path is therefore proven, continuously exercisable, and moving the agents onto it is a wiring change, not a rewrite, because both paths run the same module code.

ARCHITECTURAL IMPACT · Recorded as a reversible-by-configuration deviation. The MCP contracts — not the transport — are the architecture.

7.2 The plug-and-play inventory

By the close of the slice, every substitutable component sat behind a standard seam. This is the concrete meaning of "multiple containers and models making the architecture plug and play":

COMPONENT	INTERFACE IT SITS BEHIND	HOW IT SWAPS
Reasoning model (Qwen3-32B)	OpenAI-compatible chat API served by a NIM container	Configuration only — proven three times during selection (Nemotron → Llama 3 8B → Qwen3-32B) with zero agent-code changes
Embedding model (bge-large-en-v1.5)	OpenAI-compatible embeddings API behind a FastAPI shim container	Environment variables (endpoint, model, dimension)
Vector store (Chroma)	A thin store wrapper owned by the project	Embedded PersistentClient today; swaps to HttpClient service mode with no change to callers
Relational store (SQLite)	Store classes owning their DDL	Simple schema; PostgreSQL port tracked as a later-phase task measured in hours
Deterministic logic (3 MCP servers)	MCP tool contracts over FastMCP; also directly	In-process for the slice; stdio subprocess transport proven by the

COMPONENT	INTERFACE IT SITS BEHIND	HOW IT SWAPS
	importable modules	NAT workflow — moving between them is configuration
Observability (Phoenix)	OpenTelemetry OTLP endpoint; OpenInference auto-instrumentation	Single container; endpoint and project name are environment settings; tracing can be disabled by flag for tests
Orchestration (plain Python slice)	Typed agent classes with run() methods; NAT workflow YAML alongside	Agent code is portable; lifting orchestration into NAT YAML is a later wiring change, with the MCP smoke-test workflow already demonstrating the pattern

TABLE 2 · EVERY MODEL AND SERVICE BEHIND A STANDARD SEAM. THE ARCHITECTURE'S PROMISE — SWAP BY CONFIGURATION, NOT CODE — WAS EXERCISED REPEATEDLY DURING THE BUILD, NOT MERELY ASSERTED.

8. The Main Objective – The Trust Harness in Practice

Everything above is the stage; this is the play. The PoC's central objective was to demonstrate that agent inference and behaviour can be wrapped end to end in a trust harness: a chain of evidence, provenance and traceability for every claim, deterministic commitment of every consequential decision, and observability of the entire agent topology. As built, that harness rests on four load-bearing invariants, each enforced by code:

INVARIANT	HOW THE CODE ENFORCES IT
1. Determinism owns the score	The LLM proposes a level; engine.evaluate() commits it. The engine is, by construction, the single point in the entire system at which a score becomes COMMITTED — no other code path can write a committed level. The model builds a ScoreProposal; the engine returns a ScoreDecision; the committed level on that decision is the only value the system recognises.
2. Every handoff is typed	No free-form dictionary crosses an agent or tool boundary — only Pydantic-validated models (RunRequest → RetrievalResult → EvidenceBundle → ScoreProposal → ScoreDecision → RunResult), validated on the receiving side. Invalid items are dropped at the boundary, not discovered downstream.
3. Citations are mandatory and validated	Every evidence item carries a required chunk_id. The Evidence Extraction agent filters every id the model emits against the set of chunks actually retrieved in this run — a hallucinated citation

INVARIANT	HOW THE CODE ENFORCES IT
	cannot survive the filter, so 'every claim resolves to a real source' is an enforced property, not a hope.
4. The topology is observable	Agent boundaries emit AGENT spans, deterministic helpers emit TOOL spans, and OpenInference auto-instruments every model call as an LLM span nested under its calling agent. Phoenix shows the run as a trace tree — the structure of the system, not a flat log — and a regression points straight at the span that produced it.

TABLE 3 · THE FOUR LOAD-BEARING INVARIANTS. THEY ARE RESTATED WHERE THEY BITE IN THE AS-BUILT DOSSIER, BUT COLLECTED HERE BECAUSE THEY ARE THE ARCHITECTURE.

