ENKIDU

MIOVSKI GROUP

undefined

1. Résumé

Entre l'approbation de l'architecture conceptuelle et la clôture de la tranche verticale de la phase 6, notre équipe a conçu, configuré, construit et prouvé un système d'IA multi-agent de bout en bout sur un seul poste de travail à faible encombrement — le NVIDIA DGX Spark — pour la preuve de concept d'évaluation de la maturité des capacités d'affaires d'Environnement et Changement climatique Canada. Le système lit la documentation propre à une organisation et produit des scores de maturité CMMI déterministes et étayés par des citations, chaque score portant toute sa dérivation : de la décision engagée, en remontant par la proposition du modèle, jusqu'aux éléments de preuve cités, jusqu'aux plages de caractères exactes des documents sources exacts.

Ce document raconte honnêtement l'histoire de la mise en ?uvre : ce qui était prévu, ce que nous avons rencontré lorsque le plan a affronté le matériel réel, le comportement réel du modèle et les données réelles, comment chaque défi a été résolu et comment ces résolutions ont modifié l'architecture. Aucun de ces changements n'était une régression ; chacun était une réponse d'ingénierie délibérée, consignée à l'époque dans un registre des écarts tel que construit. Le résultat est un système fonctionnel dont la prémisse centrale — du déterminisme là où il compte, du raisonnement là où il aide — a survécu intacte à chaque changement, car elle était imposée structurellement plutôt qu'en demandant gentiment au modèle.

Le parcours démontre cinq choses en pratique : (1) la configuration pratique d'un matériel d'IA à faible encombrement et de toute sa pile de service, d'orchestration et d'observabilité ; (2) l'adaptabilité dans la sélection du modèle — trois modèles de raisonnement évalués sur l'appareil jusqu'à ce que l'un atteigne le seuil d'appels d'outils fiables et de sortie structurée dans l'enveloppe matérielle ; (3) une expérimentation rigoureuse avec des modèles de langage petits et moyens, y compris les conséquences concrètes d'ingénierie d'une fenêtre de contexte de 8 K jetons ; (4) une architecture conteneurisée et enfichable où chaque modèle et service se trouve derrière une interface standard et peut être remplacé par configuration, non par code ; et (5) — l'objectif principal de la PdC — un harnais de confiance qui enveloppe chaque inférence et comportement d'agent avec des contrats typés, des citations validées obligatoires, un moteur de notation déterministe, une chaîne de preuves persistante et une observabilité de bout en bout de toute la topologie des agents.

2. Objet et guide de lecture

Ce Parcours de mise en ?uvre accompagne deux documents sources : le document d'architecture de la PdC d'IA agentique original, qui fixait l'intention (une topologie à sept agents, une notation déterministe, une intégration d'outils fondée sur MCP), et le dossier d'architecture tel que construit (Rév. A), qui rapproche cette intention du code sur disque à la clôture de la phase 6. Là où ces documents décrivent le système, celui-ci décrit le chemin : la séquence des phases de construction, les frictions rencontrées, les décisions prises et les conséquences architecturales de chacune.

Il est écrit comme une référence d'expérience démontrée. Chaque défi et résolution décrits ici proviennent du dossier tel que construit et du dépôt du projet — non reconstitués après coup. Les écarts par rapport à l'architecture originale sont présentés délibérément et intégralement, car la capacité à rencontrer des frictions sur un matériel contraint, à les diagnostiquer et à les résoudre sans compromettre les garanties de confiance est précisément l'expérience que ce document atteste.

Note d'état pour l'examinateur : cette ébauche décrit la construction jusqu'à la tranche verticale de la

phase 6 et les travaux de la phase 7 en cours, selon la base de référence Rév. A telle que construite. Si des phases ultérieures (banc d'évaluation, agents restants, élargissement du modèle complet) doivent être reflétées dans la version finale, cet état peut être ajouté aux sections 8 et 9 sur votre indication.

3. Le point de départ — ce que l'architecture conceptuelle prévoyai

L'architecture conceptuelle définissait le problème — évaluer la maturité des capacités d'affaires à partir d'un corpus hétérogène de transcriptions d'entrevues, de documents de processus, de rapports d'incidents et de constats d'audit — et une prémisse de conception qui porte tout le système : du déterminisme là où il compte, du raisonnement là où il aide. Les formules de notation, les grilles, le modèle de capacités et la structure organisationnelle sont une vérité terrain imposée par le code ; le modèle de langage interprète, extrait et raconte, mais n'effectue jamais de calcul et n'attribue jamais de score final. Tout s'exécute sur une infrastructure locale : les documents sources contiennent des informations organisationnelles sensibles, donc rien n'est transmis à une API externe.

