ENKIDU

MIOVSKI GROUP

undefined

1. Zusammenfassung

Zwischen der Genehmigung der konzeptionellen Architektur und dem Abschluss des vertikalen Schnitts der Phase 6 hat unser Team ein End-to-End-Multi-Agenten-KI-System auf einer einzigen platzsparenden Workstation — der NVIDIA DGX Spark — für den Proof of Concept zur Bewertung der Geschäftsfähigkeitsreife von Environment and Climate Change Canada entworfen, konfiguriert, gebaut und bewiesen. Das System liest die eigene Dokumentation einer Organisation und erzeugt deterministische, zitatgestützte CMMI-Reifewerte, wobei jeder Wert seine vollständige Herleitung trägt: von der festgelegten Entscheidung zurück über den Vorschlag des Modells zu den zitierten Nachweiselementen bis zu den exakten Zeichenbereichen der exakten Quelldokumente.

Dieses Dokument erzählt die Implementierungsgeschichte ehrlich: was geplant war, worauf wir stießen, als der Plan auf reale Hardware, reales Modellverhalten und reale Daten traf, wie jede Herausforderung gelöst wurde und wie diese Lösungen die Architektur veränderten. Keine der Änderungen war ein Rückschritt; jede war eine bewusste ingenieurmäßige Antwort, seinerzeit in einem As-Built-Abweichungsregister festgehalten. Das Ergebnis ist ein funktionierendes System, dessen zentrale Prämisse — Determinismus, wo es zählt, Schlussfolgern, wo es hilft — jede Änderung intakt überstand, weil sie strukturell durchgesetzt wurde, statt das Modell freundlich zu bitten.

Die Reise demonstriert fünf Dinge in der Praxis: (1) die praktische Konfiguration platzsparender KI-Hardware und ihres gesamten Serving-, Orchestrierungs- und Beobachtbarkeits-Stacks; (2) Anpassungsfähigkeit bei der Modellauswahl — drei Schlussfolgerungsmodelle auf dem Gerät bewertet, bis eines die Schwelle für zuverlässige Werkzeugaufrufe und strukturierte Ausgabe innerhalb der Hardware-Hülle erreichte; (3) disziplinierte Experimente mit kleinen und mittelgroßen Sprachmodellen, einschließlich der konkreten ingenieurmäßigen Folgen eines Kontextfensters von 8K Tokens; (4) eine containerisierte, Plug-and-Play-Architektur, in der jedes Modell und jeder Dienst hinter einer Standardschnittstelle sitzt und per Konfiguration, nicht per Code, ausgetauscht werden kann; und (5) — das Hauptziel des PoC — ein Vertrauensgeschirr, das jede Agenteninferenz und jedes Agentenverhalten mit typisierten Verträgen, verpflichtenden validierten Zitaten, einer deterministischen Bewertungs-Engine, einer persistenten Nachweiskette und End-to-End-Beobachtbarkeit der gesamten Agententopologie umschließt.

2. Zweck und Leseleitfaden

Diese Implementierungsreise ist ein Begleitdokument zu zwei Quelldokumenten: dem ursprünglichen Architekturdokument des agentischen KI-PoC, das die Absicht festlegte (eine Sieben-Agenten-Topologie, deterministische Bewertung, MCP-basierte Werkzeugintegration), und dem As-Built-Architekturdossier (Rev. A), das diese Absicht mit dem Code auf der Festplatte zum Abschluss der Phase 6 abgleicht. Wo jene Dokumente das System beschreiben, beschreibt dieses den Weg: die Abfolge der Bauphasen, die aufgetretene Reibung, die getroffenen Entscheidungen und die architektonischen Folgen jeder einzelnen.

Es ist als Referenz nachgewiesener Erfahrung geschrieben. Jede hier beschriebene Herausforderung und Lösung stammt aus dem As-Built-Protokoll und dem Projekt-Repository — nicht im Nachhinein rekonstruiert. Abweichungen von der ursprünglichen Architektur werden bewusst und vollständig dargestellt, denn die Fähigkeit, auf beschränkter Hardware auf Reibung zu stoßen, sie zu diagnostizieren und zu lösen, ohne die Vertrauensgarantien zu gefährden, ist genau

die Erfahrung, die dieses Dokument belegt.

Statushinweis für den Prüfer: Dieser Entwurf beschreibt den Bau bis zum vertikalen Schnitt der Phase 6 und die laufende Arbeit der Phase 7 gemäß der As-Built-Rev.-A-Basislinie. Sollen spätere Phasen (Bewertungsprüfstand, verbleibende Agenten, Vollmodell-Verbreiterung) in der Endfassung berücksichtigt werden, kann dieser Status auf Ihre Anweisung hin den Abschnitten 8 und 9 hinzugefügt werden.

3. Der Ausgangspunkt — was die konzeptionelle Architektur plante

Die konzeptionelle Architektur definierte das Problem — die Bewertung der Geschäftsfähigkeitsreife aus einem heterogenen Korpus von Interviewtranskripten, Prozessdokumenten, Vorfallberichten und Prüfungsfeststellungen — und eine Entwurfsprämisse, die das ganze System trägt: Determinismus, wo es zählt, Schlussfolgern, wo es hilft. Bewertungsformeln, Rubriken, das Fähigkeitsmodell und die Organisationsstruktur sind code-durchgesetzte Grundwahrheit; das Sprachmodell interpretiert, extrahiert und erzählt, führt aber niemals Arithmetik aus und vergibt niemals einen Endwert. Alles läuft auf lokaler Infrastruktur: Quelldokumente enthalten sensible organisatorische Informationen, also wird nichts an eine externe API übertragen.

