ENKIDU

MIOVSKI GROUP

undefined

1. Resumen ejecutivo

Entre la aprobación de la arquitectura conceptual y el cierre de la rebanada vertical de la fase 6, nuestro equipo diseñó, configuró, construyó y probó un sistema de IA multiagente de extremo a extremo en un solo puesto de trabajo de bajo consumo de espacio — el NVIDIA DGX Spark — para la prueba de concepto de evaluación de la madurez de capacidades de negocio de Environment and Climate Change Canada. El sistema lee la propia documentación de una organización y produce puntajes de madurez CMMI deterministas y respaldados por citas, con cada puntaje portando toda su derivación: de la decisión comprometida, de vuelta por la propuesta del modelo, a los elementos de evidencia citados, a los tramos de caracteres exactos de los documentos fuente exactos.

Este documento cuenta honestamente la historia de la implementación: qué se planeó, qué encontramos cuando el plan se enfrentó al hardware real, al comportamiento real del modelo y a los datos reales, cómo se resolvió cada desafío y cómo esas resoluciones cambiaron la arquitectura. Ninguno de los cambios fue una regresión; cada uno fue una respuesta de ingeniería deliberada, registrada en su momento en un registro de desviaciones tal como se construyó. El resultado es un sistema funcional cuya premisa central — determinismo donde importa, razonamiento donde ayuda — sobrevivió intacta a cada cambio, porque se impuso estructuralmente en lugar de pidiéndoselo amablemente al modelo.

El recorrido demuestra cinco cosas en la práctica: (1) la configuración práctica de hardware de IA de bajo consumo de espacio y toda su pila de servicio, orquestación y observabilidad; (2) adaptabilidad en la selección del modelo — tres modelos de razonamiento evaluados en el dispositivo hasta que uno alcanzó el umbral de llamadas de herramientas fiables y salida estructurada dentro de la envolvente de hardware; (3) experimentación disciplinada con modelos de lenguaje pequeños y medianos, incluidas las consecuencias concretas de ingeniería de una ventana de contexto de 8 K tokens; (4) una arquitectura contenedorizada y enchufable en la que cada modelo y servicio se sitúa tras una interfaz estándar y puede intercambiarse por configuración, no por código; y (5) — el objetivo principal de la PdC — un arnés de confianza que envuelve cada inferencia y comportamiento de agente con contratos tipados, citas validadas obligatorias, un motor de puntuación determinista, una cadena de evidencia persistente y observabilidad de extremo a extremo de toda la topología de agentes.

2. Propósito y guía de lectura

Este Recorrido de implementación acompaña a dos documentos fuente: el documento de arquitectura de la PdC de IA agéntica original, que fijó la intención (una topología de siete agentes, puntuación determinista, integración de herramientas basada en MCP), y el expediente de arquitectura tal como se construyó (Rev. A), que concilia esa intención con el código en disco al cierre de la fase 6. Donde esos documentos describen el sistema, este describe el camino: la secuencia de fases de construcción, la fricción encontrada, las decisiones tomadas y las consecuencias arquitectónicas de cada una.

Está escrito como una referencia de experiencia demostrada. Cada desafío y resolución descritos aquí provienen del registro tal como se construyó y del repositorio del proyecto — no reconstruidos a posteriori. Las desviaciones de la arquitectura original se presentan deliberada y completamente, porque la capacidad de encontrar fricción en hardware restringido, diagnosticarla y resolverla sin

comprometer las garantías de confianza es precisamente la experiencia que este documento evidencia.

Nota de estado para el revisor: este borrador describe la construcción hasta la rebanada vertical de la fase 6 y el trabajo de la fase 7 en curso, según la línea base Rev. A tal como se construyó. Si fases posteriores (banco de evaluación, agentes restantes, ampliación del modelo completo) deben reflejarse en la versión final, ese estado puede añadirse a las secciones 8 y 9 según su indicación.

3. El punto de partida — lo que planeó la arquitectura conceptual

La arquitectura conceptual definió el problema — evaluar la madurez de las capacidades de negocio a partir de un corpus heterogéneo de transcripciones de entrevistas, documentos de proceso, informes de incidentes y hallazgos de auditoría — y una premisa de diseño que sostiene todo el sistema: determinismo donde importa, razonamiento donde ayuda. Las fórmulas de puntuación, las rúbricas, el modelo de capacidades y la estructura organizativa son verdad de terreno impuesta por el código; el modelo de lenguaje interpreta, extrae y narra, pero nunca realiza aritmética ni asigna un puntaje final. Todo se ejecuta en infraestructura local: los documentos fuente contienen información organizativa sensible, así que nada se transmite a una API externa.