TABLEAU 1 · LA PILE PRÉVUE (COUCHE · SÉLECTION · JUSTIFICATION)

Matériel	NVIDIA DGX Spark (GB10 Grace Blackwell, 128 Go de mémoire unifiée CPU/GPU). Souveraineté des données sur un seul poste de travail sur site ; capacité pour un modèle d'environ 49 G avec marge pour les plongements, vecteurs et observabilité ; pile NVIDIA préintégrée.
Modèle de raisonnement	Llama-3.3-Nemotron-Super-49B-v1.5 via NIM. Entraîné pour l'appel d'outils structuré ; raisonnement solide ; déployable sur l'appareil ; une API compatible OpenAI garde le moteur d'exécution interchangeable.
Cadre agentique	NVIDIA NeMo Agent Toolkit (NAT). Composition agnostique au cadre, client MCP natif, points d'ancrage d'évaluation et de traçage intégrés.
Mémoire vectorielle	Chroma. Embarqué/léger ; dimensionné pour des milliers de fragments ; filtrage riche par métadonnées pour les contraintes de capacité/type/poids.
Observabilité	Arize Phoenix (OpenTelemetry / OpenInference). Source ouverte, un seul conteneur, toutes les traces restent locales ; évaluateurs de fondement propres au RAG.
Intégration d'outils	Serveurs Model Context Protocol (MCP), dont des serveurs personnalisés Modèle de capacités, Grille/Notation et Générateur de rapports. Logique déterministe dans le code derrière des signatures d'outils étroites et typées — jamais dans les invites.
Motif d'orchestration	Superviseur/travailleur avec sept agents (Superviseur, Récupération, Extraction de preuves, Notation, Critique, Agrégateur, Narration). Prévisible, à coût borné, traçable ; séparation des tâches entre extraction, notation et narration.
Magasin relationnel	SQLite (évolutif vers PostgreSQL). Le modèle de capacités, les grilles, l'historique des exécutions et les scores sont des connaissances relationnelles, distinctes de la mémoire vectorielle.

La quasi-totalité a survécu au contact du réel ; là où ce n'est pas le cas, les sections 4 à 7 expliquent exactement ce qui a changé et pourquoi.

4. Mise en place de la plateforme — matériel, conteneurs et outillag

La première étape du parcours fut de l'ingénierie de plateforme pure : transformer un DGX Spark neuf en un environnement de développement et de service conteneurisé et reproductible. Trois défis sont arrivés d'emblée, et chaque résolution est devenue une partie permanente et documentée de la construction.

4.1 L'environnement tel que conçu

Le projet s'exécute comme un projet NVIDIA AI Workbench : Workbench gère un conteneur de développement qui monte le dépôt, et une pile Compose exécute les services de service de modèle et d'observabilité à ses côtés. Dans le conteneur, les services se résolvent par nom : le point de terminaison de raisonnement NIM sur le port 8000, un service de plongement local sur le port 8002, Phoenix sur 6006 (IU) avec ingestion OTLP sur 4317/4318, et Chroma et SQLite s'exécutant en processus sur l'état sur disque. Les 128 Go de mémoire unifiée CPU/GPU du DGX Spark contiennent simultanément le modèle de raisonnement, le modèle de plongement, le magasin vectoriel et la pile d'observabilité, avec de la marge — et parce que le matériel est dédié, la latence d'inférence est prévisible, ce qui importe pour l'orchestration multi-agent.

4.2 Défi — un conteneur bi-Python que la plateforme n'avait pas prévu

Défi L'image de base de Workbench livre Python 3.10 système, mais le NeMo Agent Toolkit exige Python 3.11+, et l'IU de Workbench ne permet pas de changer l'image de base. Par ailleurs, JupyterLab et les propres paquets de configuration de Workbench dépendent du maintien exact du Python système.

Résolution La construction du conteneur installe Python 3.12 aux côtés de 3.10 et achemine tout le code du projet via un environnement virtuel à /opt/venv, laissant le Python système intact pour l'outillage de la plateforme. L'arrangement est scripté dans l'étape post-construction du conteneur, donc reproductible à chaque reconstruction plutôt qu'un état machine façonné à la main.

Impact architectural Une conception de conteneur bi-exécution délibérée, encodée dans le script de construction — et un motif réutilisé partout où les valeurs par défaut de la plateforme et les exigences de la chaîne d'outils divergent : satisfaire les deux, dans le code, de façon reproductible.

4.3 Défi — le graphe de dépendances qui a vaincu pip

Défi Le graphe de dépendances combiné — Phoenix, LangChain, ChromaDB, OpenTelemetry, MCP, sentence-transformers et nvidia-nat ensemble — a vaincu le résolveur à rebours de pip, qui a échoué avec resolution-too-deep après environ huit minutes. Sur une plateforme aarch64, il y avait un second piège : installer sentence-transformers naïvement tire la roue PyTorch CPU seule de PyPI, écartant silencieusement le GPU.

Résolution La construction amorce uv, qui résout le même graphe en quelques secondes, et achemine chaque installation de dépendance du projet par lui. PyTorch est installé en premier, explicitement depuis l'index de roues CUDA-13 aarch64, pour que sentence-transformers prenne la roue GPU plutôt que la CPU par défaut.

Impact architectural L'outillage de construction est passé de pip à uv — consigné comme écart

?3 dans le registre tel que construit — et l'ordre d'installation est devenu un fait architectural du script de construction, non un savoir tribal.