TABELLE 1 · DER GEPLANTE STACK (SCHICHT · AUSWAHL · BEGRÜNDUNG)

Hardware	NVIDIA DGX Spark (GB10 Grace Blackwell, 128 GB einheitlicher CPU/GPU-Speicher). Datenhoheit auf einer einzigen lokalen Workstation; Kapazität für ein ~49-B-Modell mit Spielraum für Embeddings, Vektoren und Beobachtbarkeit; vorintegrierter NVIDIA-Stack.
Schlussfolgerungsmodell	Llama-3.3-Nemotron-Super-49B-v1.5 über NIM. Für strukturierte Werkzeugaufrufe trainiert; starkes Schlussfolgern; auf dem Gerät einsetzbar; eine OpenAI-kompatible API hält die Laufzeit austauschbar.
Agentisches Rahmenwerk	NVIDIA NeMo Agent Toolkit (NAT). Rahmenwerk-agnostische Komposition, nativer MCP-Client, integrierte Bewertungs- und Tracing-Haken.
Vektorspeicher	Chroma. Eingebettet/leichtgewichtig; passend dimensioniert für Tausende von Fragmenten; reichhaltige Metadatenfilterung für Fähigkeits-/Typ-/Gewichtsbeschränkungen.
Beobachtbarkeit	Arize Phoenix (OpenTelemetry / OpenInference). Open Source, einzelner Container, alle Trace-Daten bleiben lokal; RAG-spezifische Fundiertheits-Evaluatoren.
Werkzeugintegration	Model-Context-Protocol-(MCP-)Server, inkl. eigener Server für Fähigkeitsmodell, Rubrik/Bewertung und Report-Builder. Deterministische Logik im Code hinter schmalen, typisierten Werkzeugsignaturen — niemals in Prompts.
Orchestrierungsmuster	Supervisor/Worker mit sieben Agenten (Supervisor, Retrieval, Nachweisextraktion, Bewertung, Kritik, Aggregator, Narrativ). Vorhersehbar, kostenbegrenzt, nachvollziehbar; Aufgabentrennung zwischen Extraktion, Bewertung und Narrativ.
Relationaler Speicher	SQLite (auf PostgreSQL erweiterbar). Fähigkeitsmodell, Rubriken, Lauf-Historie und Werte sind relationales Wissen, getrennt vom

Vektorspeicher.

Fast alles überstand den Kontakt mit der Realität; wo nicht, erklären die Abschnitte 4–7 genau, was sich änderte und warum.

4. Die Plattform aufbauen — Hardware, Container und Build-Werkze

Der erste Abschnitt der Reise war reine Plattform-Ingenieurskunst: eine frische DGX Spark in eine reproduzierbare, containerisierte Entwicklungs- und Serving-Umgebung zu verwandeln. Drei Herausforderungen kamen sofort, und jede Lösung wurde ein dauerhafter, dokumentierter Teil des Baus.

4.1 Die Umgebung wie entworfen

Das Projekt läuft als NVIDIA-AI-Workbench-Projekt: Workbench verwaltet einen Entwicklungscontainer, der das Repository einbindet, und ein Compose-Stack führt die Modell-Serving- und Beobachtbarkeitsdienste daneben aus. Im Container lösen sich Dienste über den Namen auf: der NIM-Schlussfolgerungsendpunkt auf Port 8000, ein lokaler Embedding-Dienst auf Port 8002, Phoenix auf 6006 (UI) mit OTLP-Aufnahme auf 4317/4318, und Chroma und SQLite laufen im Prozess gegen den Zustand auf der Festplatte. Die 128 GB einheitlicher CPU/GPU-Speicher der DGX Spark halten das Schlussfolgerungsmodell, das Embedding-Modell, den Vektorspeicher und den Beobachtbarkeits-Stack gleichzeitig, mit Spielraum — und weil die Hardware dediziert ist, ist die Inferenzlatenz vorhersehbar, was für die Multi-Agenten-Orchestrierung wichtig ist.

4.2 Herausforderung — ein Dual-Python-Container, den die Plattform nicht vorsah

Herausforderung Das Workbench-Basis-Image liefert System-Python 3.10, aber das NeMo Agent Toolkit erfordert Python 3.11+, und die Workbench-UI erlaubt keinen Wechsel des Basis-Images. Zugleich hängen JupyterLab und Workbenchs eigene Konfigurationspakete davon ab, dass das System-Python genau dort bleibt, wo es ist.

Lösung Der Container-Build installiert Python 3.12 neben 3.10 und leitet den gesamten Projektcode über eine virtuelle Umgebung unter /opt/venv, wodurch System-Python für die Plattform-Werkzeuge unangetastet bleibt. Die Anordnung ist im Post-Build-Schritt des Containers geskriptet, also bei jedem Rebuild reproduzierbar statt ein handgefertigter Maschinenzustand.

Architektonische Auswirkung Ein bewusster Dual-Laufzeit-Container-Entwurf, im Build-Skript kodiert — und ein Muster, das überall wiederverwendet wird, wo die Standardwerte der Plattform und die Anforderungen der Toolchain auseinandergehen: beide erfüllen, im Code, reproduzierbar.

4.3 Herausforderung — der Abhängigkeitsgraph, der pip besiegte

Herausforderung Der kombinierte Abhängigkeitsgraph — Phoenix, LangChain, ChromaDB, OpenTelemetry, MCP, sentence-transformers und nvidia-nat zusammen — besiegte pips Backtracking-Resolver, der nach rund acht Minuten mit

resolution-too-deep scheiterte. Auf einer aarch64-Plattform gab es eine zweite Falle: sentence-transformers naiv zu installieren zieht das CPU-only-PyTorch-Wheel von PyPI und verwirft still die GPU.

Lösung Der Build bootet uv, das denselben Graphen in Sekunden auflöst, und leitet jede Projekt-Abhängigkeitsinstallation darüber. PyTorch wird zuerst installiert, ausdrücklich aus dem CUDA-13-aarch64-Wheel-Index, damit sentence-transformers das GPU-Wheel statt der CPU-Voreinstellung aufgreift.

Architektonische Auswirkung Die Build-Werkzeuge wechselten von pip zu uv — als Abweichung ?3 im As-Built-Register verzeichnet — und die Installationsreihenfolge wurde zu einer architektonischen Tatsache des Build-Skripts, kein Stammeswissen.

4.4 Herausforderung — Container-Lebenszyklus und die scharfe Kante des Bind-Mount

Herausforderung Die ursprüngliche Architektur wollte NIM und Phoenix als Workbench-Apps verwaltet, die den Netzwerk-Namespace und den Lebenszyklus des Projekts teilen. Den Compose-Stack richtig hinzubekommen brachte eine scharfe Kante zutage: Workbench Compose erbt nicht die HOME-Umgebung des Nutzers, sodass relative oder variablenbasierte Bind-Mount-Pfade still zu leeren Zeichenketten expandieren.

