Plan Directeur Technique : Conception de "Neo-DIAGDEF", Plateforme Souveraine de Diagnostic Industriel et de Fiabilisation Cognitive

1. Vision Stratégique et Cadrage du Projet

1.1. L'Impératif de la Maintenance Cognitive dans l'Industrie 4.0

L'industrie contemporaine affronte un paradoxe croissant : alors que les équipements de production et les systèmes d'armes atteignent des niveaux de complexité sans précédent, intégrant mécanique de précision, électronique embarquée et couches logicielles [1], l'expertise humaine nécessaire à leur maintien en condition opérationnelle (MCO) se raréfie ou se fragmente. Les systèmes traditionnels de Gestion de Maintenance Assistée par Ordinateur (GMAO) ont réussi la numérisation des processus administratifs — gestion des stocks, planification des ordres de travail, suivi budgétaire — mais ils échouent systémiquement sur le plan technique opérationnel : l'aide au diagnostic.

Dans ce contexte, le logiciel DIAGDEF, fondé sur la méthode MAXER, a établi un paradigme robuste : la structuration du raisonnement de dépannage par une approche systémique et causale.[2] Cependant, l'évolution technologique impose aujourd'hui de transcender cet héritage. Il ne s'agit plus seulement de fournir un outil de saisie graphique des pannes, mais de développer un écosystème "Neo-DIAGDEF" capable de fusionner la rigueur méthodologique historique avec les capacités cognitives de l'Intelligence Artificielle (IA) générative et la robustesse des architectures distribuées modernes.

Ce projet vise à développer une application identique à DIAGDEF dans sa philosophie — le respect absolu de la méthode MAXER — mais radicalement nouvelle dans son exécution technique. L'objectif est de déployer une solution "Offline-First", souveraine (déployable en environnements défense ou nucléaire déconnectés), interopérable via les standards S3000L et MIMOSA, et augmentée par des agents IA capables de pré-générer des arbres de défaillances et d'assister le technicien en temps réel.

1.2. Architecture de la Valeur et ROI Attendu

Le développement de cette solution répond à trois impératifs économiques et opérationnels majeurs. Premièrement, la réduction du Temps Moyen de Réparation (MTTR) : en guidant le technicien via un graphe causal probabiliste plutôt qu'une intuition empirique, l'outil vise à éliminer les tâtonnements et les remplacements de pièces inutiles (No Fault Found).[2]

Deuxièmement, la capitalisation du savoir "tribal". Le départ à la retraite des experts emporte souvent la "mémoire" des pannes rares ; Neo-DIAGDEF doit capturer ce raisonnement sous forme de graphes de connaissances (Knowledge Graphs) exploitables par l'IA.[4] Enfin, la conformité réglementaire : dans les secteurs à haute criticité (aéronautique, défense), la traçabilité complète du raisonnement de diagnostic est une exigence de sécurité et de qualité que les champs "commentaires" des GMAO standards ne peuvent satisfaire.[5]

---

2. Déconstruction Méthodologique : L'Héritage DIAGDEF et MAXER

Pour répliquer l'efficacité de DIAGDEF, le développement ne peut se contenter de cloner une interface ; il doit implémenter rigoureusement les contraintes de la méthode MAXER. Cette méthode n'est pas une simple procédure, mais une ontologie de la défaillance qui doit dicter le schéma de la base de données.

2.1. L'Ontologie Stricte : Le Couple Objet-Défaut

Le fondement de la méthode MAXER, et donc de l'application cible, est le refus du langage naturel non structuré pour la description initiale du problème. L'architecture des données doit imposer une granularité sémantique précise. Une panne n'est jamais définie par une phrase libre (ex: "La pompe fait du bruit"), mais par l'association stricte et unique d'un **Objet** et d'un **Défaut**.[4]

Cette contrainte a des implications profondes pour le modèle de données. L'application doit gérer une bibliothèque hiérarchique d'objets (Functional Locations), importée de la GMAO ou du standard S3000L, et une bibliothèque de défauts standardisés (ex: ISO 14224). Chaque nœud du raisonnement est une instance de cette paire. Cette structuration est cruciale pour deux raisons : elle force le technicien à analyser le système plutôt qu'à décrire un ressenti, et elle génère des données propres ("clean data") indispensables pour l'entraînement des algorithmes d'IA ultérieurs, évitant le coûteux nettoyage de données textuelles hétérogènes.

2.2. Le Processus de Dépannage Rationnel (Les 10 Étapes)

L'application doit guider l'utilisateur à travers un workflow séquentiel, empêchant de sauter les étapes critiques de validation. Le moteur de workflow du logiciel doit implémenter les phases suivantes [7] :

Phase Méthodologique	Implémentation Logicielle Requise	Objectif Technique
1. Analyse de la Situation	Formulaire structuré de	Créer le nœud racine du

	saisie du Symptôme (Objet + Défaut) et du contexte opérationnel (Mode de marche, environnement).	graphe de défaillance.
2. Décision d'Urgence	Champ de décision binaire (Arrêt / Mode dégradé / Poursuite) avec justification.	Sécuriser l'installation avant le diagnostic profond.
3. Diagnostic (Hypothèses)	Interface graphique d'édition de graphe (Défaillogramme). Ajout de nœuds "Causes Possibles".	Construire l'arbre causal visuel.
4. Vérification	Pour chaque hypothèse, obligation de saisir une "Méthode de vérification" et un "Résultat" (Disculpé/Incriminé).	Transformer l'hypothèse en fait avéré ou rejeté.
5. Intervention	Lien vers la gamme opératoire ou l'Ordre de Travail (OT) GMAO.	Exécuter la réparation physique.
6. Analyse (Cause Première)	Identification explicite du nœud feuille "Root Cause" dans le graphe. Classification (Matériel, Humain, Organisation).	Identifier la racine systémique pour le REX.
7-10. Fiabilisation	Module de génération de plan d'action et d'enrichissement de la base de connaissances.	Boucler le cycle REX (Retour d'Expérience).