4.4 Défi — cycle de vie des conteneurs et l'arête vive du bind-mount

Défi L'architecture originale voulait NIM et Phoenix gérés comme des applications Workbench, partageant l'espace de noms réseau et le cycle de vie du projet. Bien régler la pile Compose a révélé une arête vive : Workbench Compose n'hérite pas de l'environnement HOME de l'utilisateur, donc les chemins de bind-mount relatifs ou basés sur des variables s'étendent silencieusement à des chaînes vides.

Résolution Les deux services s'exécutent comme une pile Compose documentée avec des chemins de bind-mount absolus, des réservations GPU, un dimensionnement de mémoire partagée, des vérifications d'état (dont une période de démarrage généreuse pour le premier chargement du modèle NIM) et des politiques de redémarrage. Pour la PdC, ils s'exécutent comme des conteneurs côté hôte ; les promouvoir en applications Workbench gérées est une tâche de runbook opérateur suivie et consciemment reportée — l'arrangement actuel fonctionne mais ne survit pas proprement aux redémarrages de l'hôte, et le dire clairement fait partie de la discipline du tel-que-construit.

Impact architectural Infrastructure-en-code pour toute la pile de service et d'observabilité, avec sa limitation connue consignée dans le registre des travaux reportés plutôt que découverte par le prochain opérateur.

5. Le parcours des modèles — adaptabilité sous une enveloppe ma

L'étape la plus instructive du parcours fut la sélection du modèle de raisonnement. Le plan nommait un modèle ; la construction en a évalué trois sur l'appareil avant de s'engager. La séquence, et ce que chaque étape a enseigné, est la démonstration la plus claire de ce document de l'adaptation des modèles pour obtenir le meilleur résultat sur un matériel contraint.

5.1 Première tentative — Llama-3.3-Nemotron-Super-49B-v1.5

Le choix de l'architecture. En pratique, tirer et servir le conteneur NIM Nemotron a échoué sur le DGX Spark — le conteneur empaqueté ne se servait pas sur cette plateforme. Plutôt que de lutter contre l'empaquetage, l'équipe a traité le modèle comme ce que l'architecture avait toujours dit qu'il était : un composant remplaçable derrière une API compatible OpenAI.

5.2 Deuxième tentative — Llama 3 8B, et le seuil d'appel d'outils

Un NIM Llama 3 8B plus petit se servait correctement — preuve que le chemin de service fonctionnait — mais il ne prenait pas en charge l'appel d'outils/fonctions de style OpenAI de façon assez fiable. C'est un échec disqualifiant dans cette architecture, car chaque agent dépend d'appels d'outils structurés et d'une sortie JSON valide selon le schéma. L'expérience a établi le seuil de sélection avec précision : non les scores de banc d'essai, mais un comportement fiable de sortie structurée et d'appel d'outils sur ce matériel, dans cette pile de service, pour ces agents.

5.3 L'atterrissage — Qwen3-32B, et l'ingénierie à son enveloppe

Qwen3-32B via NIM a atteint le seuil : un comportement solide d'appel d'outils et de sortie structurée

avec un ajustement GPU confortable dans la mémoire unifiée du Spark. Il est venu avec un vrai compromis — une longueur de contexte maximale de 8 192 jetons, bien en deçà des 32 K+ du Nemotron — et l'équipe a fait de l'ingénierie à cette enveloppe plutôt que de faire comme si de rien n'était :

- Budgétisation dynamique des jetons. L'assistant LLM partagé calcule max_tokens dynamiquement par rapport au budget de contexte restant à chaque appel, de sorte qu'une invite longue ne peut jamais tronquer silencieusement la réponse.
- Récupération dimensionnée à la fenêtre. La récupération tire 5 à 10 fragments par requête plutôt que les 20+ qu'un modèle à contexte 32 K prendrait — une décision délibérée de densité de qualité, non une limitation découverte en production.
- Discipline du mode réflexion. Qwen3 est livré avec le « mode réflexion » activé par défaut ; pour les tâches de sortie structurée, il brûle des jetons en interne et multiplie la latence sans bénéfice de qualité. La configuration du flux NAT le désactive via chat_template_kwargs ; appliquer le même indicateur dans l'assistant LLM en Python pur est un élément de pré-vol suivi — un exemple du registre des travaux reportés attrapant une petite incohérence avant qu'elle ne devienne un mystère.

La leçon stratégique que prouve le parcours : parce que chaque agent parle au modèle via le client OpenAI standard contre un point de terminaison NIM, trois changements de modèle ont coûté zéro changement de code d'agent. La sélection du modèle fut de la configuration plus de la mesure — exactement la propriété enfichable promise par l'architecture.

5.4 Le second modèle — les plongements comme service de petit modèle séparé

La décision de soutien de l'architecture — ne jamais utiliser le modèle de raisonnement pour les plongements — fut implémentée comme son propre service de petit modèle : BAAI/bge-large-en-v1.5 (vecteurs de dimension 1 024) derrière un adaptateur FastAPI compact compatible OpenAI sur le port 8002. Les valeurs par défaut de fragmentation (1 500 caractères cibles avec 200 de chevauchement) ont été réglées à la limite de séquence de 512 jetons du plongeur. Comme pour le modèle de raisonnement, changer le modèle de plongement est un changement de variable d'environnement. Deux modèles, deux charges de travail, deux services bien dimensionnés — la discipline des petits modèles de langage en miniature.