TABLA 1 · LA PILA PLANEADA (CAPA · SELECCIÓN · JUSTIFICACIÓN)

Hardware	NVIDIA DGX Spark (GB10 Grace Blackwell, 128 GB de memoria unificada CPU/GPU). Soberanía de datos en un solo puesto de trabajo local; capacidad para un modelo de ~49 B con margen para embeddings, vectores y observabilidad; pila NVIDIA preintegrada.
Modelo de razonamiento	Llama-3.3-Nemotron-Super-49B-v1.5 vía NIM. Entrenado para llamadas de herramientas estructuradas; razonamiento sólido; desplegable en el dispositivo; una API compatible con OpenAI mantiene el motor de ejecución intercambiable.
Marco agéntico	NVIDIA NeMo Agent Toolkit (NAT). Composición agnóstica al marco, cliente MCP nativo, ganchos de evaluación y trazado integrados.
Memoria vectorial	Chroma. Embebido/ligero; dimensionado para miles de fragmentos; filtrado rico por metadatos para restricciones de capacidad/tipo/peso.
Observabilidad	Arize Phoenix (OpenTelemetry / OpenInference). Código abierto, un solo contenedor, todas las trazas permanecen locales; evaluadores de fundamentación específicos de RAG.
Integración de herramientas	Servidores Model Context Protocol (MCP), incl. servidores personalizados de Modelo de Capacidades, Rúbrica/Puntuación y Generador de Informes. Lógica determinista en el código tras firmas de herramientas estrechas y tipadas — nunca en los prompts.
Patrón de orquestación	Supervisor/trabajador con siete agentes (Supervisor, Recuperación, Extracción de Evidencia, Puntuación, Crítica, Agregador, Narrativa). Predecible, de costo acotado, trazable; separación de tareas entre extracción, puntuación y narrativa.
Almacén relacional	SQLite (escalable a PostgreSQL). El modelo de capacidades, las rúbricas, el historial de ejecuciones y los puntajes son conocimiento relacional, distinto de la memoria vectorial.

Casi todo sobrevivió al contacto con la realidad; donde no, las secciones 4 a 7 explican exactamente qué cambió y por qué.

4. Levantar la plataforma — hardware, contenedores y herramienta

El primer tramo del recorrido fue pura ingeniería de plataforma: convertir un DGX Spark nuevo en un entorno de desarrollo y servicio contenedorizado y reproducible. Tres desafíos llegaron de inmediato, y cada resolución se convirtió en una parte permanente y documentada de la construcción.

4.1 El entorno tal como se diseñó

El proyecto se ejecuta como un proyecto de NVIDIA AI Workbench: Workbench gestiona un contenedor de desarrollo que monta el repositorio, y una pila Compose ejecuta los servicios de servicio de modelo y observabilidad junto a él. Dentro del contenedor, los servicios se resuelven por nombre: el punto final de razonamiento NIM en el puerto 8000, un servicio de embedding local en el puerto 8002, Phoenix en 6006 (IU) con ingesta OTLP en 4317/4318, y Chroma y SQLite ejecutándose en proceso contra el estado en disco. Los 128 GB de memoria unificada CPU/GPU del DGX Spark contienen simultáneamente el modelo de razonamiento, el modelo de embedding, el almacén vectorial y la pila de observabilidad, con margen — y como el hardware es dedicado, la latencia de inferencia es predecible, lo que importa para la orquestación multiagente.

4.2 Desafío — un contenedor de doble Python que la plataforma no anticipó

Desafío La imagen base de Workbench trae Python 3.10 del sistema, pero el NeMo Agent Toolkit requiere Python 3.11+, y la IU de Workbench no permite cambiar la imagen base. Mientras tanto, JupyterLab y los propios paquetes de configuración de Workbench dependen de que el Python del sistema permanezca exactamente donde está.

Resolución La construcción del contenedor instala Python 3.12 junto a 3.10 y enruta todo el código del proyecto a través de un entorno virtual en /opt/venv, dejando el Python del sistema intacto para las herramientas de la plataforma. La disposición está guionizada en el paso posterior a la construcción del contenedor, así que es reproducible en cualquier reconstrucción en lugar de un estado de máquina hecho a mano.

Impacto arquitectónico Un diseño de contenedor de doble ejecución deliberado, codificado en el script de construcción — y un patrón reutilizado dondequiera que los valores predeterminados de la plataforma y los requisitos de la cadena de herramientas diverjan: satisfacer ambos, en código, de forma reproducible.

4.3 Desafío — el grafo de dependencias que venció a pip

Desafío El grafo de dependencias combinado — Phoenix, LangChain, ChromaDB, OpenTelemetry, MCP, sentence-transformers y nvidia-nat juntos — venció al resolvedor con retroceso de pip, que falló con resolution-too-deep tras unos ocho minutos. En una plataforma aarch64 había una segunda trampa: instalar

sentence-transformers ingenuamente tira la rueda PyTorch solo CPU de PyPI, descartando silenciosamente la GPU.

Resolución La construcción arranca uv, que resuelve el mismo grafo en segundos, y enruta cada instalación de dependencia del proyecto por él. PyTorch se instala primero, explícitamente desde el índice de ruedas CUDA-13 aarch64, para que sentence-transformers tome la rueda GPU en lugar de la CPU por defecto.

Impacto arquitectónico Las herramientas de construcción cambiaron de pip a uv — registrado como desviación ?3 en el registro tal como se construyó — y el orden de instalación se convirtió en un hecho arquitectónico del script de construcción, no en conocimiento tribal.