Lösung Beide Dienste laufen als dokumentierter Compose-Stack mit absoluten Bind-Mount-Pfaden, GPU-Reservierungen, Shared-Memory-Dimensionierung, Health-Checks (einschließlich einer großzügigen Startperiode für das erstmalige Laden des NIM-Modells) und Neustart-Richtlinien. Für den PoC laufen sie als host-seitige Container; ihre Beförderung zu verwalteten Workbench-Apps ist eine verfolgte, bewusst zurückgestellte Betreiber-Runbook-Aufgabe — die aktuelle Anordnung funktioniert, übersteht aber Host-Neustarts nicht sauber, und das klar zu sagen ist Teil der As-Built-Disziplin.

Architektonische Auswirkung Infrastruktur als Code für den gesamten Serving- und Beobachtbarkeits-Stack, mit seiner bekannten Einschränkung im Register der zurückgestellten Arbeit verzeichnet, statt vom nächsten Betreiber entdeckt zu werden.

5. Die Modellreise — Anpassungsfähigkeit unter einer Hardware-Hü

Der lehrreichste Abschnitt der Reise war die Auswahl des Schlussfolgerungsmodells. Der Plan nannte ein Modell; der Build bewertete drei auf dem Gerät, bevor er sich festlegte. Die Abfolge und was jeder Schritt lehrte, ist die klarste Demonstration in diesem Dokument, Modelle anzupassen, um auf beschränkter Hardware das beste Ergebnis zu erzielen.

5.1 Erster Versuch — Llama-3.3-Nemotron-Super-49B-v1.5

Die Wahl der Architektur. In der Praxis scheiterte das Ziehen und Servieren des Nemotron-NIM-Containers auf der DGX Spark — der paketierte Container ließ sich auf dieser Plattform nicht servieren. Statt gegen die Paketierung zu kämpfen, behandelte das Team das Modell als das, was die Architektur immer gesagt hatte: eine austauschbare Komponente hinter einer

OpenAI-kompatiblen API.

5.2 Zweiter Versuch — Llama 3 8B, und die Werkzeugaufruf-Schwelle

Ein kleineres Llama-3-8B-NIM servierte korrekt — Beweis, dass der Serving-Pfad funktionierte — unterstützte aber OpenAI-artige Werkzeug-/Funktionsaufrufe nicht zuverlässig genug. Das ist ein disqualifizierendes Versagen in dieser Architektur, weil jeder Agent von strukturierten Werkzeugaufrufen und schemavalider JSON-Ausgabe abhängt. Das Experiment legte die Auswahlschwelle präzise fest: nicht Benchmark-Werte, sondern zuverlässiges Verhalten bei strukturierter Ausgabe und Werkzeugaufrufen auf dieser Hardware, in diesem Serving-Stack, für diese Agenten.

5.3 Die Landung — Qwen3-32B, und das Engineering zu seiner Hülle

Qwen3-32B über NIM erreichte die Schwelle: solides Verhalten bei Werkzeugaufrufen und strukturierter Ausgabe mit bequemem GPU-Sitz im einheitlichen Speicher der Spark. Es kam mit einem echten Kompromiss — einer maximalen Kontextlänge von 8.192 Tokens, weit unter den 32K+ des Nemotron — und das Team betrieb Engineering zu dieser Hülle, statt sie wegzuwünschen:

- Dynamische Token-Budgetierung. Der gemeinsame LLM-Helfer berechnet max_tokens bei jedem Aufruf dynamisch gegen das verbleibende Kontextbudget, sodass ein langer Prompt die Antwort niemals still abschneiden kann.
- Auf das Fenster dimensioniertes Retrieval. Das Retrieval zieht 5 bis 10 Fragmente pro Abfrage statt der 20+, die ein Modell mit 32K-Kontext nähme — eine bewusste Qualitätsdichte-Entscheidung, keine im Produktivbetrieb entdeckte Einschränkung.
- Denkmodus-Disziplin. Qwen3 wird mit standardmäßig aktiviertem ?Denkmodus" geliefert; für Aufgaben mit strukturierter Ausgabe verbrennt er intern Tokens und vervielfacht die Latenz ohne Qualitätsnutzen. Die NAT-Workflow-Konfiguration deaktiviert ihn über chat_template_kwargs; dasselbe Flag im reinen Python-LLM-Helfer anzuwenden ist ein verfolgter Pre-Flight-Punkt — ein Beispiel dafür, wie das Register der zurückgestellten Arbeit eine kleine Inkonsistenz abfängt, bevor sie zum Rätsel wird.

Die strategische Lektion, die die Reise beweist: Weil jeder Agent über den Standard-OpenAI-Client gegen einen NIM-Endpunkt mit dem Modell spricht, kosteten drei Modellwechsel null Änderungen am Agentencode. Die Modellauswahl war Konfiguration plus Messung — genau die Plug-and-Play-Eigenschaft, die die Architektur versprach.

5.4 Das zweite Modell — Embeddings als separater Kleinmodell-Dienst

Die unterstützende Entscheidung der Architektur — niemals das Schlussfolgerungsmodell für Embeddings zu verwenden — wurde als eigener Kleinmodell-Dienst umgesetzt: BAAI/bge-large-en-v1.5 (1.024-dimensionale Vektoren) hinter einem kompakten OpenAI-kompatiblen FastAPI-Shim auf Port 8002. Die Chunking-Standardwerte (1.500 Zielzeichen mit 200 Überlappung) wurden auf die 512-Token-Sequenzgrenze des Embedders abgestimmt. Wie beim Schlussfolgerungsmodell ist der Wechsel des Embedding-Modells eine Änderung einer Umgebungsvariablen. Zwei Modelle, zwei Arbeitslasten, zwei passend dimensionierte Dienste — die Kleinmodell-Disziplin im Kleinen.