6. Construire la base de preuves — le substrat déterministe

Avant l'exécution de tout agent, la construction a mis en place le substrat déterministe : le flux d'amorçage des données de référence et le pipeline d'ingestion de documents. Les deux sont du traitement de données pur — aucun appel de modèle, aucune orchestration d'agent — et les deux sont idempotents par conception : réexécuter l'un ou l'autre sur des entrées inchangées ne produit aucune nouvelle ligne. C'est ce qui garantit que les agents raisonnent sur une base de preuves stable et reproductible, et c'est la première couche de la chaîne de preuves : chaque fragment porte un identifiant de document dérivé du contenu, un ordinal, des décalages de caractères et des métadonnées organisationnelles, de sorte qu'une citation se résout à une plage exacte d'une source exacte.

6.1 Défi — le modèle organisationnel a changé sous la construction

Défi Les premières phases ont amorcé la structure organisationnelle à partir d'un tableur de modèle de maturité utilisant des identifiants pointés (A.1, A.2). À la mi-phase 6, la source de vérité canonique est passée à programs.xlsx — la vraie hiérarchie des centres financiers, avec des codes à six chiffres, 15 directions générales, 94 directions et 344 divisions, présentée dans un format Excel hiérarchique creux où les cellules héritent de la ligne au-dessus.

Résolution Un chargeur dédié analyse la disposition creuse avec remplissage vers l'avant et déduplication à la première occurrence (une direction peut légitimement apparaître sous plusieurs contextes de programme) ; le schéma a gagné Division comme niveau de première classe et des colonnes de contexte de programme dénormalisées pour qu'une invite puisse dire « cette direction travaille sur X sous Y » sans jointure. L'amorçage est scindé pour que les tables organisationnelles puissent être réinitialisées et réamorcées sans jamais toucher à la taxonomie des capacités, de source indépendante.

Impact architectural L'unité d'évaluation est devenue le vrai schéma d'identifiants organisationnels. Une conséquence a été consciemment reportée : le Générateur de rapports, écrit contre les identifiants pointés, doit être mis à jour vers le schéma à six chiffres avant que la génération de rapports ne soit câblée — suivi dans le registre avec le couplage exact documenté, car la tranche de la phase 6 se termine à une décision engagée et ne l'exerce jamais.

6.2 Défi — étiqueter les fragments à la structure organisationnelle, et une qualité de données honnête

Défi La notation par direction exige de savoir à quelle direction appartient chaque fragment — mais les documents sources n'annoncent pas leur structure organisationnelle de façon uniforme. Un document d'entrevue avait des en-têtes de section propres par nom de direction ; un autre utilisait des en-têtes par nom de personne interrogée et ne nommait jamais aucune direction.

Résolution Une nouvelle étape d'étiquetage a été insérée dans le pipeline d'ingestion — entre le chargement et la fragmentation — qui segmente le texte en sections par correspondances de noms canoniques de la structure organisationnelle (à l'aide des propres outils de recherche de noms du Modèle de capacités), fragmente chaque section séparément et estampille chaque fragment avec la direction, la direction générale et la division de sa section, avec un repli au niveau de la direction générale là où aucune direction ne correspond. Le pipeline compte sept étapes tel que construit : découvrir - charger - document - étiqueter - fragmenter - plonger - persister. L'ingestion des deux corpus d'entrevues a produit 685 fragments ; 112 se sont étiquetés proprement à la direction cible — assez pour exécuter et prouver la tranche — et le résultat de zéro direction du second document a été consigné comme une lacune de qualité de données nommée avec deux correctifs candidats (une pré-passe d'insertion d'en-têtes, ou une carte de correspondance personne interrogée-direction organisée), planifiée pour une phase ultérieure.

Impact architectural Le pipeline a gagné une étape ; les identifiants de fragments reflètent

délibérément la structure du document sous le modèle organisationnel actuel ; et une limitation de qualité de données a été mesurée, quantifiée et suivie au lieu d'être camouflée — la même discipline d'honnêteté que le moteur de notation applique aux preuves.

7. La couche logique MCP — enfichable par construction

Trois serveurs Model Context Protocol personnalisés détiennent la vérité terrain imposée par le code : le MCP Modèle de capacités (pures recherches sur la taxonomie et la structure organisationnelle — aucun LLM, aucune mutation par les agents), le MCP Grille/Notation (la grille CMMI projetée sur une capacité, plus le moteur de notation déterministe) et le MCP Générateur de rapports (assemblage de livrables à partir de décisions engagées). Chaque serveur est un mince habillage FastMCP sur un module importable — un choix de conception qui a directement payé lorsque la friction de transport est arrivée.

7.1 Défi — déboguer à travers les frontières de processus

Défi La conception originale fait atteindre par les agents les outils MCP via le transport de sous-processus stdio. Pendant le développement de la tranche, le débogage multi-processus — plusieurs serveurs lancés, échecs entrelacés — a ralenti le diagnostic exactement quand l'itération rapide importait le plus.