8.1 The keystone – a model proposes, an engine commits

The transition from ScoreProposal to ScoreDecision is the architectural keystone. When the Scoring agent has read the rubric and the evidence, it proposes the CMMI level the evidence will defensibly support — the prompt is explicit that it should be conservative — and submits the proposal to the deterministic engine. The engine applies four rules in order: citations mandatory; the level inside the 1–5 range (a defence-in-depth schema guard); evidence sufficiency scaled to the level claimed; and referential integrity — every cited id must exist. The sufficiency ladder is explicit:

PROPOSED LEVEL	EVIDENCE THE ENGINE REQUIRES BEFORE COMMITTING
L1 — Initial	≥ 1 evidence item, any polarity
L2–L3 — Managed / Defined	≥ 2 items, ≥ 1 positive
L4 — Quantitatively Managed	≥ 3 items, ≥ 2 positive, and a measurement/metrics signal
L5 — Optimizing	≥ 4 items, ≥ 3 positive, and measurement and continuous-improvement signals

TABLE 4 · THE EVIDENCE-SUFFICIENCY RULES. A PROPOSAL THAT FAILS RETURNS DEFERRED — THE SYSTEM REFUSES TO COMMIT AND SURFACES THE GAP FOR HUMAN REVIEW RATHER THAN GUESSING; A RANGE VIOLATION RETURNS REJECTED. THE ENGINE NEVER ADJUSTS A LEVEL: IT COMMITS EXACTLY WHAT WAS PROPOSED, OR DECLINES.

8.2 Chain of evidence, provenance, and the audit walk-back

The harness makes every committed score auditable by walk-back. A decision persists to the score_decisions table under its run id; the decision carries the evidence chunk ids; each evidence item carries its quote, assertion, and polarity against a required chunk_id; each chunk carries its document id, ordinal, page, and character offsets; and each document id is a content hash of the

source text. For any score the system commits, you can therefore walk backwards — decision, to proposal, to evidence items, to chunk ids, to the exact character spans of the exact source documents — and identify precisely where the language model touched the chain and where it could not. Memory is layered to protect this: the knowledge base (Chroma plus the relational reference tables) is written only by ingestion and seeders, never by agents; working memory is the typed handoff objects within a run; run memory is the persisted decisions table.

8.3 Wrapping inference and behaviour with observability

Observability was designed to make the multi-agent system legible, not merely logged. Importing the project's configuration package wires up tracing exactly once, so any entry point gets Phoenix instrumentation automatically. The Supervisor emits the root AGENT span; each worker nests beneath it; every deterministic helper call — embed_query, vector_search, render_rubric, engine_evaluate, decisions-store upsert — is a TOOL span; every model call auto-traces with full prompt, response, token counts, and latency. Two deliberate refinements show the discipline: embedding calls are suppressed from instrumentation (deterministic, high-volume, and a rate-limiting risk during bulk ingestion), and the span structure falls out of the code by construction — adding a future agent means wrapping its run() in agent_span(), nothing more.

8.4 The proof – the Phase 6 vertical slice

The slice's success criterion was one working capability × directorate combination, end to end. The run assessed capability INF.DQA (Data Quality & Accuracy) for directorate 332000 (Wildlife Enforcement): retrieval filtered to the 112 chunks tagged to that directorate; the Evidence Extraction agent produced five cited evidence items, all resolving to real chunks; the Scoring agent proposed; and the engine COMMITTED at CMMI level 1, with the full RunResult persisted and the entire topology visible as a single Phoenix trace tree.

A committed level 1 is the correct result for this input, not a weak one: the evidence supports a defensible floor, and the engine committed exactly what the rules permit. The system declined to over-claim — which is the behaviour the whole architecture is built to produce, and the single most important line in this document. The trust harness did not just constrain the model; it demonstrably shaped the outcome toward defensibility.