4.4 Desafío — ciclo de vida de contenedores y el borde afilado del bind-mount

Desafío La arquitectura original quería NIM y Phoenix gestionados como aplicaciones de Workbench, compartiendo el espacio de nombres de red y el ciclo de vida del proyecto. Afinar bien la pila Compose reveló un borde afilado: Workbench Compose no hereda el entorno HOME del usuario, así que las rutas de bind-mount relativas o basadas en variables se expanden silenciosamente a cadenas vacías.

Resolución Ambos servicios se ejecutan como una pila Compose documentada con rutas de bind-mount absolutas, reservas de GPU, dimensionamiento de memoria compartida, comprobaciones de salud (incluido un período de arranque generoso para la primera carga del modelo NIM) y políticas de reinicio. Para la PdC se ejecutan como contenedores del lado del host; promoverlos a aplicaciones de Workbench gestionadas es una tarea de manual de operador seguida y conscientemente diferida — la disposición actual funciona pero no sobrevive limpiamente a los reinicios del host, y decirlo con claridad es parte de la disciplina del tal-como-se-construyó.

Impacto arquitectónico Infraestructura como código para toda la pila de servicio y observabilidad, con su limitación conocida registrada en el registro de trabajo diferido en lugar de descubierta por el siguiente operador.

5. El recorrido de los modelos — adaptabilidad bajo una envolvente

El tramo más instructivo del recorrido fue la selección del modelo de razonamiento. El plan nombró un modelo; la construcción evaluó tres en el dispositivo antes de comprometerse. La secuencia, y lo que enseñó cada paso, es la demostración más clara de este documento de adaptar modelos para obtener el mejor resultado en hardware restringido.

5.1 Primer intento — Llama-3.3-Nemotron-Super-49B-v1.5

La elección de la arquitectura. En la práctica, tirar y servir el contenedor NIM de Nemotron falló en el DGX Spark — el contenedor empaquetado no se servía en esta plataforma. En lugar de luchar contra el empaquetado, el equipo trató el modelo como lo que la arquitectura siempre dijo que era: un componente reemplazable tras una API compatible con OpenAI.

5.2 Segundo intento — Llama 3 8B, y el umbral de llamada de herramientas

Un NIM Llama 3 8B más pequeño se servía correctamente — prueba de que la ruta de servicio funcionaba — pero no soportaba la llamada de herramientas/funciones estilo OpenAI de forma suficientemente fiable. Ese es un fallo descalificante en esta arquitectura, porque cada agente depende de llamadas de herramientas estructuradas y salida JSON válida según el esquema. El experimento estableció el umbral de selección con precisión: no los puntajes de banco de pruebas, sino un comportamiento fiable de salida estructurada y llamada de herramientas en este hardware, en esta pila de servicio, para estos agentes.

5.3 El aterrizaje — Qwen3-32B, e ingeniería a su envolvente

Qwen3-32B vía NIM alcanzó el umbral: comportamiento sólido de llamada de herramientas y salida estructurada con un ajuste cómodo de GPU dentro de la memoria unificada del Spark. Vino con una verdadera concesión — una longitud de contexto máxima de 8.192 tokens, muy por debajo de los 32 K+ del Nemotron — y el equipo hizo ingeniería a esa envolvente en lugar de fingir que no existía:

- Presupuesto dinámico de tokens. El asistente LLM compartido calcula max_tokens dinámicamente frente al presupuesto de contexto restante en cada llamada, de modo que un prompt largo nunca puede truncar silenciosamente la respuesta.
- Recuperación dimensionada a la ventana. La recuperación extrae 5 a 10 fragmentos por consulta en lugar de los 20+ que tomaría un modelo de contexto 32 K — una decisión deliberada de densidad de calidad, no una limitación descubierta en producción.
- Disciplina del modo pensamiento. Qwen3 viene con el «modo pensamiento» activado por defecto; para tareas de salida estructurada quema tokens internamente y multiplica la latencia sin beneficio de calidad. La configuración del flujo NAT lo desactiva vía chat_template_kwargs; aplicar el mismo indicador en el asistente LLM de Python puro es un elemento de pre-vuelo seguido — un ejemplo del registro de trabajo diferido atrapando una pequeña inconsistencia antes de que se convierta en un misterio.

La lección estratégica que prueba el recorrido: porque cada agente habla con el modelo a través del cliente OpenAI estándar contra un punto final NIM, tres cambios de modelo costaron cero cambios de código de agente. La selección del modelo fue configuración más medición — exactamente la propiedad enchufable que prometió la arquitectura.

5.4 El segundo modelo — embeddings como un servicio de modelo pequeño separado

La decisión de apoyo de la arquitectura — nunca usar el modelo de razonamiento para embeddings — se implementó como su propio servicio de modelo pequeño: BAAI/bge-large-en-v1.5 (vectores de dimensión 1.024) tras un adaptador FastAPI compacto compatible con OpenAI en el puerto 8002. Los valores predeterminados de fragmentación (1.500 caracteres objetivo con 200 de solape) se ajustaron al límite de secuencia de 512 tokens del embedder. Como con el modelo de razonamiento, cambiar el modelo de embedding es un cambio de variable de entorno. Dos modelos, dos cargas de trabajo, dos servicios bien dimensionados — la disciplina de los modelos de lenguaje pequeños en miniatura.