Résolution Les deux chemins ont été conservés. La tranche d'agents de la phase 6 importe directement les modules sous-jacents des serveurs, en processus — un processus, une trace de pile — tandis qu'une définition de flux NAT lance les trois serveurs comme sous-processus stdio exactement comme conçu et pointe un agent ReAct sur l'ensemble complet des outils contre le NIM Qwen3-32B local. Le chemin de transport est donc prouvé, exerçable en continu, et déplacer les agents dessus est un changement de câblage, non une réécriture, car les deux chemins exécutent le même code de module.

Impact architectural Consigné comme écart réversible par configuration. Les contrats MCP — non le transport — sont l'architecture.

7.2 L'inventaire enfichable

À la clôture de la tranche, chaque composant substituable se trouvait derrière une couture standard. C'est le sens concret de « plusieurs conteneurs et modèles rendant l'architecture enfichable » :

Modèle de raisonnement (Qwen3-32B) Derrière une API de conversation compatible OpenAI servie par un conteneur NIM. Se change par configuration seulement — prouvé trois fois lors de la sélection (Nemotron - Llama 3 8B - Qwen3-32B) avec zéro changement de code d'agent.

Modèle de plongement (bge-large-en-v1.5) Derrière une API de plongements compatible OpenAI derrière un conteneur adaptateur FastAPI. Se change par variables d'environnement (point de terminaison, modèle, dimension).

Magasin vectoriel (Chroma) Derrière un mince habillage de magasin appartenant au projet. PersistentClient embarqué aujourd'hui ; se change en mode service

HttpClient sans changement pour les appelants.

Magasin relationnel (SQLite) Derrière des classes de magasin possédant leur DDL. Schéma simple ; le portage PostgreSQL est suivi comme une tâche de phase ultérieure mesurée en heures.

Logique déterministe (3 serveurs MCP) Derrière des contrats d'outils MCP sur FastMCP ; aussi des modules directement importables. En processus pour la tranche ; le transport de sous-processus stdio prouvé par le flux NAT — passer de l'un à l'autre est de la configuration.

Observabilité (Phoenix) Derrière un point de terminaison OTLP OpenTelemetry ; auto-instrumentation OpenInference. Un seul conteneur ; le point de terminaison et le nom du projet sont des réglages d'environnement ; le traçage peut être désactivé par indicateur pour les tests.

Orchestration (tranche Python pur) Classes d'agents typées avec méthodes run() ; YAML de flux NAT à côté. Le code d'agent est portable ; élever l'orchestration dans le YAML NAT est un changement de câblage ultérieur, le flux de test de fumée MCP démontrant déjà le motif.

Chaque modèle et service derrière une couture standard. La promesse de l'architecture — changer par configuration, non par code — a été exercée à répétition durant la construction, non simplement affirmée.

8. L'objectif principal — le harnais de confiance en pratique

Tout ce qui précède est la scène ; ceci est la pièce. L'objectif central de la PdC était de démontrer que l'inférence et le comportement des agents peuvent être enveloppés de bout en bout dans un harnais de confiance : une chaîne de preuves, de provenance et de traçabilité pour chaque affirmation, un engagement déterministe de chaque décision conséquente, et l'observabilité de toute la topologie des agents. Tel que construit, ce harnais repose sur quatre invariants porteurs, chacun imposé par le code :

1. Le déterminisme possède le score Le LLM propose un niveau ; engine.evaluate() l'engage. Le moteur est, par construction, le point unique de tout le système où un score devient ENGAGÉ — aucun autre chemin de code ne peut écrire un niveau engagé. Le modèle construit une ScoreProposal ; le moteur retourne une ScoreDecision ; le niveau engagé sur cette décision est la seule valeur que le système reconnaît.

2. Chaque transfert est typé Aucun dictionnaire libre ne franchit une frontière d'agent ou d'outil — seulement des modèles validés par Pydantic (RunRequest - RetrievalResult - EvidenceBundle - ScoreProposal - ScoreDecision - RunResult), validés du côté récepteur. Les éléments invalides sont écartés à la frontière, non découverts en aval.

3. Les citations sont obligatoires et validées Chaque élément de preuve porte un chunk_id requis. L'agent d'Extraction de preuves filtre chaque identifiant émis par le modèle contre

	l'ensemble des fragments réellement récupérés dans cette exécution — une citation hallucinée ne peut survivre au filtre, donc « chaque affirmation se résout à une source réelle » est une propriété imposée, non un espoir.
4. La topologie est observable	Les frontières d'agents émettent des spans AGENT, les assistants déterministes émettent des spans TOOL, et OpenInference auto-instrumente chaque appel de modèle comme un span LLM imbriqué sous son agent appelant. Phoenix montre l'exécution comme un arbre de traces — la structure du système, non un journal plat — et une régression pointe droit sur le span qui l'a produite.

8.1 La clé de voûte — un modèle propose, un moteur engage

La transition de ScoreProposal à ScoreDecision est la clé de voûte architecturale. Quand l'agent de Notation a lu la grille et les preuves, il propose le niveau CMMI que les preuves soutiendront de façon défendable — l'invite est explicite qu'il doit être conservateur — et soumet la proposition au moteur déterministe. Le moteur applique quatre règles dans l'ordre : citations obligatoires ; le niveau dans la plage 1 à 5 (une garde de schéma en défense en profondeur) ; suffisance des preuves proportionnée au niveau revendiqué ; et intégrité référentielle — chaque identifiant cité doit exister. L'échelle de suffisance est explicite :

N1 — Initial ? 1 élément de preuve, toute polarité.