6. Die Nachweisbasis bauen — das deterministische Substrat

Bevor ein Agent lief, richtete der Build das deterministische Substrat ein: den Referenzdaten-Seed-Fluss und die Dokumenten-Aufnahme-Pipeline. Beide sind reine Datenverarbeitung — keine Modellaufrufe, keine Agentenorchestrierung — und beide sind per Entwurf idempotent: eine erneute Ausführung gegen unveränderte Eingaben erzeugt keine neuen Zeilen. Das garantiert, dass die Agenten über eine stabile, reproduzierbare Nachweisbasis schlussfolgern, und es ist die erste Schicht der Nachweiskette: jedes Fragment trägt eine inhaltsabgeleitete Dokument-ID, eine Ordnungszahl, Zeichen-Offsets und organisatorische Metadaten, sodass ein Zitat sich zu einem exakten Bereich einer exakten Quelle auflöst.

6.1 Herausforderung — das Organisationsmodell änderte sich unter dem Build

Herausforderung Frühe Phasen bestückten die Organisationsstruktur aus einer Reifegrad-Vorlagen-Tabelle mit gepunkteten Kennungen (A.1, A.2). Mitten in Phase 6 wechselte die kanonische Wahrheitsquelle zu programs.xlsx — der echten Finanzzentren-Hierarchie mit sechsstelligen Codes, 15 Zweigen, 94 Direktionen und 344 Abteilungen, in einem dünn besetzten hierarchischen Excel-Format, in dem Zellen von der Zeile darüber erben.

Lösung Ein dedizierter Loader parst das dünne Layout mit Forward-Fill und Erst-Vorkommen-Deduplizierung (eine Direktion kann legitim unter mehreren Programmkontexten erscheinen); das Schema erhielt Abteilung als erstklassige Ebene und denormalisierte Programmkontext-Spalten, sodass ein Prompt sagen kann ?diese Direktion arbeitet an X unter Y" ohne Join. Das Seeding ist aufgeteilt, sodass die Org-Tabellen zurückgesetzt und neu bestückt werden können, ohne je die unabhängig bezogene Fähigkeits-Taxonomie zu berühren.

Architektonische Auswirkung Die Bewertungseinheit wurde das echte organisatorische Kennungsschema. Eine Folge wurde bewusst zurückgestellt: der Report-Builder, gegen die gepunkteten IDs geschrieben, muss vor der Verdrahtung der Berichterzeugung auf das sechsstellige Schema aktualisiert werden — im Register mit der exakt dokumentierten Kopplung verfolgt, weil der Phase-6-Schnitt an einer festgelegten Entscheidung endet und ihn nie ausübt.

6.2 Herausforderung — Fragmente an die Org-Struktur kennzeichnen, und ehrliche Datenqualität

Herausforderung Die Bewertung je Direktion erfordert zu wissen, zu welcher Direktion jedes Fragment gehört — aber Quelldokumente kündigen ihre Org-Struktur nicht einheitlich an. Ein Interviewdokument hatte saubere Abschnittsüberschriften mit Direktionsnamen; ein anderes nutzte Überschriften mit Befragtennamen und nannte nie Direktionen.

Lösung Eine neue Kennzeichnungsstufe wurde in die Aufnahme-Pipeline eingefügt — zwischen Laden und Chunking — die Text nach kanonischen Org-Struktur-Namensübereinstimmungen (mithilfe der eigenen Namenssuchwerkzeuge des Fähigkeitsmodells) in Abschnitte segmentiert, jeden

Abschnitt separat chunkt und jedes Fragment mit den Direktions-, Zweig- und Abteilungs-Metadaten seines Abschnitts stempelt, mit einem Rückfall auf Zweigebene, wo keine Direktion passt. Die Pipeline hat as-built sieben Stufen: entdecken - laden - Dokument - kennzeichnen - chunken - einbetten - persistieren. Die Aufnahme der beiden Interviewkorpora erzeugte 685 Fragmente; 112 kennzeichneten sich sauber zur Zieldirektion — genug, um den Schnitt auszuführen und zu beweisen — und das Null-Direktions-Ergebnis des zweiten Dokuments wurde als benannte Datenqualitätslücke mit zwei Kandidaten-Fixes (ein überschrifteneinfügender Vorlauf oder eine kuratierte Befragter-Direktion-Übersteuerungskarte) verzeichnet, für eine spätere Phase geplant.

Architektonische Auswirkung Die Pipeline erhielt eine Stufe; Fragment-Kennungen spiegeln bewusst die Struktur des Dokuments unter dem aktuellen Organisationsmodell wider; und eine Datenqualitätsbeschränkung wurde gemessen, quantifiziert und verfolgt, statt übertüncht — dieselbe Ehrlichkeitsdisziplin, die die Bewertungs-Engine auf Nachweise anwendet.

7. Die MCP-Logikschicht — Plug-and-Play durch Konstruktion

Drei eigene Model-Context-Protocol-Server halten die code-durchgesetzte Grundwahrheit: das Fähigkeitsmodell-MCP (reine Nachschläge über die Taxonomie und Org-Struktur — kein LLM, keine Mutation durch Agenten), das Rubrik/Bewertungs-MCP (die auf eine Fähigkeit projizierte CMMI-Rubrik plus die deterministische Bewertungs-Engine) und das Report-Builder-MCP (Zusammenstellung von Ergebnissen aus festgelegten Entscheidungen). Jeder Server ist ein dünner FastMCP-Wrapper über einem importierbaren Modul — eine Entwurfsentscheidung, die sich direkt auszahlte, als Transportreibung auftrat.

7.1 Herausforderung — Debugging über Prozessgrenzen hinweg

Herausforderung Der ursprüngliche Entwurf lässt Agenten MCP-Werkzeuge über den stdio-Subprozess-Transport erreichen. Während der Schnitt-Entwicklung verlangsamte Mehrprozess-Debugging — mehrere gestartete Server, verschachtelte Fehler — die Diagnose genau dann, als schnelle Iteration am wichtigsten war.

Lösung Beide Pfade wurden beibehalten. Der Phase-6-Agentenschnitt importiert die zugrunde liegenden Module der Server direkt, im Prozess — ein Prozess, ein Stacktrace — während eine NAT-Workflow-Definition alle drei Server als stdio-Subprozesse genau wie entworfen startet und einen ReAct-Agenten auf den vollen Werkzeugsatz gegen das lokale Qwen3-32B-NIM richtet. Der Transportpfad ist daher bewiesen, kontinuierlich ausübbar, und die Agenten darauf zu verschieben ist eine Verdrahtungsänderung, keine Neuschreibung, weil beide Pfade denselben Modulcode ausführen.