6. Construir la base de evidencia — el sustrato determinista

Antes de que se ejecutara cualquier agente, la construcción levantó el sustrato determinista: el flujo de siembra de datos de referencia y el pipeline de ingesta de documentos. Ambos son

procesamiento de datos puro — sin llamadas de modelo, sin orquestación de agentes — y ambos son idempotentes por diseño: reejecutar cualquiera contra entradas sin cambios no produce filas nuevas. Esto es lo que garantiza que los agentes razonen sobre una base de evidencia estable y reproducible, y es la primera capa de la cadena de evidencia: cada fragmento porta un id de documento derivado del contenido, un ordinal, desplazamientos de caracteres y metadatos organizativos, de modo que una cita se resuelve a un tramo exacto de una fuente exacta.

6.1 Desafío — el modelo organizativo cambió bajo la construcción

Desafío Las primeras fases sembraron la estructura organizativa a partir de una hoja de cálculo de plantilla de madurez usando identificadores punteados (A.1, A.2). A mitad de la fase 6, la fuente de verdad canónica cambió a programs.xlsx — la verdadera jerarquía de centros financieros, con códigos de seis dígitos, 15 ramas, 94 direcciones y 344 divisiones, dispuesta en un formato Excel jerárquico disperso donde las celdas heredan de la fila superior.

Resolución Un cargador dedicado analiza la disposición dispersa con relleno hacia adelante y deduplicación en la primera ocurrencia (una dirección puede aparecer legítimamente bajo varios contextos de programa); el esquema ganó División como nivel de primera clase y columnas de contexto de programa desnormalizadas para que un prompt pueda decir «esta dirección trabaja en X bajo Y» sin una unión. La siembra se divide para que las tablas organizativas puedan reiniciarse y resembrarse sin tocar nunca la taxonomía de capacidades, de fuente independiente.

Impacto arquitectónico La unidad de evaluación se convirtió en el verdadero esquema de identificadores organizativos. Una consecuencia se difirió conscientemente: el Generador de Informes, escrito contra los ids punteados, necesita actualizarse al esquema de seis dígitos antes de que se cablee la generación de informes — seguido en el registro con el acoplamiento exacto documentado, porque la rebanada de la fase 6 termina en una decisión comprometida y nunca lo ejercita.

6.2 Desafío — etiquetar fragmentos a la estructura organizativa, y calidad de datos honesta

Desafío La puntuación por dirección requiere saber a qué dirección pertenece cada fragmento — pero los documentos fuente no anuncian su estructura organizativa de forma uniforme. Un documento de entrevista tenía encabezados de sección limpios por nombre de dirección; otro usaba encabezados por nombre de entrevistado y nunca nombraba direcciones.

Resolución Se insertó una nueva etapa de etiquetado en el pipeline de ingesta — entre carga y fragmentación — que segmenta el texto en secciones por coincidencias de nombres canónicos de la estructura organizativa (usando las propias herramientas de búsqueda de nombres del Modelo de Capacidades), fragmenta cada sección por separado y estampa cada fragmento con la dirección, rama y división de su sección, con un repliegue a nivel de rama donde ninguna dirección coincide. El pipeline tiene siete etapas tal como se construyó: descubrir - cargar - documento - etiquetar - fragmentar - embeber - persistir. La ingesta de los dos corpus de entrevistas produjo 685 fragmentos; 112 se etiquetaron limpiamente a la dirección objetivo — suficiente

para ejecutar y probar la rebanada — y el resultado de cero direcciones del segundo documento se registró como una brecha de calidad de datos nombrada con dos correcciones candidatas (una pre-pasada de inserción de encabezados, o un mapa de anulación entrevistado-dirección curado), programada para una fase posterior.

Impacto arquitectónico El pipeline ganó una etapa; los identificadores de fragmentos reflejan deliberadamente la estructura del documento bajo el modelo organizativo actual; y una limitación de calidad de datos se midió, cuantificó y siguió en lugar de disimularse — la misma disciplina de honestidad que el motor de puntuación aplica a la evidencia.

7. La capa lógica MCP — enchufable por construcción

Tres servidores Model Context Protocol personalizados sostienen la verdad de terreno impuesta por el código: el MCP Modelo de Capacidades (búsquedas puras sobre la taxonomía y la estructura organizativa — sin LLM, sin mutación por los agentes), el MCP Rúbrica/Puntuación (la rúbrica CMMI proyectada sobre una capacidad, más el motor de puntuación determinista) y el MCP Generador de Informes (ensamblaje de entregables a partir de decisiones comprometidas). Cada servidor es una envoltura fina FastMCP sobre un módulo importable — una elección de diseño que rindió directamente cuando llegó la fricción de transporte.

7.1 Desafío — depurar a través de los límites de proceso

Desafío El diseño original hace que los agentes alcancen las herramientas MCP por el transporte de subproceso stdio. Durante el desarrollo de la rebanada, la depuración multiproceso — varios servidores lanzados, fallos entrelazados — ralentizó el diagnóstico exactamente cuando la iteración rápida más importaba.