N2–N3 — Géré / Défini ? 2 éléments, ? 1 positif.

N4 — Géré quantitativement ? 3 éléments, ? 2 positifs, et un signal de mesure/métriques.

N5 — En optimisation ? 4 éléments, ? 3 positifs, et des signaux de mesure et d'amélioration continue.

Une proposition qui échoue retourne DIFFÉRÉE — le système refuse d'engager et fait remonter la lacune pour examen humain plutôt que de deviner ; une violation de plage retourne REJETÉE. Le moteur n'ajuste jamais un niveau : il engage exactement ce qui a été proposé, ou décline.

8.2 Chaîne de preuves, provenance et la remontée d'audit

Le harnais rend chaque score engagé auditable par remontée. Une décision persiste dans la table score_decisions sous son identifiant d'exécution ; la décision porte les identifiants des fragments de preuve ; chaque élément de preuve porte sa citation, son assertion et sa polarité contre un chunk_id requis ; chaque fragment porte son identifiant de document, son ordinal, sa page et ses décalages de caractères ; et chaque identifiant de document est un hachage de contenu du texte source. Pour tout score que le système engage, vous pouvez donc remonter — décision, vers proposition, vers éléments de preuve, vers identifiants de fragments, vers les plages de caractères exactes des documents sources exacts — et identifier précisément où le modèle de langage a touché la chaîne et où il ne l'a pas pu. La mémoire est en couches pour protéger ceci : la base de connaissances (Chroma plus les tables de référence relationnelles) n'est écrite que par l'ingestion et les amorceurs, jamais par les agents ; la mémoire de travail est les objets de transfert typés dans une exécution ; la mémoire d'exécution est la table des décisions persistée.

8.3 Envelopper l'inférence et le comportement d'observabilité

L'observabilité a été conçue pour rendre le système multi-agent lisible, non simplement journalisé. Importer le paquet de configuration du projet câble le traçage exactement une fois, de sorte que tout point d'entrée obtient l'instrumentation Phoenix automatiquement. Le Superviseur émet le span AGENT racine ; chaque travailleur s'imbrique dessous ; chaque appel d'assistant déterministe — embed_query, vector_search, render_rubric, engine_evaluate, mise à jour du magasin de décisions — est un span TOOL ; chaque appel de modèle se trace automatiquement avec l'invite complète, la réponse, les comptes de jetons et la latence. Deux raffinements délibérés montrent la discipline : les appels de plongement sont supprimés de l'instrumentation (déterministes, à haut volume, et un risque de limitation de débit pendant l'ingestion en masse), et la structure des spans découle du code par construction — ajouter un agent futur signifie envelopper son run() dans agent_span(), rien de plus.

8.4 La preuve — la tranche verticale de la phase 6

Le critère de succès de la tranche était une combinaison capacité × direction fonctionnelle, de bout en bout. L'exécution a évalué la capacité INF.DQA (Qualité et exactitude des données) pour la direction 332000 (Application de la loi sur la faune) : la récupération filtrée aux 112 fragments étiquetés à cette direction ; l'agent d'Extraction de preuves a produit cinq éléments de preuve cités, tous se résolvant à des fragments réels ; l'agent de Notation a proposé ; et le moteur a ENGAGÉ au niveau CMMI 1, avec le RunResult complet persisté et toute la topologie visible comme un seul arbre de traces Phoenix.

Un niveau 1 engagé est le résultat correct pour cette entrée, non un résultat faible : les preuves soutiennent un plancher défendable, et le moteur a engagé exactement ce que les règles permettent. Le système a décliné de sur-revendiquer — ce qui est le comportement que toute l'architecture est bâtie pour produire, et la ligne la plus importante de ce document. Le harnais de confiance n'a pas seulement contraint le modèle ; il a manifestement orienté le résultat vers la défendabilité.

9. Le registre des écarts — défi, résolution, changement architectu

Chaque écart par rapport à l'architecture conceptuelle a été consigné au fur et à mesure. Aucun n'est une régression ; chacun était une réponse délibérée à une friction de matériel, d'outillage ou de données. Consolidé (domaine · original - tel que construit · pourquoi) :

Modèle de raisonnement Llama-3.3-Nemotron-Super-49B - Qwen3-32B (via un intermédiaire Llama 3 8B). Le NIM Nemotron ne se servait pas sur DGX Spark ; Llama 3 8B manquait d'appels d'outils fiables. Coût accepté et pris en compte : un contexte de 8 192 jetons.

Orchestration Flux YAML NAT - tranche Python pur ; YAML NAT conservé pour les tests de fumée MCP. Un flux de contrôle explicite est bien plus débogable qu'une boucle ReAct YAML ; le code d'agent reste portable.

Outillage de construction pip - uv. Le graphe de dépendances combiné a atteint resolution-too-deep de pip ; uv le résout en quelques secondes.