2.3. Le Défaillogramme : Spécification du Moteur Graphique

La fonctionnalité centrale de DIAGDEF est la représentation graphique de la causalité. Le projet doit inclure un éditeur visuel performant permettant de manipuler des chaînes causales

complexes. Contrairement aux arbres de défaillances statiques (Fault Trees) utilisés en sûreté de fonctionnement (conception), les défaillogrammes MAXER sont dynamiques et construits en temps réel pendant le dépannage.[3] L'interface doit permettre de distinguer visuellement les états des nœuds (ex: Rouge pour "Cause Avérée", Gris pour "Disculpé", Orange pour "Suspect"). Elle doit également gérer la notion de "Facteur Contributif" (une cause secondaire qui a favorisé la panne sans la déclencher directement) et les liaisons logiques (Portes ET/OU). Cette visualisation est essentielle pour que le raisonnement soit auditable par un tiers ou par l'IA.[3]

---

3. Architecture Technique : Résilience et Souveraineté

L'intégration dans des environnements militaires ou industriels critiques impose des contraintes de disponibilité et de sécurité qui excluent les architectures purement SaaS (Software as a Service). L'architecture proposée est "Local-First", garantissant l'opérabilité totale en l'absence de réseau.

3.1. Stack Technologique et Choix d'Infrastructure

Pour répondre aux exigences de performance graphique et de calcul IA, tout en assurant la portabilité sur des terminaux durcis, la stack technologique suivante est préconisée :

- **Frontend (Interface Utilisateur) :** Utilisation de **React.js** couplé à la bibliothèque **React Flow** pour le moteur de diagrammes. React Flow offre les primitives nécessaires (nœuds, arêtes, mini-map, contrôles interactifs) pour recréer l'expérience DIAGDEF avec une ergonomie moderne. Le tout est encapsulé dans une Progressive Web App (PWA) ou une application Electron pour le déploiement sur tablettes de terrain et postes fixes.[9]
- **Backend (Logique Métier et IA) : Python** est le choix stratégique incontournable. Il permet d'unifier la logique applicative (via **FastAPI** pour sa performance asynchrone) et les pipelines d'Intelligence Artificielle (PyTorch, LangChain, bibliothèques de graphes). Cette unification simplifie la maintenance et le déploiement par rapport à une architecture polyglotte (Node.js + Python).[11]
- **Base de Données Hybride :** Le stockage des données requiert une approche polyglotte pour gérer efficacement les relations complexes :
    - **PostgreSQL** pour les données relationnelles classiques (utilisateurs, équipements, droits, logs).
    - **Neo4j** ou **ArangoDB** pour le stockage natif des Défaillogrammes. La structure de MAXER étant intrinsèquement un graphe (Symptôme -> Causes -> Causes Premières), une base orientée graphe permet des requêtes de traversée et de détection de motifs (Pattern Matching) infiniment plus performantes que des jointures SQL récursives, surtout pour l'analyse de similarité par l'IA.
    - **PouchDB / RxDB** pour la persistance locale sur le client (navigateur/tablette), assurant le fonctionnement Offline.

3.2. Stratégie "Offline-First" pour les Opérations de Terrain

Dans les contextes de défense (bases avancées, navires) ou industriels (sous-sols, zones blindées), la connectivité est intermittente ou inexistante. L'application ne doit pas simplement "gérer le cache", mais être conçue comme une application distribuée.[12] L'architecture de synchronisation repose sur le protocole de réplication CouchDB. Le client (PouchDB) stocke l'intégralité des données nécessaires à la mission du technicien (sous-ensemble de la flotte d'équipements). Toutes les opérations (création de diagnostic, modification de graphe) sont effectuées sur la base locale. Lorsque la connectivité est rétablie, une synchronisation bidirectionnelle se déclenche. La gestion des conflits (Conflict Resolution) est critique : si deux techniciens modifient le même arbre de défaillance simultanément, l'application doit appliquer des règles de fusion déterministes (ex: l'ajout d'une branche est toujours conservé, la modification d'un statut privilégie l'état le plus critique) ou solliciter un arbitrage manuel lors de la synchronisation.[13]

3.3. Sécurité "Defense-Grade" et Déploiement On-Premise

La souveraineté des données est un prérequis absolu pour les acteurs de la défense et de l'énergie nucléaire cités dans les cas d'usage DIAGDEF.[14] L'envoi de données de pannes (qui peuvent révéler l'état de disponibilité d'une flotte stratégique) vers des cloud publics est exclu. L'application doit être conteneurisée (Docker/Kubernetes) pour permettre un déploiement "Air-Gapped" (totalement isolé d'Internet) sur des serveurs tactiques ou des datacenters privés. La sécurité applicative doit inclure une authentification forte (MFA, intégration aux cartes à puce militaires via OIDC), un chiffrement AES-256 des bases de données locales et serveurs, et une journalisation immuable de toutes les actions pour l'auditabilité. L'IA elle-même doit tourner en local (Inférence On-Premise), ce qui dimensionne le matériel serveur (besoin de GPU type NVIDIA L40S ou A100 en local).[16]

---

4. Intégration Systémique : GMAO et Standards d'Interopérabilité

Neo-DIAGDEF ne doit pas être un îlot de données isolé. Sa valeur réside dans sa capacité à s'insérer dans le flux de travail existant de la GMAO tout en respectant les standards internationaux de soutien logistique.