Architektonische Auswirkung Als per Konfiguration umkehrbare Abweichung verzeichnet. Die MCP-Verträge — nicht der Transport — sind die Architektur.

7.2 Das Plug-and-Play-Inventar

Zum Abschluss des Schnitts saß jede austauschbare Komponente hinter einer Standard-Naht. Das

ist die konkrete Bedeutung von ?mehrere Container und Modelle, die die Architektur Plug-and-Play machen":

Schlussfolgerungsmodell (Qwen3-32B)	Hinter einer OpenAI-kompatiblen Chat-API, von einem NIM-Container serviert. Wechselt nur per Konfiguration — dreimal während der Auswahl bewiesen (Nemotron - Llama 3 8B - Qwen3-32B) mit null Änderungen am Agentencode.
Embedding-Modell (bge-large-en-v1.5)	Hinter einer OpenAI-kompatiblen Embeddings-API hinter einem FastAPI-Shim-Container. Wechselt per Umgebungsvariablen (Endpunkt, Modell, Dimension).
Vektorspeicher (Chroma)	Hinter einem dünnen, dem Projekt gehörenden Speicher-Wrapper. Heute eingebetteter PersistentClient; wechselt in den HttpClient-Dienstmodus ohne Änderung für die Aufrufer.
Relationaler Speicher (SQLite)	Hinter Speicherklassen, die ihr DDL besitzen. Einfaches Schema; die PostgreSQL-Portierung als Aufgabe einer späteren Phase verfolgt, in Stunden gemessen.
Deterministische Logik (3 MCP-Server)	Hinter MCP-Werkzeugverträgen über FastMCP; auch direkt importierbare Module. Im Prozess für den Schnitt; der stdio-Subprozess-Transport durch den NAT-Workflow bewiesen — der Wechsel dazwischen ist Konfiguration.
Beobachtbarkeit (Phoenix)	Hinter einem OpenTelemetry-OTLP-Endpunkt; OpenInference-Auto-Instrumentierung. Einzelner Container; Endpunkt und Projektname sind Umgebungseinstellungen; das Tracing kann für Tests per Flag deaktiviert werden.
Orchestrierung (reiner Python-Schnitt)	Typisierte Agentenklassen mit run()-Methoden; NAT-Workflow-YAML daneben. Der Agentencode ist portabel; die Orchestrierung ins NAT-YAML zu heben ist eine spätere Verdrahtungsänderung, wobei der MCP-Smoke-Test-Workflow das Muster bereits demonstriert.

Jedes Modell und jeder Dienst hinter einer Standard-Naht. Das Versprechen der Architektur — per Konfiguration wechseln, nicht per Code — wurde während des Baus wiederholt ausgeübt, nicht bloß behauptet.

8. Das Hauptziel — das Vertrauensgeschirr in der Praxis

Alles Obige ist die Bühne; dies ist das Stück. Das zentrale Ziel des PoC war zu demonstrieren, dass Agenteninferenz und -verhalten End-to-End in einem Vertrauensgeschirr umschlossen werden können: eine Nachweis-, Provenienz- und Nachvollziehbarkeitskette für jede Behauptung, ein deterministisches Festlegen jeder folgenreichen Entscheidung und Beobachtbarkeit der gesamten Agententopologie. As-built ruht dieses Geschirr auf vier tragenden Invarianten, jede vom Code durchgesetzt:

1. Der Determinismus besitzt den Wert	Das LLM schlägt eine Stufe vor; engine.evaluate() legt sie fest. Die Engine ist konstruktionsbedingt der einzige

Punkt im ganzen System, an dem ein Wert FESTGELEGT wird — kein anderer Codepfad kann eine festgelegte Stufe schreiben. Das Modell baut einen ScoreProposal; die Engine gibt einen ScoreDecision zurück; die festgelegte Stufe auf dieser Entscheidung ist der einzige Wert, den das System anerkennt.

2. Jede Übergabe ist typisiert Kein freies Dictionary überquert eine Agenten- oder Werkzeuggrenze — nur Pydantic-validierte Modelle (RunRequest - RetrievalResult - EvidenceBundle - ScoreProposal - ScoreDecision - RunResult), auf der Empfängerseite validiert. Ungültige Elemente werden an der Grenze verworfen, nicht stromabwärts entdeckt.

3. Zitate sind verpflichtend und validiert Jedes Nachweiselement trägt eine erforderliche chunk_id. Der Nachweisextraktions-Agent filtert jede vom Modell ausgegebene ID gegen die Menge der in diesem Lauf tatsächlich abgerufenen Fragmente — ein halluziniertes Zitat kann den Filter nicht überleben, sodass ?jede Behauptung löst sich zu einer echten Quelle auf" eine durchgesetzte Eigenschaft ist, keine Hoffnung.

4. Die Topologie ist beobachtbar Agentengrenzen geben AGENT-Spans aus, deterministische Helfer geben TOOL-Spans aus, und OpenInference instrumentiert jeden Modellaufruf automatisch als LLM-Span, verschachtelt unter seinem aufrufenden Agenten. Phoenix zeigt den Lauf als Trace-Baum — die Struktur des Systems, kein flaches Protokoll — und eine Regression zeigt direkt auf den Span, der sie erzeugte.

8.1 Der Schlussstein — ein Modell schlägt vor, eine Engine legt fest

Der Übergang von ScoreProposal zu ScoreDecision ist der architektonische Schlussstein. Wenn der Bewertungs-Agent die Rubrik und die Nachweise gelesen hat, schlägt er die CMMI-Stufe vor, die die Nachweise belastbar stützen — der Prompt ist explizit, dass er konservativ sein soll — und übergibt den Vorschlag an die deterministische Engine. Die Engine wendet vier Regeln der Reihe nach an: Zitate verpflichtend; die Stufe im Bereich 1 bis 5 (ein Defense-in-Depth-Schema-Schutz); Nachweishinlänglichkeit skaliert zur beanspruchten Stufe; und referenzielle Integrität — jede zitierte ID muss existieren. Die Hinlänglichkeitsleiter ist explizit:

S1 — Initial ? 1 Nachweiselement, beliebige Polarität.