9. The Deviations Ledger – Challenge, Resolution, Architectural Change

Every departure from the conceptual architecture was recorded as it was made. None is a regression; each was a deliberate response to hardware, tooling, or data friction. Consolidated:

AREA	ORIGINAL ARCHITECTURE	AS BUILT	WHY — THE CHALLENGE IT RESOLVED
Reasoning model	Llama-3.3-Nemotron-Super-49B	Qwen3-32B (via a Llama 3 8B intermediate)	The Nemotron NIM would not serve on DGX Spark; Llama 3 8B lacked reliable tool calling. Cost accepted and engineered for: an 8,192-token context
Orchestration	NAT YAML workflow	Plain Python slice; NAT YAML retained for MCP smoke tests	Explicit control flow is far more debuggable than a YAML ReAct loop; agent code stays portable
Build tooling	pip	uv	The combined dependency graph hit pip's resolution-too-deep; uv resolves it in seconds
Python runtime	Single system Python	Dual 3.10 + 3.12 with /opt/venv	nvidia-nat needs 3.11+; the base image ships 3.10 and cannot be swapped
Org-structure source	Maturity template, dotted ids (A.1)	programs.xlsx, six-digit financial-centre codes; Division added as a level	The richer canonical hierarchy — the codes are the real organizational identifiers
MCP transport (slice)	stdio subprocess transport	Direct in-process import; stdio path exercised by the NAT workflow	One process, one stack trace during slice development; both paths run the same module code
Service hosting	NIM & Phoenix as Workbench Apps	Host-side Docker containers	Expedient for the build; the clean-lifecycle promotion is tracked, with the reboot limitation stated

TABLE 5 · THE AS-BUILT DEVIATIONS LEDGER. TWO ENTRIES (ORCHESTRATION AND MCP TRANSPORT) ARE REVERSIBLE BY CONFIGURATION — THE CODE WAS WRITTEN SO THAT MOVING ONTO NAT ORCHESTRATION AND THE MCP TRANSPORT IS WIRING, NOT REWRITING.

Alongside it, a deferred-work ledger tracks everything consciously postponed with its target phase — the Qwen3 thinking-mode flag in the LLM helper, the evaluation harness build-out, calibration of the L4/L5 keyword heuristics against gold data, the second corpus's directorate-tagging gap, the remaining three agents, widening beyond the single validated combination, the Report Builder id-scheme update, tagger refinement, the MCP transport move, service-lifecycle promotion, and the

eventual SQLite-to-PostgreSQL port. Nothing on it is forgotten; everything on it slots into the existing structure. The most important property of the journey's end state is that the architecture did not have to bend to reach it.

10. What the Journey Demonstrates

- **Small-footprint AI hardware, configured hands-on.** A DGX Spark (GB10 Grace Blackwell, 128 GB unified memory, aarch64, CUDA 13) taken from unboxing to a reproducible containerized stack: NIM model serving, a local embedding service, Phoenix observability, vector and relational stores, and an AI Workbench development container — including the platform-specific friction (dual Python runtimes, aarch64 GPU wheels, Compose environment sharp edges) that only shows up when you actually do the work.
- **Small and mid-sized language models, implemented with their constraints engineered for.** A 32B reasoning model and a compact embedding model serving concurrently on one workstation, with the 8K context window treated as an engineering input: dynamic token budgeting, retrieval sized to the window, chunking tuned to the embedder's sequence limit, and inference-mode flags managed deliberately.
- **Adaptability, evidenced by the model odyssey.** Three models evaluated on-device against a precise, architecture-driven selection bar — reliable tool calling and structured output on this hardware — with each change costing zero agent-code modifications because the seams were designed for it.
- **A plug-and-play, multi-container architecture.** Every model and service behind a standard interface — OpenAI-compatible APIs, MCP tool contracts, store wrappers, OTLP — with the swap property exercised repeatedly during the build rather than claimed on a diagram.
- **The trust harness as the centre of gravity.** Agent inference and behaviour wrapped end to end: typed contracts at every boundary, mandatory and validated citations, a deterministic engine as the sole writer of committed decisions, a persistent chain of evidence supporting character-level walk-back, layered memory that agents cannot corrupt, and a trace tree that exposes the full agent topology. Proven by a run whose most telling result was restraint: the system committed the level the evidence defensibly supported, and no more.

Taken together, the journey is evidence of a team that plans rigorously, meets friction honestly, resolves it without compromising the trust guarantees, and records every deviation so the plan and the system never silently diverge — the same discipline our forward proposals commit to.