4.1. Protocole d'Intégration GMAO Bidirectionnelle

L'intégration doit être transparente pour l'utilisateur final. Le flux de données proposé est le suivant :

1. **Déclenchement (GMAO -> Neo-DIAGDEF) :** Lorsqu'un Ordre de Travail (OT) correctif est créé dans la GMAO (SAP PM, IBM Maximo, Carl Source), un **Webhook** est envoyé à

Neo-DIAGDEF. Ce payload JSON contient l'ID de l'équipement, le code fonctionnel, et la description initiale. Neo-DIAGDEF instancie alors un "Brouillon de Diagnostic" pré-rempli et notifie le technicien.[18]

2. **Contextualisation (Deep Linking) :** Dans l'interface de la GMAO, un lien contextuel (URL Scheme neodiagdef://open?wo_id=12345) permet d'ouvrir l'application de diagnostic directement sur le bon dossier, assurant une continuité numérique fluide sans double saisie.
3. **Clôture et Enrichissement (Neo-DIAGDEF -> GMAO) :** Une fois la cause première identifiée et validée, Neo-DIAGDEF renvoie à la GMAO les données structurées : le code de la cause (pour les statistiques Pareto), le temps passé, et un lien vers le rapport PDF du défaillogramme archivé. Cela enrichit l'historique de la GMAO avec une profondeur technique qu'elle ne possède pas nativement.[20]

4.2. Adhésion aux Standards S3000L et MIMOSA

Pour garantir la pérennité et l'échangeabilité des données, notamment dans les programmes multinationaux (OTAN, programmes d'armement européens), le modèle de données interne doit s'aligner sur les standards de l'ASD (Aerospace and Defence Industries Association of Europe).

4.2.1. Standard ASD S3000L (Analyse du Soutien Logistique)

Le standard S3000L structure l'analyse du soutien logistique, incluant l'AMDEC (FMEA). Notre application doit utiliser ce standard pour importer les données initiales. Lors de la mise en service d'un nouvel équipement, les fichiers XML S3000L fournis par le constructeur (contenant l'arborescence physique et les modes de défaillance prévus) sont ingérés pour peupler les bibliothèques d'Objets et de Défauts.[22] L'entité FailureMode du S3000L correspond directement au couple "Objet-Défaut" de MAXER. L'adoption de ce schéma de données (ou d'un mapping strict vers celui-ci) assure que les diagnostics de terrain peuvent être remontés vers l'ingénierie de maintenance pour mettre à jour les plans de maintenance préventive.[24]

4.2.2. Standard MIMOSA OSA-CBM et OSA-EAI

Pour l'intégration future avec la maintenance prévisionnelle (capteurs IoT, vibrations), le système doit exposer des interfaces conformes à MIMOSA OSA-CBM (Open System Architecture for Condition-Based Maintenance). Cela permet à un système de surveillance vibratoire de déclencher automatiquement un diagnostic dans Neo-DIAGDEF en injectant non seulement le symptôme ("Vibration Seuil 2"), mais aussi les données spectrales associées, qui seront attachées comme "Preuves" au nœud racine du défaillogramme.[26]

---

5. La Couche Intelligence Artificielle Cognitive

C'est ici que le projet dépasse la simple numérisation pour entrer dans l'ère de l'assistance cognitive. L'IA n'est pas un gadget, mais le moteur qui rend la méthode MAXER applicable à grande échelle en réduisant la charge cognitive.

5.1. Module RAG (Retrieval-Augmented Generation) pour la Documentation Technique

Les techniciens passent jusqu'à 30% de leur temps à chercher de l'information dans des milliers de pages de documentation PDF. Nous implémenterons un moteur RAG souverain.[29]

- **Architecture :** Un pipeline d'ingestion découpe les manuels constructeurs (format S1000D ou PDF) en segments sémantiques. Ces segments sont vectorisés (embeddings) et stockés dans une base vectorielle (pgvector intégré à PostgreSQL).
- **Fonctionnalité :** Lorsqu'un technicien inspecte un nœud "Pompe Hydraulique", l'IA propose contextuellement les extraits du manuel pertinents (procédures de test, vues éclatées, couples de serrage).
- **Précision :** Contrairement à un ChatGPT générique, le RAG force l'IA à répondre *uniquement* sur la base des documents indexés, citant ses sources (ex: "Voir Manuel Maint. Niv 2, Page 45"), éliminant ainsi les hallucinations dangereuses dans un contexte industriel.[31]

5.2. Génération Automatisée d'Arbres de Défaillances (Cold Start)

L'un des freins à l'adoption de DIAGDEF est l'effort initial pour construire les modèles de diagnostic. L'IA générative peut automatiser cette phase "Cold Start".[32] En analysant les manuels de maintenance et les historiques d'interventions textuels (logs GMAO), un LLM spécialisé peut extraire les relations de causalité et générer des squelettes de défaillogrammes au format JSON de l'application.

- **Processus :** Le système lit le chapitre "Troubleshooting" d'un manuel PDF -> Le LLM identifie "Si symptôme A, alors causes possibles B, C, ou D" -> Le système crée les nœuds et les liens correspondants dans la base graphe.
- **Validation :** Ces arbres générés sont marqués comme "Suggestion IA" et doivent être validés par un expert humain, accélérant la modélisation de facteur 10.