S2–S3 — Verwaltet / Definiert ? 2 Elemente, ? 1 positiv.

S4 — Quantitativ verwaltet ? 3 Elemente, ? 2 positiv, und ein Mess-/Metrik-Signal.

S5 — Optimierend ? 4 Elemente, ? 3 positiv, und Mess- und Kontinuierliche-Verbesserungs-Signale.

Ein Vorschlag, der scheitert, gibt ZURÜCKGESTELLT zurück — das System weigert sich festzulegen und bringt die Lücke zur menschlichen Prüfung, statt zu raten; eine Bereichsverletzung gibt ABGELEHNT zurück. Die Engine passt eine Stufe nie an: sie legt genau das Vorgeschlagene fest, oder lehnt ab.

8.2 Nachweiskette, Provenienz und die Audit-Rückverfolgung

Das Geschirr macht jeden festgelegten Wert durch Rückverfolgung prüfbar. Eine Entscheidung persistiert in der Tabelle score_decisions unter ihrer Lauf-ID; die Entscheidung trägt die Nachweis-Fragment-IDs; jedes Nachweiselement trägt sein Zitat, seine Behauptung und seine Polarität gegen eine erforderliche chunk_id; jedes Fragment trägt seine Dokument-ID, Ordnungszahl, Seite und Zeichen-Offsets; und jede Dokument-ID ist ein Inhalts-Hash des Quelltextes. Für jeden Wert, den das System festlegt, kann man daher rückwärts gehen — Entscheidung, zu Vorschlag, zu Nachweiselementen, zu Fragment-IDs, zu den exakten Zeichenbereichen der exakten Quelldokumente — und genau bestimmen, wo das Sprachmodell die Kette berührte und wo es das nicht konnte. Der Speicher ist geschichtet, um dies zu schützen: die Wissensbasis (Chroma plus die relationalen Referenztabellen) wird nur von Aufnahme und Seedern beschrieben, nie von Agenten; der Arbeitsspeicher sind die typisierten Übergabeobjekte innerhalb eines Laufs; der Lauf-Speicher ist die persistierte Entscheidungstabelle.

8.3 Inferenz und Verhalten mit Beobachtbarkeit umschließen

Beobachtbarkeit wurde entworfen, um das Multi-Agenten-System lesbar zu machen, nicht bloß protokolliert. Der Import des Konfigurationspakets des Projekts verdrahtet das Tracing genau einmal, sodass jeder Einstiegspunkt die Phoenix-Instrumentierung automatisch erhält. Der Supervisor gibt den Wurzel-AGENT-Span aus; jeder Worker verschachtelt sich darunter; jeder deterministische Helferaufruf — embed_query, vector_search, render_rubric, engine_evaluate, decisions-store-Upsert — ist ein TOOL-Span; jeder Modellaufruf wird automatisch mit vollem Prompt, Antwort, Token-Zählungen und Latenz getract. Zwei bewusste Verfeinerungen zeigen die Disziplin: Embedding-Aufrufe werden aus der Instrumentierung unterdrückt (deterministisch, hochvolumig und ein Rate-Limiting-Risiko bei Massenaufnahme), und die Span-Struktur ergibt sich konstruktionsbedingt aus dem Code — einen künftigen Agenten hinzuzufügen bedeutet, seinen run() in agent_span() zu umschließen, nichts weiter.

8.4 Der Beweis — der vertikale Schnitt der Phase 6

Das Erfolgskriterium des Schnitts war eine funktionierende Fähigkeit-×-Direktion-Kombination, End-to-End. Der Lauf bewertete die Fähigkeit INF.DQA (Datenqualität und -genauigkeit) für die Direktion 332000 (Wildlife Enforcement): das Retrieval auf die 112 dieser Direktion zugeordneten Fragmente gefiltert; der Nachweisextraktions-Agent erzeugte fünf zitierte Nachweiselemente, alle zu echten Fragmenten auflösend; der Bewertungs-Agent schlug vor; und die Engine LEGTE auf CMMI-Stufe 1 FEST, mit dem vollständigen, persistierten RunResult und der gesamten Topologie sichtbar als einzelner Phoenix-Trace-Baum.

Eine festgelegte Stufe 1 ist das korrekte Ergebnis für diese Eingabe, kein schwaches: die Nachweise stützen einen belastbaren Boden, und die Engine legte genau das fest, was die Regeln erlauben. Das System lehnte es ab, zu viel zu behaupten — was das Verhalten ist, das die ganze Architektur zu erzeugen gebaut ist, und die einzige wichtigste Zeile in diesem Dokument. Das Vertrauensgeschirr beschränkte nicht nur das Modell; es formte das Ergebnis nachweislich zur Belastbarkeit.

9. Das Abweichungsregister — Herausforderung, Lösung, architekt

Jede Abweichung von der konzeptionellen Architektur wurde festgehalten, während sie gemacht

wurde. Keine ist ein Rückschritt; jede war eine bewusste Antwort auf Hardware-, Werkzeug- oder Datenreibung. Konsolidiert (Bereich · ursprünglich - as built · warum):

Schlussfolgerungsmodell Llama-3.3-Nemotron-Super-49B - Qwen3-32B (über ein Llama-3-8B-Zwischenmodell). Das Nemotron-NIM ließ sich auf der DGX Spark nicht servieren; Llama 3 8B fehlten zuverlässige Werkzeugaufrufe. Kosten akzeptiert und einkalkuliert: ein Kontext von 8.192 Tokens.

Orchestrierung NAT-YAML-Workflow - reiner Python-Schnitt; NAT-YAML für MCP-Smoke-Tests beibehalten. Expliziter Kontrollfluss ist weit besser debuggbar als eine YAML-ReAct-Schleife; der Agentencode bleibt portabel.

Build-Werkzeuge pip - uv. Der kombinierte Abhängigkeitsgraph traf pips resolution-too-deep; uv löst ihn in Sekunden.