Resolución Ambas rutas se conservaron. La rebanada de agentes de la fase 6 importa directamente los módulos subyacentes de los servidores, en proceso — un proceso, una traza de pila — mientras que una definición de flujo NAT lanza los tres servidores como subprocesos stdio exactamente como se diseñó y apunta un agente ReAct al conjunto completo de herramientas contra el NIM Qwen3-32B local. La ruta de transporte está por tanto probada, ejercitable de forma continua, y mover los agentes a ella es un cambio de cableado, no una reescritura, porque ambas rutas ejecutan el mismo código de módulo.

Impacto arquitectónico Registrado como una desviación reversible por configuración. Los contratos MCP — no el transporte — son la arquitectura.

7.2 El inventario enchufable

Al cierre de la rebanada, cada componente sustituible se situaba tras una costura estándar. Este es el significado concreto de «múltiples contenedores y modelos que hacen la arquitectura enchufable»:

Modelo de razonamiento (Qwen3-32B) Tras una API de chat compatible con OpenAI servida por un contenedor NIM. Se intercambia solo por configuración — probado tres veces durante la selección (Nemotron - Llama 3 8B - Qwen3-32B) con cero cambios de código de agente.

Modelo de embedding (bge-large-en-v1.5)	Tras una API de embeddings compatible con OpenAI tras un contenedor adaptador FastAPI. Se intercambia por variables de entorno (punto final, modelo, dimensión).
Almacén vectorial (Chroma)	Tras una envoltura fina de almacén propiedad del proyecto. PersistentClient embebido hoy; se intercambia a modo servicio HttpClient sin cambio para los llamadores.
Almacén relacional (SQLite)	Tras clases de almacén que poseen su DDL. Esquema simple; el porte a PostgreSQL seguido como tarea de fase posterior medida en horas.
Lógica determinista (3 servidores MCP)	Tras contratos de herramientas MCP sobre FastMCP; también módulos directamente importables. En proceso para la rebanada; el transporte de subproceso stdio probado por el flujo NAT — moverse entre ellos es configuración.
Observabilidad (Phoenix)	Tras un punto final OTLP de OpenTelemetry; autoinstrumentación OpenInference. Un solo contenedor; el punto final y el nombre del proyecto son ajustes de entorno; el trazado puede desactivarse por indicador para las pruebas.
Orquestación (rebanada Python puro)	Clases de agente tipadas con métodos run(); YAML de flujo NAT al lado. El código de agente es portable; elevar la orquestación al YAML NAT es un cambio de cableado posterior, con el flujo de prueba de humo MCP ya demostrando el patrón.

Cada modelo y servicio tras una costura estándar. La promesa de la arquitectura — intercambiar por configuración, no por código — se ejercitó repetidamente durante la construcción, no meramente afirmada.

8. El objetivo principal — el arnés de confianza en la práctica

Todo lo anterior es el escenario; esto es la obra. El objetivo central de la PdC era demostrar que la inferencia y el comportamiento de los agentes pueden envolverse de extremo a extremo en un arnés de confianza: una cadena de evidencia, procedencia y trazabilidad para cada afirmación, un compromiso determinista de cada decisión consecuente, y observabilidad de toda la topología de agentes. Tal como se construyó, ese arnés descansa en cuatro invariantes portantes, cada uno impuesto por el código:

1. El determinismo posee el puntaje	El LLM propone un nivel; engine.evaluate() lo compromete. El motor es, por construcción, el único punto de todo el sistema en el que un puntaje se vuelve COMPROMETIDO — ninguna otra ruta de código puede escribir un nivel comprometido. El modelo construye un ScoreProposal; el motor devuelve un ScoreDecision; el nivel comprometido en esa decisión es el único valor que el sistema reconoce.
2. Cada traspaso es tipado	Ningún diccionario libre cruza un límite de agente o herramienta — solo modelos validados por Pydantic (RunRequest - RetrievalResult -

EvidenceBundle - ScoreProposal - ScoreDecision - RunResult), validados en el lado receptor. Los elementos inválidos se descartan en el límite, no se descubren aguas abajo.

3. Las citas son obligatorias y validadas Cada elemento de evidencia porta un chunk_id requerido. El agente de Extracción de Evidencia filtra cada id que emite el modelo contra el conjunto de fragmentos realmente recuperados en esta ejecución — una cita alucinada no puede sobrevivir al filtro, así que «cada afirmación se resuelve a una fuente real» es una propiedad impuesta, no una esperanza.

4. La topología es observable Los límites de agente emiten spans AGENT, los asistentes deterministas emiten spans TOOL, y OpenInference autoinstrumenta cada llamada de modelo como un span LLM anidado bajo su agente llamador. Phoenix muestra la ejecución como un árbol de trazas — la estructura del sistema, no un registro plano — y una regresión apunta directo al span que la produjo.

8.1 La clave de bóveda — un modelo propone, un motor compromete

La transición de ScoreProposal a ScoreDecision es la clave de bóveda arquitectónica. Cuando el agente de Puntuación ha leído la rúbrica y la evidencia, propone el nivel CMMI que la evidencia sostendrá de forma defendible — el prompt es explícito en que debe ser conservador — y envía la propuesta al motor determinista. El motor aplica cuatro reglas en orden: citas obligatorias; el nivel dentro del rango 1 a 5 (una guarda de esquema en defensa en profundidad); suficiencia de evidencia escalada al nivel reclamado; e integridad referencial — cada id citado debe existir. La escalera de suficiencia es explícita:

N1 — Inicial ? 1 elemento de evidencia, cualquier polaridad.