5.3. Agent de Diagnostic Cognitif (LangGraph)

Pour le diagnostic en temps réel, nous déploierons un agent conversationnel avancé basé sur le framework **LangGraph**.[34] Cet agent ne se contente pas de répondre aux questions, il pilote le processus de diagnostic.

- **Architecture d'État :** L'agent maintient un "état" du diagnostic (Symptômes connus, Tests effectués, Hypothèses restantes).
- **Raisonnement Graphique :** Il utilise la base de données graphe (Neo4j) pour calculer les chemins les plus probables. Si le technicien signale "Bruit sur Pompe", l'agent parcourt le

graphe historique pour voir que dans 80% des cas similaires, la cause était "Roulement". Il suggère alors ce test en priorité.
- **Boucle de Réflexion :** Si le test infirme l'hypothèse, l'agent met à jour son graphe de probabilités et propose la prochaine meilleure hypothèse (Next Best Action), agissant comme un mentor expert virtuel à côté du technicien.[36]

5.4. Souveraineté de l'Inférence (Local LLM)

Pour respecter les contraintes de défense, l'IA doit fonctionner sans appel API vers l'extérieur (OpenAI/Anthropic).

- **Moteur d'Inférence :** Utilisation de **vLLM** ou **Ollama** déployé sur des serveurs d'inférence GPU internes.
- **Modèles :** Sélection de modèles "Open Weights" performants comme **Llama 3** ou **Mistral Large**. Ces modèles peuvent être "quantizés" (4-bit) pour réduire l'empreinte mémoire tout en conservant des performances de raisonnement suffisantes pour l'analyse technique.[37]

---

6. Plan de Développement et Roadmap

Le projet est structuré en quatre phases sur 18 mois, adoptant une méthodologie hybride : Agile pour le développement logiciel (Sprints de 2 semaines), mais avec des jalons de validation V-Model pour les modules critiques de sécurité et d'IA.[39]

6.1. Phase 1 : Le Socle (Mois 1-6) - "Digital MAXER"

- **Objectif :** Reconstituer les fonctionnalités core de DIAGDEF avec une techno moderne.
- **Livrables :**
  - Modèle de données relationnel et graphe.
  - Éditeur graphique React Flow (Création/Édition de défaillogrammes).
  - Gestion des bibliothèques Objet/Défaut.
  - Import S3000L basique.
- **Ressources :** 1 Architecte, 2 Devs Backend Python, 2 Devs Frontend React.

6.2. Phase 2 : Intégration et Terrain (Mois 7-10) - "Field Ready"

- **Objectif :** Rendre l'outil utilisable en conditions réelles et connecté.
- **Livrables :**
  - Moteur de synchronisation Offline (PouchDB/CouchDB).
  - Connecteurs GMAO (Webhooks, API).
  - Application Mobile durcie.
  - Sécurité (OIDC, Chiffrement).

6.3. Phase 3 : L'IA Cognitive (Mois 11-15) - "Smart Assistant"

- **Objectif :** Déployer les capacités d'assistance intelligente.
- **Livrables :**
  - Infrastructure RAG (Vector DB + Pipeline d'ingestion).
  - Agent LangGraph de diagnostic.
  - Serveur d'inférence vLLM On-Premise.
  - Génération automatique d'arbres depuis les manuels.

6.4. Phase 4 : Industrialisation et Validation (Mois 16-18)

- **Objectif :** Durcissement et déploiement final.
- **Livrables :**
  - Tests de pénétration et qualification sécurité défense.
  - Optimisation des performances sur gros volumes de données.
  - Formation des utilisateurs et Fine-tuning des modèles IA sur les données réelles.

---

7. Conclusion

Le projet "Neo-DIAGDEF" est bien plus qu'une mise à jour technologique ; c'est une refonte conceptuelle qui place la connaissance structurée au centre de la maintenance. En fusionnant la rigueur sémantique de la méthode MAXER, indispensable pour structurer le chaos des données industrielles, avec la puissance d'analyse et de synthèse de l'IA générative souveraine, nous créons un système vertueux. Chaque panne résolue enrichit le graphe de connaissance, rendant l'IA plus pertinente pour la panne suivante. Ce système répond aux exigences critiques de souveraineté, de traçabilité et d'efficacité opérationnelle, positionnant l'organisation à l'avant-garde de la Maintenance 5.0 : une maintenance résiliente, cognitive et centrée sur l'humain.

(Note : Le rapport complet ci-après détaille les spécifications techniques, les modèles de données, les architectures d'IA et les plans de test nécessaires à la réalisation de cette vision.)

Rapport Technique Détaillé : Spécifications et Architecture pour "Neo-DIAGDEF"

1. Introduction Détaillée et Contexte

1.1. La Problématique du Diagnostic Industriel

L'industrie lourde, la défense et les infrastructures critiques font face à une érosion des compétences techniques ("Brain Drain") due aux départs massifs à la retraite des experts baby-boomers. Le savoir-faire tacite, acquis par des décennies d'expérience sur des machines spécifiques, disparait. Les techniciens juniors, bien que formés aux outils numériques, manquent souvent de la méthodologie de raisonnement logique nécessaire pour diagnostiquer efficacement des pannes complexes sur des systèmes hérités ou hybrides.

Les outils actuels ne comblent pas ce vide :

- **GMAO (SAP, Maximo) :** Ce sont des outils de gestion. Ils enregistrent *que* la machine a été réparée, à quel coût, et avec quelles pièces, mais rarement *comment* le diagnostic a été posé. Les champs de saisie sont souvent du texte libre ("Pompe HS"), inexploitable pour l'analyse statistique ou l'IA.
- **Systèmes Experts (Ancienne génération) :** Souvent rigides, difficiles à mettre à jour, et déconnectés de la réalité opérationnelle dynamique.