11. Glossary

TERM	DEFINITION

TERM	DEFINITION
Agentic AI	An AI system in which language-model-driven components (agents) plan, call tools, and hand work to one another to accomplish a task, rather than answering a single prompt.
Agent span / Tool span / LLM span	The three span kinds in this system's traces: one per agent boundary, one per deterministic helper call, and one (auto-emitted) per model call, nesting into a single trace tree per run.
AI Workbench (NVIDIA)	NVIDIA's managed development environment that builds and runs the project's development container and Compose services.
bge-large-en-v1.5	The compact open embedding model (1,024-dimensional vectors, 512-token sequence limit) served locally for all vector generation.
Chain of evidence	The unbroken, persisted linkage from a committed score back through its proposal and cited evidence items to exact character spans of source documents.
Chroma	The vector database holding embedded document chunks; run embedded in-process for the PoC, keyed by chunk id so it joins cleanly to the relational store.
Chunk	A contiguous span of a source document — with document id, ordinal, character offsets, and organizational metadata — that is embedded, retrieved, and cited.
CMMI	Capability Maturity Model Integration: the five-level maturity scale (Initial, Managed, Defined, Quantitatively Managed, Optimizing) used for scoring.
DGX Spark	NVIDIA's desktop-class AI workstation built on the GB10 Grace Blackwell superchip with 128 GB of unified CPU/GPU memory — the small-footprint hardware the entire PoC runs on.
Directorate	The ECCC organizational unit of assessment, identified by a six-digit financial-centre code (e.g., 332000).
Embedding	A numeric vector representation of text used for semantic similarity search; produced here by a dedicated small model, never by the reasoning model.
FastMCP	The lightweight Python framework used to expose each MCP server; each server is a thin wrapper over an importable module.
Guardrails	Code-enforced constraints on agent inputs, outputs, and tool use — as distinct from instructions in prompts.

TERM	DEFINITION
Idempotent	Producing the same result when re-run against unchanged inputs — the property that makes ingestion and seeding reproducible.
MCP (Model Context Protocol)	An open standard for exposing tools and data to AI systems through typed tool contracts; the PoC built three custom MCP servers.
NAT (NeMo Agent Toolkit)	NVIDIA's agentic framework for composing agents, tools, and workflows; used here as the MCP client for the transport smoke tests, with the slice orchestrated in plain Python.
NIM (NVIDIA Inference Microservice)	A containerized, optimized model-serving service exposing an OpenAI-compatible API; how the reasoning model is served.
OpenAI-compatible API	The de-facto standard chat/embeddings HTTP interface; every model in the system sits behind one, which is what makes models swappable by configuration.
OpenInference / OpenTelemetry (OTLP)	The instrumentation convention and telemetry protocol used to capture every model call and span, shipped to Phoenix.
Phoenix (Arize Phoenix)	The open-source observability backend that stores and visualizes the trace trees; runs as a single local container so all trace data stays on-device.
Provenance	The recorded origin of every datum and every inference — which document, which chunk, which run, which model call produced it.
Pydantic	The Python validation library used for every schema in the system; the mechanism behind 'every handoff is typed.'
Qwen3-32B	The 32-billion-parameter open reasoning model selected after on-device evaluation; served via NIM with an 8,192-token context window.
RAG (Retrieval-Augmented Generation)	Supplying a model with retrieved source material to ground its output; here, retrieval is deterministic and directorate-filtered.
ScoreProposal / ScoreDecision	The keystone pair: the model's proposed level with rationale and evidence, and the deterministic engine's committed/deferred/rejected verdict — the only score the system recognises.
SLM (Small Language Model)	A compact language model deployable on constrained hardware; in this PoC, the embedding model and, relative to frontier models, the 32B reasoning model.

TERM	DEFINITION
Supervisor/worker pattern	The orchestration shape: a supervisor validates inputs and drives specialist worker agents in a fixed, traceable sequence.
Trace tree	The nested span structure of one run in Phoenix — the agent topology made visible, with a regression pointing at the span that produced it.
Trust harness	The combined set of enforced mechanisms — typed contracts, validated citations, deterministic commitment, layered memory, and observability — that wraps agent inference and behaviour.
uv	The fast Python package resolver/installer adopted after pip failed to resolve the project's dependency graph.
Vector store / vector search	A database of embeddings queried by semantic similarity; how evidence-candidate chunks are found.
Vertical slice	A minimal but complete end-to-end path through every layer of the system — the Phase 6 deliverable and proof.