N2–N3 — Gestionado / Definido ? 2 elementos, ? 1 positivo.

N4 — Gestionado cuantitativamente ? 3 elementos, ? 2 positivos, y una señal de medición/métricas.

N5 — En optimización ? 4 elementos, ? 3 positivos, y señales de medición y mejora continua.

Una propuesta que falla devuelve DIFERIDA — el sistema se niega a comprometer y hace aflorar la brecha para revisión humana en lugar de adivinar; una violación de rango devuelve RECHAZADA. El motor nunca ajusta un nivel: compromete exactamente lo que se propuso, o declina.

8.2 Cadena de evidencia, procedencia y el recorrido de auditoría hacia atrás

El arnés hace cada puntaje comprometido auditable mediante recorrido hacia atrás. Una decisión persiste en la tabla score_decisions bajo su id de ejecución; la decisión porta los ids de fragmento de evidencia; cada elemento de evidencia porta su cita, aserción y polaridad contra un chunk_id requerido; cada fragmento porta su id de documento, ordinal, página y desplazamientos de caracteres; y cada id de documento es un hash de contenido del texto fuente. Para cualquier puntaje que el sistema comprometa, puede por tanto recorrer hacia atrás — decisión, a propuesta, a elementos de evidencia, a ids de fragmento, a los tramos de caracteres exactos de los documentos

fuente exactos — e identificar con precisión dónde tocó el modelo de lenguaje la cadena y dónde no pudo. La memoria está en capas para proteger esto: la base de conocimiento (Chroma más las tablas de referencia relacionales) solo la escriben la ingesta y los sembradores, nunca los agentes; la memoria de trabajo son los objetos de traspaso tipados dentro de una ejecución; la memoria de ejecución es la tabla de decisiones persistida.

8.3 Envolver la inferencia y el comportamiento con observabilidad

La observabilidad se diseñó para hacer el sistema multiagente legible, no meramente registrado. Importar el paquete de configuración del proyecto cablea el trazado exactamente una vez, de modo que cualquier punto de entrada obtiene la instrumentación Phoenix automáticamente. El Supervisor emite el span AGENT raíz; cada trabajador se anida debajo; cada llamada de asistente determinista — embed_query, vector_search, render_rubric, engine_evaluate, upsert del almacén de decisiones — es un span TOOL; cada llamada de modelo se traza automáticamente con el prompt completo, la respuesta, los conteos de tokens y la latencia. Dos refinamientos deliberados muestran la disciplina: las llamadas de embedding se suprimen de la instrumentación (deterministas, de alto volumen, y un riesgo de limitación de tasa durante la ingesta en masa), y la estructura de spans surge del código por construcción — añadir un agente futuro significa envolver su run() en agent_span(), nada más.

8.4 La prueba — la rebanada vertical de la fase 6

El criterio de éxito de la rebanada era una combinación capacidad × dirección funcional, de extremo a extremo. La ejecución evaluó la capacidad INF.DQA (Calidad y Exactitud de Datos) para la dirección 332000 (Aplicación de la Ley de Vida Silvestre): la recuperación filtrada a los 112 fragmentos etiquetados a esa dirección; el agente de Extracción de Evidencia produjo cinco elementos de evidencia citados, todos resolviéndose a fragmentos reales; el agente de Puntuación propuso; y el motor COMPROMETIÓ en el nivel CMMI 1, con el RunResult completo persistido y toda la topología visible como un solo árbol de trazas Phoenix.

Un nivel 1 comprometido es el resultado correcto para esta entrada, no uno débil: la evidencia sostiene un suelo defendible, y el motor comprometió exactamente lo que las reglas permiten. El sistema declinó sobre-reclamar — que es el comportamiento que toda la arquitectura está construida para producir, y la línea más importante de este documento. El arnés de confianza no solo restringió el modelo; moldeó demostrablemente el resultado hacia la defendibilidad.

9. El registro de desviaciones — desafío, resolución, cambio arquit

Cada apartamiento de la arquitectura conceptual se registró a medida que se hacía. Ninguno es una regresión; cada uno fue una respuesta deliberada a fricción de hardware, herramientas o datos. Consolidado (área · original - tal como se construyó · por qué):

Modelo de razonamiento Llama-3.3-Nemotron-Super-49B - Qwen3-32B (vía un intermedio Llama 3 8B). El NIM Nemotron no se servía en DGX Spark; Llama 3 8B carecía de llamadas de herramientas fiables. Costo aceptado y trabajado por ingeniería: un contexto de 8.192 tokens.

Orquestación Flujo YAML NAT - rebanada Python puro; YAML NAT retenido para pruebas de humo MCP. El flujo de control explícito es mucho más depurable que un bucle ReAct YAML; el código de agente sigue siendo portable.

Herramientas de construcción pip - uv. El grafo de dependencias combinado alcanzó

resolution-too-deep de pip; uv lo resuelve en segundos.