1.2. La Solution "Neo-DIAGDEF"

La proposition est de recréer l'outil DIAGDEF, qui a fait ses preuves en implémentant la méthode MAXER, mais en le repensant comme une plateforme native du cloud (cloud-native) déployable on-premise, intégrant les technologies de 2026. L'ambition est de fournir un "GPS du Dépannage" :

1. **Guider :** Fournir le chemin logique le plus court vers la cause racine.
2. **Apprendre :** Transformer chaque intervention en une nouvelle route connue (REX).
3. **Assister :** Utiliser l'IA pour lire les cartes (manuels) et suggérer des itinéraires.

---

2. Analyse Approfondie de la Méthode MAXER et Spécifications Fonctionnelles

La méthode MAXER (Méthode AXée sur l'Amélioration de l'Efficacité du Raisonnement) structure la pensée. Le logiciel doit être le garant de cette structure.

2.1. L'Atome de Donnée : Objet - Défaut

L'erreur fondamentale des systèmes de maintenance classiques est l'utilisation du symptôme vague. MAXER impose de définir le problème par un couple.

- **Spécification Fonctionnelle :**
  - L'utilisateur ne peut pas saisir de texte libre pour le titre du problème.
  - Il doit sélectionner un **Objet** dans l'arborescence (ex: "Moteur M1").
  - Il doit sélectionner un **Défaut** applicable à cet objet dans une liste déroulante filtrée

(ex: "Surchauffe", "Vibration", "Ne démarre pas").
    - Cette liste de défauts doit être administrable et conforme aux standards (ISO 14224, S3000L).
- **Justification IA :** Cette contrainte structurelle est le prérequis absolu pour l'IA. Elle crée des "étiquettes" (labels) propres. Si l'on veut qu'un réseau de neurones prédise une panne, il faut que l'historique d'entraînement soit classifié proprement, pas rempli de "ça marche pas" ou "bruit bizarre".

2.2. Le Raisonnement Graphique : Le Défaillogramme

Le cerveau humain traite mal les listes linéaires de causes possibles. La visualisation en arbre permet de gérer la complexité.

- **Structure du Graphe :**
    - **Racine :** Le Symptôme (Objet + Défaut).
    - **Niveaux 1 à N :** Les Causes (Intermédiaires). Une cause est elle-même un couple Objet-Défaut (ex: "Filtre -> Colmaté").
    - **Feuilles :** Les Causes Premières (Root Causes).
- **Attributs d'État (Coloration Sémantique) :**
    - Chaque nœud doit avoir un état visuel clair.
    - **Blanc :** Hypothèse émise, non vérifiée.
    - **Vert :** Hypothèse vérifiée et écartée (Disculpée).
    - **Rouge :** Hypothèse vérifiée et confirmée (Cause réelle).
    - **Orange :** Hypothèse suspecte, vérification en cours ou impossible.
- **Logique de Flux :** Le logiciel doit permettre de "naviguer" l'arbre. En cliquant sur un nœud, le technicien accède aux "Tests de Vérification" associés (ex: "Mesurer la pression en sortie").

2.3. La Gestion du REX (Retour d'Expérience)

DIAGDEF est avant tout une base de REX.

- **Base de Cas (Case Base) :** Une fois un diagnostic validé, il devient un "Cas de Référence".
- **Recherche par Similitude :** Lorsqu'un nouveau problème survient, le système doit proposer les défaillogrammes existants pour des problèmes similaires (ex: "Ce symptôme sur ce type de pompe a déjà été résolu 3 fois par le passé, voici l'arbre type").
- **Analyse Statistique :** Calcul automatique des taux de défaillance par mode et par composant, alimentant les études de fiabilité (MTBF).

---

3. Architecture Logicielle et Technique

L'architecture doit répondre à des contraintes contradictoires : être riche fonctionnellement

(Graphique, IA) mais légère et robuste (Offline, Mobile).

3.1. Stack Technologique

Couche	Technologie	Rôle et Justification
Client Web / Mobile	**React.js + TypeScript**	Typage fort pour la robustesse. Écosystème riche.
Moteur Graphique	**React Flow**	Bibliothèque open-source performante pour les diagrammes nodaux interactifs. Permet la personnalisation totale des nœuds (Nodes) et des liens (Edges).
Stockage Local	**RxDB (sur IndexedDB)**	Base de données NoSQL réactive côté client. Permet de stocker les graphes JSON et de souscrire aux changements en temps réel.
API Gateway	**FastAPI (Python)**	Haute performance (Asynchrone), documentation automatique (OpenAPI/Swagger), intégration native avec les bibliothèques IA.
Base Graph	**Neo4j**	Leader des bases graphes. Le langage de requête Cypher est idéal pour traverser les arbres de causes. Version Community suffisante pour démarrer.

Base Relationnelle	**PostgreSQL**	Le standard industriel pour les données transactionnelles et vectorielles (via pgvector).
Sync Server	**CouchDB**	Backend de synchronisation pour RxDB. Gère la réplication maître-maître et les conflits offline de manière native.

3.2. Modélisation des Données : Approche Hybride Graph-Document

Le modèle de données est complexe car il doit représenter des structures arborescentes dynamiques.

3.2.1. Le Document "Diagnostic" (Format JSON)

Ce document est stocké dans CouchDB/RxDB pour l'usage offline. Il contient l'état courant d'une session de dépannage.