Exécution Python Python système unique - double 3.10 + 3.12 avec /opt/venv. nvidia-nat a besoin de 3.11+ ; l'image de base livre 3.10 et ne peut être changée.

Source de structure organisationnelle	Modèle de maturité, identifiants pointés (A.1) - programs.xlsx, codes de centres financiers à six chiffres ; Division ajoutée comme niveau. La hiérarchie canonique plus riche — les codes sont les vrais identifiants organisationnels.
Transport MCP (tranche)	Transport de sous-processus stdio - import direct en processus ; chemin stdio exercé par le flux NAT. Un processus, une trace de pile pendant le développement de la tranche ; les deux chemins exécutent le même code de module.
Hébergement de services	NIM et Phoenix comme applications Workbench - conteneurs Docker côté hôte. Expédient pour la construction ; la promotion à cycle de vie propre est suivie, avec la limitation de redémarrage énoncée.

Deux entrées (orchestration et transport MCP) sont réversibles par configuration — le code a été écrit pour que passer à l'orchestration NAT et au transport MCP soit du câblage, non une réécriture. À ses côtés, un registre des travaux reportés suit tout ce qui est consciemment reporté avec sa phase cible — l'indicateur de mode réflexion Qwen3 dans l'assistant LLM, la construction du banc d'évaluation, le calibrage des heuristiques par mots-clés N4/N5 contre des données de référence, la lacune d'étiquetage par direction du second corpus, les trois agents restants, l'élargissement au-delà de la seule combinaison validée, la mise à jour du schéma d'identifiants du Générateur de rapports, le raffinement de l'étiqueteur, le déplacement du transport MCP, la promotion du cycle de vie des services et le portage éventuel de SQLite vers PostgreSQL. Rien n'y est oublié ; tout s'y insère dans la structure existante. La propriété la plus importante de l'état final du parcours est que l'architecture n'a pas eu à plier pour l'atteindre.

10. Ce que le parcours démontre

- Matériel d'IA à faible encombrement, configuré à la main. Un DGX Spark (GB10 Grace Blackwell, 128 Go de mémoire unifiée, aarch64, CUDA 13) pris du déballage à une pile conteneurisée reproductible : service de modèle NIM, un service de plongement local, l'observabilité Phoenix, magasins vectoriel et relationnel, et un conteneur de développement AI Workbench — y compris la friction propre à la plateforme (doubles exécutions Python, roues GPU aarch64, arêtes vives de l'environnement Compose) qui n'apparaît que lorsqu'on fait réellement le travail.
- Modèles de langage petits et moyens, implémentés avec leurs contraintes prises en compte. Un modèle de raisonnement de 32 G et un modèle de plongement compact servant simultanément sur un poste de travail, avec la fenêtre de contexte 8 K traitée comme une entrée d'ingénierie : budgétisation dynamique des jetons, récupération dimensionnée à la fenêtre, fragmentation réglée à la limite de séquence du plongeur, et indicateurs de mode d'inférence gérés délibérément.
- Adaptabilité, attestée par l'odyssée des modèles. Trois modèles évalués sur l'appareil contre un seuil de sélection précis et piloté par l'architecture — appels d'outils fiables et sortie structurée sur ce matériel — chaque changement coûtant zéro modification de code d'agent parce que les coutures étaient conçues pour cela.
- Une architecture enfichable, multi-conteneurs. Chaque modèle et service derrière une interface standard — API compatibles OpenAI, contrats d'outils MCP, habillages de magasin, OTLP — avec la propriété de changement exercée à répétition durant la construction plutôt qu'affirmée

sur un schéma.

- Le harnais de confiance comme centre de gravité. Inférence et comportement des agents enveloppés de bout en bout : contrats typés à chaque frontière, citations obligatoires et validées, un moteur déterministe comme unique rédacteur des décisions engagées, une chaîne de preuves persistante soutenant une remontée au niveau du caractère, une mémoire en couches que les agents ne peuvent corrompre, et un arbre de traces qui expose toute la topologie des agents. Prouvé par une exécution dont le résultat le plus éloquent fut la retenue : le système a engagé le niveau que les preuves soutenaient de façon défendable, et pas plus.

Pris ensemble, le parcours est la preuve d'une équipe qui planifie rigoureusement, affronte les frictions honnêtement, les résout sans compromettre les garanties de confiance, et consigne chaque écart pour que le plan et le système ne divergent jamais silencieusement — la même discipline à laquelle nos propositions futures s'engagent.

11. Glossaire

IA agentique Un système d'IA dans lequel des composants pilotés par un modèle de langage (agents) planifient, appellent des outils et se transmettent le travail pour accomplir une tâche, plutôt que de répondre à une seule invite.

Span Agent / Outil / LLM Les trois types de spans dans les traces de ce système : un par frontière d'agent, un par appel d'assistant déterministe, et un (auto-émis) par appel de modèle, s'imbriquant en un seul arbre de traces par exécution.

AI Workbench (NVIDIA) L'environnement de développement géré de NVIDIA qui construit et exécute le conteneur de développement et les services Compose du projet.

bge-large-en-v1.5 Le modèle de plongement ouvert compact (vecteurs de dimension 1 024, limite de séquence de 512 jetons) servi localement pour toute génération de vecteurs.