Ejecución de Python	Python del sistema único - doble 3.10 + 3.12 con /opt/venv. nvidia-nat necesita 3.11+; la imagen base trae 3.10 y no puede cambiarse.
Fuente de estructura organizativa	Plantilla de madurez, ids punteados (A.1) - programs.xlsx, códigos de centros financieros de seis dígitos; División añadida como nivel. La jerarquía canónica más rica — los códigos son los verdaderos identificadores organizativos.
Transporte MCP (rebanada)	Transporte de subproceso stdio - importación directa en proceso; ruta stdio ejercitada por el flujo NAT. Un proceso, una traza de pila durante el desarrollo de la rebanada; ambas rutas ejecutan el mismo código de módulo.
Alojamiento de servicios	NIM y Phoenix como aplicaciones de Workbench - contenedores Docker del lado del host. Conveniente para la construcción; la promoción a ciclo de vida limpio está seguida, con la limitación de reinicio enunciada.

Dos entradas (orquestación y transporte MCP) son reversibles por configuración — el código se escribió para que pasar a la orquestación NAT y al transporte MCP sea cableado, no reescritura. Junto a él, un registro de trabajo diferido sigue todo lo conscientemente pospuesto con su fase objetivo — el indicador de modo pensamiento de Qwen3 en el asistente LLM, la construcción del banco de evaluación, la calibración de las heurísticas de palabras clave N4/N5 contra datos de referencia, la brecha de etiquetado por dirección del segundo corpus, los tres agentes restantes, la ampliación más allá de la única combinación validada, la actualización del esquema de ids del Generador de Informes, el refinamiento del etiquetador, el movimiento del transporte MCP, la promoción del ciclo de vida de los servicios y el porte eventual de SQLite a PostgreSQL. Nada en él se olvida; todo en él encaja en la estructura existente. La propiedad más importante del estado final del recorrido es que la arquitectura no tuvo que doblarse para alcanzarlo.

10. Lo que demuestra el recorrido

- Hardware de IA de bajo consumo de espacio, configurado a mano. Un DGX Spark (GB10 Grace Blackwell, 128 GB de memoria unificada, aarch64, CUDA 13) llevado del desembalaje a una pila contenedorizada reproducible: servicio de modelo NIM, un servicio de embedding local, observabilidad Phoenix, almacenes vectorial y relacional, y un contenedor de desarrollo AI Workbench — incluida la fricción específica de la plataforma (dobles ejecuciones de Python, ruedas GPU aarch64, bordes afilados del entorno Compose) que solo aparece cuando realmente haces el trabajo.
- Modelos de lenguaje pequeños y medianos, implementados con sus restricciones trabajadas por ingeniería. Un modelo de razonamiento de 32 B y un modelo de embedding compacto sirviendo simultáneamente en un puesto de trabajo, con la ventana de contexto 8 K tratada como una entrada de ingeniería: presupuesto dinámico de tokens, recuperación dimensionada a la ventana, fragmentación ajustada al límite de secuencia del embedder, e indicadores de modo de inferencia gestionados deliberadamente.
- Adaptabilidad, evidenciada por la odisea de los modelos. Tres modelos evaluados en el dispositivo contra un umbral de selección preciso y guiado por la arquitectura — llamadas de herramientas fiables y salida estructurada en este hardware — con cada cambio costando cero

modificaciones de código de agente porque las costuras se diseñaron para ello.

- Una arquitectura enchufable, multicontenedor. Cada modelo y servicio tras una interfaz estándar — APIs compatibles con OpenAI, contratos de herramientas MCP, envolturas de almacén, OTLP — con la propiedad de intercambio ejercitada repetidamente durante la construcción en lugar de afirmada en un diagrama.
- El arnés de confianza como centro de gravedad. Inferencia y comportamiento de los agentes envueltos de extremo a extremo: contratos tipados en cada límite, citas obligatorias y validadas, un motor determinista como único escritor de decisiones comprometidas, una cadena de evidencia persistente que soporta un recorrido hacia atrás a nivel de carácter, memoria en capas que los agentes no pueden corromper, y un árbol de trazas que expone toda la topología de agentes. Probado por una ejecución cuyo resultado más elocuente fue la contención: el sistema comprometió el nivel que la evidencia sostenía de forma defendible, y no más.

Tomado en conjunto, el recorrido es evidencia de un equipo que planifica rigurosamente, afronta la fricción honestamente, la resuelve sin comprometer las garantías de confianza, y registra cada desviación para que el plan y el sistema nunca diverjan silenciosamente — la misma disciplina a la que se comprometen nuestras propuestas futuras.

11. Glosario

IA agéntica Un sistema de IA en el que componentes impulsados por modelos de lenguaje (agentes) planifican, llaman herramientas y se traspasan trabajo para lograr una tarea, en lugar de responder a un solo prompt.

Span Agente / Herramienta / LLM Los tres tipos de span en las trazas de este sistema: uno por límite de agente, uno por llamada de asistente determinista, y uno (autoemitido) por llamada de modelo, anidándose en un solo árbol de trazas por ejecución.

AI Workbench (NVIDIA) El entorno de desarrollo gestionado de NVIDIA que construye y ejecuta el contenedor de desarrollo y los servicios Compose del proyecto.

bge-large-en-v1.5 El modelo de embedding abierto compacto (vectores de dimensión 1.024, límite de secuencia de 512 tokens) servido localmente para toda generación de vectores.