Python-Laufzeit Einzelnes System-Python - dual 3.10 + 3.12 mit /opt/venv. nvidia-nat braucht 3.11+; das Basis-Image liefert 3.10 und kann nicht gewechselt werden.

Org-Struktur-Quelle Reifegrad-Vorlage, gepunktete IDs (A.1) - programs.xlsx, sechsstellige Finanzzentren-Codes; Abteilung als Ebene hinzugefügt. Die reichere kanonische Hierarchie — die Codes sind die echten organisatorischen Kennungen.

MCP-Transport (Schnitt) stdio-Subprozess-Transport - direkter In-Prozess-Import; stdio-Pfad durch den NAT-Workflow ausgeübt. Ein Prozess, ein Stacktrace während der Schnitt-Entwicklung; beide Pfade führen denselben Modulcode aus.

Dienst-Hosting NIM und Phoenix als Workbench-Apps - host-seitige Docker-Container. Zweckmäßig für den Build; die Beförderung zum sauberen Lebenszyklus ist verfolgt, mit der genannten Neustart-Einschränkung.

Zwei Einträge (Orchestrierung und MCP-Transport) sind per Konfiguration umkehrbar — der Code wurde so geschrieben, dass der Wechsel zur NAT-Orchestrierung und zum MCP-Transport Verdrahtung ist, keine Neuschreibung. Daneben verfolgt ein Register der zurückgestellten Arbeit alles bewusst Aufgeschobene mit seiner Zielphase — das Qwen3-Denkmodus-Flag im LLM-Helfer, den Aufbau des Bewertungsprüfstands, die Kalibrierung der S4/S5-Schlüsselwort-Heuristiken gegen Golddaten, die Direktions-Kennzeichnungslücke des zweiten Korpus, die verbleibenden drei Agenten, die Verbreiterung über die einzige validierte Kombination hinaus, die ID-Schema-Aktualisierung des Report-Builders, die Verfeinerung des Kennzeichners, den MCP-Transport-Umzug, die Dienst-Lebenszyklus-Beförderung und die eventuelle SQLite-zu-PostgreSQL-Portierung. Nichts darin ist vergessen; alles darin fügt sich in die bestehende Struktur. Die wichtigste Eigenschaft des Endzustands der Reise ist, dass die Architektur sich nicht biegen musste, um ihn zu erreichen.

10. Was die Reise demonstriert

- Platzsparende KI-Hardware, praktisch konfiguriert. Eine DGX Spark (GB10 Grace Blackwell, 128 GB einheitlicher Speicher, aarch64, CUDA 13) vom Auspacken zu einem reproduzierbaren containerisierten Stack gebracht: NIM-Modell-Serving, ein lokaler Embedding-Dienst, Phoenix-Beobachtbarkeit, Vektor- und relationale Speicher und ein AI-Workbench-Entwicklungscontainer — einschließlich der plattformspezifischen Reibung (duale

Python-Laufzeiten, aarch64-GPU-Wheels, scharfe Kanten der Compose-Umgebung), die nur auftaucht, wenn man die Arbeit tatsächlich macht.

- Kleine und mittelgroße Sprachmodelle, mit einkalkulierten Beschränkungen umgesetzt. Ein 32-B-Schlussfolgerungsmodell und ein kompaktes Embedding-Modell, die gleichzeitig auf einer Workstation servieren, mit dem 8K-Kontextfenster als ingenieurmäßige Eingabe behandelt: dynamische Token-Budgetierung, auf das Fenster dimensioniertes Retrieval, auf die Sequenzgrenze des Embedders abgestimmtes Chunking und bewusst verwaltete Inferenzmodus-Flags.
- Anpassungsfähigkeit, belegt durch die Modell-Odyssee. Drei Modelle auf dem Gerät gegen eine präzise, architekturgetriebene Auswahlschwelle bewertet — zuverlässige Werkzeugaufrufe und strukturierte Ausgabe auf dieser Hardware — wobei jede Änderung null Agentencode-Modifikationen kostete, weil die Nähte dafür entworfen waren.
- Eine Plug-and-Play-Multi-Container-Architektur. Jedes Modell und jeder Dienst hinter einer Standardschnittstelle — OpenAI-kompatible APIs, MCP-Werkzeugverträge, Speicher-Wrapper, OTLP — mit der Wechseleigenschaft wiederholt während des Baus ausgeübt, statt auf einem Diagramm behauptet.
- Das Vertrauensgeschirr als Schwerpunkt. Agenteninferenz und -verhalten End-to-End umschlossen: typisierte Verträge an jeder Grenze, verpflichtende und validierte Zitate, eine deterministische Engine als einziger Schreiber festgelegter Entscheidungen, eine persistente Nachweiskette, die eine Rückverfolgung auf Zeichenebene stützt, geschichteter Speicher, den Agenten nicht korrumpieren können, und ein Trace-Baum, der die gesamte Agententopologie offenlegt. Bewiesen durch einen Lauf, dessen aussagekräftigstes Ergebnis Zurückhaltung war: das System legte die Stufe fest, die die Nachweise belastbar stützten, und nicht mehr.

Zusammengenommen ist die Reise der Beleg eines Teams, das rigoros plant, Reibung ehrlich begegnet, sie ohne Gefährdung der Vertrauensgarantien löst und jede Abweichung festhält, sodass Plan und System nie still auseinanderdriften — dieselbe Disziplin, zu der sich unsere künftigen Vorschläge verpflichten.

11. Glossar

Agentische KI Ein KI-System, in dem sprachmodellgesteuerte Komponenten (Agenten) planen, Werkzeuge aufrufen und einander Arbeit übergeben, um eine Aufgabe zu erledigen, statt einen einzigen Prompt zu beantworten.

Agent- / Tool- / LLM-Span Die drei Span-Arten in den Traces dieses Systems: einer je Agentengrenze, einer je deterministischem Helferaufruf und einer (automatisch ausgegeben) je Modellaufruf, verschachtelt zu einem einzigen Trace-Baum je Lauf.

AI Workbench (NVIDIA) NVIDIAs verwaltete Entwicklungsumgebung, die den Entwicklungscontainer und die Compose-Dienste des Projekts baut und ausführt.

bge-large-en-v1.5 Das kompakte offene Embedding-Modell (1.024-dimensionale Vektoren, 512-Token-Sequenzgrenze), lokal für die gesamte Vektorgenerierung serviert.