JSON

```
{
  "_id": "diag_2026_001_A",
  "work_order_ref": "WO-998877",
  "equipment_tag": "P-101-A",
  "status": "IN_PROGRESS",
  "created_at": "2026-05-20T08:00:00Z",
  "technician_id": "TECH_55",
  "symptom": {
    "object_id": "OBJ_PUMP_CENTR",
    "defect_id": "DEF_VIB_HIGH"
  },
  "graph_data": {
    "nodes":,
    "edges": [
      { "id": "e1-2", "source": "1", "target": "2" },
      { "id": "e1-3", "source": "1", "target": "3" }
```

```
  ]
 }
}
```

3.2.2. Le Graphe de Connaissance (Neo4j)

Une fois le diagnostic terminé et synchronisé, il est "éclaté" dans Neo4j pour l'analyse globale.

- **Nœuds :** (:Object), (:Defect), (:Intervention).
- **Relations :** (:Defect)-->(:Defect).
- Cette projection permet à l'IA de répondre à : "Quel est le chemin causal le plus fréquent pour une vibration sur une pompe?" en faisant une agrégation de tous les graphes passés.

3.3. Architecture de Synchronisation Offline-First

Pour les déploiements militaires ou miniers [12] :

1. **Pré-chargement (Provisioning) :** Avant de partir en mission, le terminal du technicien se connecte au serveur. L'API détermine son périmètre (ex: "Zone Nord"). RxDB lance une réplication filtrée pour télécharger uniquement les objets, défauts et historiques de cette zone.
2. **Opération Déconnectée :** Le technicien travaille sur sa copie locale. Les modifications sont empilées dans le journal de modifications de RxDB.
3. **Réconciliation :** Au retour du réseau, RxDB pousse les modifications vers CouchDB.
4. **Gestion de Conflits :** Si le serveur a été modifié entre temps (ex: mise à jour de la bibliothèque de défauts), une stratégie de résolution "Last-Write-Wins" est appliquée par défaut, ou une interface de résolution manuelle est proposée à l'administrateur pour les conflits structurels complexes.

---

4. Intégration GMAO et Standards Industriels

4.1. Standards d'Échange de Données

Pour s'assurer que Neo-DIAGDEF ne soit pas une "boîte noire" propriétaire, nous adopterons les standards ouverts.

4.1.1. S3000L (Logistic Support Analysis)

Le standard S3000L [22] définit comment structurer les données d'analyse du soutien.

- **Intérêt :** Il fournit le vocabulaire standard pour les "Failure Modes" (Modes de défaillance).
- **Implémentation :**
  - L'application sera capable d'importer un fichier d'échange S3000L (format

XML/PLCS) pour initialiser sa base de données d'équipements et de pannes potentielles (FMECA - AMDEC).
    - Cela permet de ne pas partir d'une page blanche : l'application connaît déjà les pannes théoriques définies par le bureau d'études.

4.1.2. MIMOSA OSA-CBM

Le standard MIMOSA [26] est utilisé pour l'interopérabilité avec les systèmes de surveillance (Condition Monitoring).

- **Scénario d'Intégration :** Un système de surveillance vibratoire (CMS) détecte une alarme. Il publie un message XML MIMOSA sur un bus d'entreprise (ESB). Neo-DIAGDEF s'abonne à ce message et crée automatiquement une "Analyse de Situation" avec le bon Objet et le bon Défaut pré-sélectionnés, ainsi que les données de mesure (Amplitude, Fréquence) attachées comme pièces jointes.

4.2. Connecteurs GMAO

L'intégration avec les mastodontes du marché (SAP, Maximo) se fait via une couche d'abstraction.

- **Architecture Hexagonale :** Le cœur de Neo-DIAGDEF est agnostique. Des "Adapters" spécifiques sont développés pour chaque GMAO.
- **Workflows Trigger :**
    - **Création :** Webhook sur l'événement WorkOrder.Created.
    - **Mise à jour :** Si le statut de l'OT change dans la GMAO (ex: "En attente de pièces"), l'info est remontée dans Neo-DIAGDEF pour informer le technicien.
    - **Clôture :** La validation du diagnostic dans Neo-DIAGDEF peut déclencher la clôture technique de l'OT dans la GMAO via API.

---

5. Intelligence Artificielle et Agents Cognitifs

L'ajout d'une couche IA est le différenciateur clé pour moderniser DIAGDEF. Nous utiliserons une approche combinant **IA Symbolique** (Graphes de Connaissance) et **IA Connexionniste** (LLM/Deep Learning).

5.1. Génération Automatique d'Arbres de Défaillances (LLM + RAG)

Le "Cold Start" (démarrage à froid) est le problème n°1 : remplir la base de données demande des milliers d'heures d'experts.

- **Solution :** Pipeline d'extraction de connaissances à partir de documents non structurés (PDF Manuels, Notes techniques).
- **Technologie :**
    - **Ingestion :** Utilisation de unstructured (librairie Python) pour parser les PDF

complexes (tableaux, schémas).
    - **Extraction :** Utilisation d'un LLM puissant (Llama 3 70B ou GPT-4o si cloud autorisé) avec un prompt "Few-Shot" (exemples fournis) pour extraire les triplets (Cause) -> -> (Effet).
    - **Validation :** Les triplets extraits sont proposés sous forme de graphe provisoire à l'expert humain pour validation.
    - **Résultat :** Réduction de 90% du temps de modélisation initiale.[32]

5.2. L'Agent de Diagnostic "Compagnon" (LangGraph)

Pour assister le technicien en temps réel, nous concevons un agent autonome basé sur **LangGraph**.[34] Contrairement à un chatbot linéaire, LangGraph permet de définir des cycles de raisonnement et des prises de décision.