Chaîne de preuves Le lien ininterrompu et persisté d'un score engagé, en remontant par sa proposition et ses éléments de preuve cités, jusqu'aux plages de caractères exactes des documents sources.

Chroma La base de données vectorielle contenant les fragments de documents plongés ; exécutée embarquée en processus pour la PdC, indexée par identifiant de fragment pour se joindre proprement au magasin relationnel.

Fragment (chunk) Une plage contiguë d'un document source — avec identifiant de document, ordinal, décalages de caractères et métadonnées organisationnelles — qui est plongée, récupérée et citée.

CMMI Capability Maturity Model Integration : l'échelle de maturité à cinq niveaux (Initial, Géré, Défini, Géré quantitativement, En optimisation) utilisée pour la notation.

DGX Spark Le poste de travail d'IA de classe bureau de NVIDIA bâti sur la superpuce GB10 Grace Blackwell avec 128 Go de mémoire unifiée CPU/GPU — le matériel à faible encombrement sur lequel toute la PdC s'exécute.

Direction L'unité organisationnelle d'évaluation d'ECCC, identifiée par un code de centre financier à six chiffres (p. ex. 332000).

Plongement (embedding) Une représentation vectorielle numérique de texte utilisée pour la recherche de similarité sémantique ; produite ici par un petit modèle dédié, jamais par le modèle de raisonnement.

FastMCP Le cadre Python léger utilisé pour exposer chaque serveur MCP ; chaque serveur est un mince habillage sur un module importable.

Garde-fous Contraintes imposées par le code sur les entrées, sorties et l'usage d'outils des agents — par opposition aux instructions dans les invites.

Idempotent Produire le même résultat lorsqu'on réexécute sur des entrées inchangées — la propriété qui rend l'ingestion et l'amorçage reproductibles.

MCP (Model Context Protocol) Un standard ouvert pour exposer outils et données aux systèmes d'IA via des contrats d'outils typés ; la PdC a construit trois serveurs MCP personnalisés.

NAT (NeMo Agent Toolkit) Le cadre agentique de NVIDIA pour composer agents, outils et flux ; utilisé ici comme client MCP pour les tests de fumée de transport, la tranche étant orchestrée en Python pur.

NIM (NVIDIA Inference Microservice) Un service de service de modèle conteneurisé et optimisé exposant une API compatible OpenAI ; la façon dont le modèle de raisonnement est servi.

API compatible OpenAI L'interface HTTP de conversation/plongements standard de facto ; chaque modèle du système se trouve derrière une telle interface, ce qui rend les modèles interchangeables par configuration.

OpenInference / OpenTelemetry (OTLP) La convention d'instrumentation et le protocole de télémétrie utilisés pour capturer chaque appel de modèle et chaque span, expédiés à Phoenix.

Phoenix (Arize Phoenix) Le back-end d'observabilité open source qui stocke et visualise les arbres de traces ; s'exécute comme un seul conteneur local pour que toutes les traces restent sur l'appareil.

Provenance L'origine consignée de chaque donnée et de chaque inférence — quel document, quel fragment, quelle exécution, quel appel de modèle l'a produit.

Pydantic La bibliothèque de validation Python utilisée pour chaque schéma du système ; le mécanisme derrière « chaque transfert est typé ».

Qwen3-32B Le modèle de raisonnement ouvert de 32 milliards de paramètres sélectionné après évaluation sur l'appareil ; servi via NIM avec une fenêtre de contexte de 8 192 jetons.

RAG (Génération augmentée par récupération) Fournir à un modèle du matériel source récupéré pour ancrer sa sortie ; ici, la récupération est déterministe et filtrée par direction.

ScoreProposal / ScoreDecision La paire clé de voûte : le niveau proposé du modèle avec justification et preuves, et le verdict engagé/différé/rejeté du moteur déterministe — le seul score que le système reconnaît.

SLM (Petit modèle de langage) Un modèle de langage compact déployable sur un matériel contraint ; dans cette PdC, le modèle de plongement et, par rapport aux modèles de pointe, le modèle de raisonnement de 32 G.

Motif superviseur/travailleur La forme d'orchestration : un superviseur valide les entrées et pilote des agents travailleurs spécialisés dans une séquence fixe et traçable.

Arbre de traces La structure de spans imbriqués d'une exécution dans Phoenix — la topologie des agents rendue visible, une régression pointant sur le span qui l'a produite.

Harnais de confiance L'ensemble combiné de mécanismes imposés — contrats typés, citations validées, engagement déterministe, mémoire en couches et observabilité — qui enveloppe l'inférence et le comportement des agents.

uv Le résolveur/installateur de paquets Python rapide adopté après l'échec de pip à résoudre le graphe de dépendances du projet.

Magasin vectoriel / recherche vectorielle Une base de données de plongements interrogée par similarité sémantique ; la façon dont les fragments candidats de preuve sont trouvés.

Tranche verticale Un chemin minimal mais complet de bout en bout à travers chaque couche du système — le livrable et la preuve de la phase 6.

FIN DU DOCUMENT