Nachweiskette Die ununterbrochene, persistierte Verknüpfung von einem festgelegten Wert zurück über seinen Vorschlag und die zitierten Nachweiselemente zu exakten Zeichenbereichen der Quelldokumente.

Chroma Die Vektordatenbank, die die eingebetteten Dokumentfragmente hält; für den PoC eingebettet im Prozess ausgeführt, per Fragment-ID indexiert, sodass sie sauber mit dem relationalen Speicher verbindet.

Fragment (Chunk) Ein zusammenhängender Bereich eines Quelldokuments — mit Dokument-ID, Ordnungszahl, Zeichen-Offsets und organisatorischen Metadaten — der eingebettet, abgerufen und zitiert wird.

CMMI Capability Maturity Model Integration: die fünfstufige Reifeskala (Initial, Verwaltet, Definiert, Quantitativ verwaltet, Optimierend), die zur Bewertung genutzt wird.

DGX Spark NVIDIAs KI-Workstation der Desktop-Klasse, auf dem GB10-Grace-Blackwell-Superchip mit 128 GB einheitlichem CPU/GPU-Speicher gebaut — die platzsparende Hardware, auf der der gesamte PoC läuft.

Direktion Die organisatorische Bewertungseinheit von ECCC, identifiziert durch einen sechsstelligen Finanzzentren-Code (z. B. 332000).

Embedding Eine numerische Vektordarstellung von Text für die semantische Ähnlichkeitssuche; hier von einem dedizierten kleinen Modell erzeugt, nie vom Schlussfolgerungsmodell.

FastMCP Das leichtgewichtige Python-Rahmenwerk, das jeden MCP-Server bereitstellt; jeder Server ist ein dünner Wrapper über einem importierbaren Modul.

Guardrails Code-durchgesetzte Beschränkungen von Agenteneingaben, -ausgaben und Werkzeugnutzung — im Unterschied zu Anweisungen in Prompts.

Idempotent Dasselbe Ergebnis erzeugen, wenn gegen unveränderte Eingaben erneut ausgeführt — die Eigenschaft, die Aufnahme und Seeding reproduzierbar macht.

MCP (Model Context Protocol) Ein offener Standard, um Werkzeuge und Daten KI-Systemen über typisierte Werkzeugverträge bereitzustellen; der PoC baute drei eigene MCP-Server.

NAT (NeMo Agent Toolkit) NVIDIAs agentisches Rahmenwerk zum Komponieren von Agenten, Werkzeugen und Workflows; hier als MCP-Client für die Transport-Smoke-Tests genutzt, mit dem in reinem Python orchestrierten Schnitt.

NIM (NVIDIA Inference Microservice) Ein containerisierter, optimierter Modell-Serving-Dienst, der eine OpenAI-kompatible API bereitstellt; wie das Schlussfolgerungsmodell serviert wird.

OpenAI-kompatible API Die De-facto-Standard-HTTP-Schnittstelle für Chat/Embeddings; jedes Modell im System sitzt hinter einer, was Modelle per Konfiguration austauschbar macht.

OpenInference / OpenTelemetry (OTLP) Die Instrumentierungskonvention und das Telemetrieprotokoll, mit denen jeder Modellaufruf und Span erfasst und an Phoenix gesendet wird.

Phoenix (Arize Phoenix) Das Open-Source-Beobachtbarkeits-Backend, das die Trace-Bäume speichert und visualisiert; läuft als einzelner lokaler Container, sodass alle Trace-Daten auf dem Gerät bleiben.

Provenienz Der erfasste Ursprung jedes Datums und jeder Inferenz — welches Dokument, welches Fragment, welcher Lauf, welcher Modellaufruf es erzeugte.

Pydantic Die Python-Validierungsbibliothek, die für jedes Schema im System genutzt wird; der

Mechanismus hinter ?jede Übergabe ist typisiert".

Qwen3-32B Das offene Schlussfolgerungsmodell mit 32 Milliarden Parametern, nach Bewertung auf dem Gerät ausgewählt; über NIM mit einem Kontextfenster von 8.192 Tokens serviert.

RAG (Retrieval-Augmented Generation) Einem Modell abgerufenes Quellmaterial liefern, um seine Ausgabe zu fundieren; hier ist das Retrieval deterministisch und nach Direktion gefiltert.

ScoreProposal / ScoreDecision Das Schlussstein-Paar: die vom Modell vorgeschlagene Stufe mit Begründung und Nachweisen und das festgelegte/zurückgestellte/abgelehnte Urteil der deterministischen Engine — der einzige Wert, den das System anerkennt.

SLM (kleines Sprachmodell) Ein kompaktes Sprachmodell, das auf beschränkter Hardware einsetzbar ist; in diesem PoC das Embedding-Modell und, relativ zu Spitzenmodellen, das 32-B-Schlussfolgerungsmodell.

Supervisor/Worker-Muster Die Orchestrierungsform: ein Supervisor validiert Eingaben und steuert spezialisierte Worker-Agenten in einer festen, nachvollziehbaren Abfolge.

Trace-Baum Die verschachtelte Span-Struktur eines Laufs in Phoenix — die sichtbar gemachte Agententopologie, wobei eine Regression auf den Span zeigt, der sie erzeugte.

Vertrauensgeschirr Die kombinierte Menge durchgesetzter Mechanismen — typisierte Verträge, validierte Zitate, deterministisches Festlegen, geschichteter Speicher und Beobachtbarkeit — die Agenteninferenz und -verhalten umschließt.

uv Der schnelle Python-Paket-Resolver/-Installer, der übernommen wurde, nachdem pip den Abhängigkeitsgraphen des Projekts nicht auflösen konnte.

Vektorspeicher / Vektorsuche Eine Datenbank von Embeddings, per semantischer Ähnlichkeit abgefragt; wie die Nachweis-Kandidatenfragmente gefunden werden.

Vertikaler Schnitt Ein minimaler, aber vollständiger End-to-End-Pfad durch jede Schicht des Systems — das Ergebnis und der Beweis der Phase 6.

ENDE DES DOKUMENTS