5.2.1. Architecture de l'Agent (State Machine)

L'agent est modélisé comme un graphe d'états :

1. **Node "Perception" :** Analyse l'entrée utilisateur ("Ça fume blanc"). Utilise le RAG pour comprendre le contexte technique.
2. **Node "Consultation Graphe" :** Interroge la base Neo4j. "Quelles sont les causes de fumée blanche sur ce moteur diesel dans l'historique?"
3. **Node "Raisonnement" :** Le LLM synthétise les infos RAG (Théorie) et Neo4j (Pratique/REX). Il détermine qu'il y a 2 hypothèses probables : "Joint de culasse" (Grave, fréquent) ou "Condensation" (Bénin, fréquent à froid).
4. **Node "Action" :** L'agent pose une question discriminante : "Le moteur est-il froid ou chaud?".
5. **Node "Mise à jour" :** En fonction de la réponse, l'agent met à jour les probabilités des branches du défaillogramme affiché à l'écran.[36]

5.3. Infrastructure d'Inférence IA Souveraine

Pour les clients défense, les données ne doivent pas sortir.

- **Serveur d'Inférence Local :** Utilisation de **vLLM** [37] déployé dans un conteneur Docker avec accès GPU (NVIDIA Container Toolkit).
- **Modèles Quantizés :** Utilisation de modèles au format **GGUF** ou **AWQ** (ex: Llama-3-8B-Instruct-Q5_K_M.gguf) qui peuvent tourner sur des GPU grand public ou même des CPU puissants, garantissant que l'IA fonctionne même sur un serveur tactique modeste dans un camion de commandement.[40]

---

6. Sécurité, Sûreté et Conformité

6.1. Stratégie de Sécurité "Defense-in-Depth"

- **Isolation Réseau (Air-Gap) :** L'application est conçue pour fonctionner sans aucun accès Internet. Les mises à jour de modèles IA ou de binaires se font par clé USB sécurisée ou passerelle unidirectionnelle (Data Diode).
- **Chiffrement de Bout en Bout :**
  - TLS 1.3 avec Mutual Auth (mTLS) pour les communications Client-Serveur.
  - Chiffrement des bases de données au repos (PostgreSQL TDE, CouchDB encryption).
- **Contrôle d'Accès Basé sur les Rôles (RBAC) :** Matrice fine de droits.
  - *Technicien :* Peut créer/modifier ses propres diagnostics.
  - *Expert Méthode :* Peut valider les REX et modifier les modèles de référence.
  - *Admin :* Gestion des utilisateurs et de la configuration système.

6.2. Auditabilité

Chaque action dans le système (surtout les modifications de statuts de pannes ou les validations de réparations) génère une entrée de log signée cryptographiquement. Cela permet, en cas d'accident industriel ultérieur, de reconstruire exactement quel raisonnement a été tenu, quelles vérifications ont été faites (ou omises), et qui a validé la remise en service.

---

7. Plan de Projet et Allocation des Ressources

7.1. Phases de Développement (18 Mois)

Phase	Durée	Objectifs Clés	Livrables Majeurs
1. Conception & Core	Mois 1-4	Socle technique, Modèle de données, UX/UI React Flow.	MVP "Digital MAXER" (saisie manuelle graphes).
2. Intégration & Mobile	Mois 5-9	Offline-first, Synchro CouchDB, Connecteurs API.	App Mobile Tablette, Connecteur SAP.
3. IA Cognitive	Mois 10-14	RAG, Agent LangGraph, Génération d'arbres.	Module "AI Assistant", Ingestion PDF.
4. Industrialisation	Mois 15-18	Tests pénétration, Optimisation,	Version 1.0 "Defense Ready".

		Documentation S1000D.	

7.2. Équipe Type (Squad Agile)

- **1 Lead Tech / Architecte :** Garant de la vision technique et des choix d'infrastructure.
- **2 Développeurs Backend (Python) :** API, Logique MAXER, Intégration IA (LangChain/Neo4j).
- **2 Développeurs Frontend (React) :** Interface graphique complexe, gestion Offline (RxDB).
- **1 Ingénieur DevOps / MLOps :** CI/CD, Conteneurisation, Déploiement GPU, Sécurité.
- **1 Product Owner (Expert Maintenance) :** Garant de la fidélité à la méthode MAXER et des besoins métier.

---

8. Conclusion

Le projet "Neo-DIAGDEF" est une réponse technologique ambitieuse à un problème industriel critique. En ne se contentant pas de numériser une méthode papier mais en l'augmentant par l'IA et l'architecture distribuée, il offre une solution pérenne au défi de la maintenance des systèmes complexes.

La combinaison de l'approche systémique MAXER (pour la rigueur) et de l'IA Générative (pour la vélocité et l'assistance) crée une synergie unique : l'IA apprend de la rigueur humaine, et l'humain est augmenté par la mémoire synthétique de l'IA.

Ce projet est techniquement réalisable avec les briques Open Source actuelles (React Flow, LangGraph, Neo4j, vLLM) et présente un potentiel de ROI considérable par la réduction des arrêts de production et la préservation du capital de connaissances de l'entreprise.