Cadena de evidencia El enlace ininterrumpido y persistido de un puntaje comprometido, de vuelta por su propuesta y elementos de evidencia citados, hasta los tramos de caracteres exactos de los documentos fuente.

Chroma La base de datos vectorial que contiene los fragmentos de documento embebidos; ejecutada embebida en proceso para la PdC, indexada por id de fragmento para unirse limpiamente al almacén relacional.

Fragmento (chunk) Un tramo contiguo de un documento fuente — con id de documento, ordinal, desplazamientos de caracteres y metadatos organizativos — que se embebe, recupera y cita.

CMMI Capability Maturity Model Integration: la escala de madurez de cinco niveles (Inicial, Gestionado, Definido, Gestionado cuantitativamente, En optimización) usada para la puntuación.

DGX Spark El puesto de trabajo de IA de clase escritorio de NVIDIA construido sobre el superchip GB10 Grace Blackwell con 128 GB de memoria unificada CPU/GPU — el hardware

de bajo consumo de espacio en el que se ejecuta toda la PdC.

Dirección La unidad organizativa de evaluación de ECCC, identificada por un código de centro financiero de seis dígitos (p. ej., 332000).

Embedding Una representación vectorial numérica de texto usada para búsqueda de similitud semántica; producida aquí por un modelo pequeño dedicado, nunca por el modelo de razonamiento.

FastMCP El marco Python ligero usado para exponer cada servidor MCP; cada servidor es una envoltura fina sobre un módulo importable.

Barandillas (guardrails) Restricciones impuestas por el código sobre las entradas, salidas y uso de herramientas de los agentes — a diferencia de las instrucciones en los prompts.

Idempotente Producir el mismo resultado al reejecutar contra entradas sin cambios — la propiedad que hace reproducibles la ingesta y la siembra.

MCP (Model Context Protocol) Un estándar abierto para exponer herramientas y datos a sistemas de IA mediante contratos de herramientas tipados; la PdC construyó tres servidores MCP personalizados.

NAT (NeMo Agent Toolkit) El marco agéntico de NVIDIA para componer agentes, herramientas y flujos; usado aquí como cliente MCP para las pruebas de humo de transporte, con la rebanada orquestada en Python puro.

NIM (NVIDIA Inference Microservice) Un servicio de servicio de modelo contenedorizado y optimizado que expone una API compatible con OpenAI; cómo se sirve el modelo de razonamiento.

API compatible con OpenAI La interfaz HTTP de chat/embeddings estándar de facto; cada modelo del sistema se sitúa tras una, lo que hace los modelos intercambiables por configuración.

OpenInference / OpenTelemetry (OTLP) La convención de instrumentación y el protocolo de telemetría usados para capturar cada llamada de modelo y span, enviados a Phoenix.

Phoenix (Arize Phoenix) El backend de observabilidad de código abierto que almacena y visualiza los árboles de trazas; se ejecuta como un solo contenedor local para que todas las trazas permanezcan en el dispositivo.

Procedencia El origen registrado de cada dato y cada inferencia — qué documento, qué fragmento, qué ejecución, qué llamada de modelo lo produjo.

Pydantic La biblioteca de validación de Python usada para cada esquema del sistema; el mecanismo tras «cada traspaso es tipado».

Qwen3-32B El modelo de razonamiento abierto de 32 mil millones de parámetros seleccionado tras evaluación en el dispositivo; servido vía NIM con una ventana de contexto de 8.192 tokens.

RAG (Generación aumentada por recuperación) Suministrar a un modelo material fuente recuperado para fundamentar su salida; aquí, la recuperación es determinista y filtrada por dirección.

ScoreProposal / ScoreDecision El par clave de bóveda: el nivel propuesto del modelo con

justificación y evidencia, y el veredicto comprometido/diferido/rechazado del motor determinista — el único puntaje que el sistema reconoce.

SLM (Modelo de lenguaje pequeño) Un modelo de lenguaje compacto desplegable en hardware restringido; en esta PdC, el modelo de embedding y, relativo a los modelos de frontera, el modelo de razonamiento de 32 B.

Patrón supervisor/trabajador La forma de orquestación: un supervisor valida las entradas e impulsa agentes trabajadores especializados en una secuencia fija y trazable.

Árbol de trazas La estructura de spans anidados de una ejecución en Phoenix — la topología de agentes hecha visible, con una regresión apuntando al span que la produjo.

Arnés de confianza El conjunto combinado de mecanismos impuestos — contratos tipados, citas validadas, compromiso determinista, memoria en capas y observabilidad — que envuelve la inferencia y el comportamiento de los agentes.

uv El resolvedor/instalador rápido de paquetes de Python adoptado tras el fallo de pip al resolver el grafo de dependencias del proyecto.

Almacén vectorial / búsqueda vectorial Una base de datos de embeddings consultada por similitud semántica; cómo se encuentran los fragmentos candidatos de evidencia.

Rebanada vertical Un camino mínimo pero completo de extremo a extremo a través de cada capa del sistema — el entregable y la prueba de la fase 6.

FIN DEL DOCUMENTO