Annexes Techniques

- *Annexe A : Schéma détaillé du modèle de données Graph (Nodes/Relationships).*
- *Annexe B : Spécification de l'API REST v1 (OpenAPI Spec).*
- *Annexe C : Architecture réseau pour déploiement en zone militaire (DMZ, Proxy).*

Works cited

1. Logiciel Embarqué : Fonctionnalités Clés, Architecture et Domaines d'Application - Syslearn, accessed January 22, 2026, https://syslearn.fr/logiciel-embarque-fonctionnalite-domaine-dapplication/
2. Logiciel défaillances industrielles | REX | Diagdef | SIROM, accessed January 22, 2026, https://www.sirom.fr/logiciel-defaillances-industrielles-rex/

3. Méthode Maxer - Wikipédia, accessed January 22, 2026, https://fr.wikipedia.org/wiki/M%C3%A9thode_Maxer
4. Livre Blanc Informatique - Sigmaxer, accessed January 22, 2026, https://www.sigmaxer.fr/box/Livre-Blanc-Informatique_VF.pdf
5. Planning Informatisé pour Maintenance Hospitalière | PDF | Informatique | Feuille de calcul - Scribd, accessed January 22, 2026, https://fr.scribd.com/document/497741812/Rapport-stage-Creation-outil-de-planification-de-maintenance
6. Processus de maintenance dans l'industrie pharmaceutique - Sigmaxer, accessed January 22, 2026, https://www.sigmaxer.fr/box/Maintenance_Pharmaceutique.pdf
7. en améliorant l'efficacité des Techniciens en Maintenance avec la Méthode MAXER+ - Sigmaxer, accessed January 22, 2026, https://www.sigmaxer.fr/box/S.Consultants_Disponibilite_MAXER.pdf
8. Methode de Maxer | PDF - Scribd, accessed January 22, 2026, https://fr.scribd.com/document/495622014/Methode-de-Maxer
9. Why offline-first apps are gaining importance - Locize, accessed January 22, 2026, https://www.locize.com/blog/offline-first-apps
10. What's the Difference Between Offline-First and Online-First App Design?, accessed January 22, 2026, https://thisisglance.com/learning-centre/whats-the-difference-between-offline-first-and-online-first-app-design
11. AI-Network-Troubleshooting-PoC/llm_agent/app.py at main - GitHub, accessed January 22, 2026, https://github.com/jillesca/AI-Network-Troubleshooting-PoC/blob/main/llm_agent/app.py
12. Offline-First Architecture: Designing for Reality, Not Just the Cloud | by Jusuf Topic | Medium, accessed January 22, 2026, https://medium.com/@jusuftopic/offline-first-architecture-designing-for-reality-not-just-the-cloud-e5fd18e50a79
13. Offline-First Application - Design and Architecture - BigThinkCode, accessed January 22, 2026, https://www.bigthinkcode.com/insights/offline-first-application
14. Regroupement d'experts de la fiabilité des systèmes et des composants électroniques, accessed January 22, 2026, https://www.cff-fiabilite.fr/wp-content/uploads/sites/12/2023/03/Book_CFF_2023_WEB.pdf
15. DU BLAVET - Gest'eau, accessed January 22, 2026, https://www.gesteau.fr/sites/default/files/doc_SAGE04007-1207661227.pdf
16. On-Premise LLM Deployment: Why Enterprises Need AI Inside Their Firewall - AIVeda, accessed January 22, 2026, https://aiveda.io/blog/on-premise-llm-deployment-why-enterprises-need-ai-inside-their-firewall
17. How On-Premise LLMs Deliver Absolute Data Sovereignty and Compliance for Privacy-First Enterprises - WeblineIndia, accessed January 22, 2026, https://www.weblineindia.com/blog/on-premise-llm-secure-compliant-ai/
18. Real-time Automations with Cityworks Webhooks - FME Support Center,

accessed January 22, 2026, https://support.safe.com/hc/en-us/articles/25407532065037-Real-time-Automations-with-Cityworks-Webhooks
19. Webhook only trigger when job status changes to work order? - ServiceM8 Developer Portal, accessed January 22, 2026, https://developer.servicem8.com/discuss/67e0fa0f8c21e000455b6718
20. TPM, MAXER et vous qui ... - club des ingenieurs de maintenance, accessed January 22, 2026, https://clubdesingenieursdemaintenance.blogspot.com/2016/09/tpm-maxer-et-vous-qui-recherchez-la.html
21. Maintenance 4.0 ? Des conditions de réussite, accessed January 22, 2026, https://clubdesingenieursdemaintenance.blogspot.com/2017/02/maintenance-40-des-conditions-de.html
22. SLICwave Life Cycle - Integrated Support Systems Inc., accessed January 22, 2026, https://www.isscorp.com/slicwave-life-cycle.html
23. S3000L, accessed January 22, 2026, https://old.cals.ru/collaboration/S3000L/doc/S3000L_Introduction_with_data_part.pdf
24. S3000L: The integrated logistic support standard - ADAM - 4D concept, accessed January 22, 2026, https://adam.4dconcept.fr/s3000l-the-integrated-logistic-support-standard/?lang=en
25. FMEA/FMECA process in the S3000L. | Download Scientific Diagram - ResearchGate, accessed January 22, 2026, https://www.researchgate.net/figure/FMEA-FMECA-process-in-the-S3000L_fig6_269274241
26. MIMOSA OSA-CBM, accessed January 22, 2026, https://www.mimosa.org/mimosa-osa-cbm/
27. Distributed Embedded Condition Monitoring Systems based on OSA-CBM Standard - CERES Research Repository, accessed January 22, 2026, https://dspace.lib.cranfield.ac.uk/bitstreams/5d48cd7d-f16c-4776-9c06-5f9dff0c68f7/download
28. (PDF) Standardization Issues in Condition-Based Maintenance - ResearchGate, accessed January 22, 2026, https://www.researchgate.net/publication/240262171_Standardization_Issues_in_Condition-Based_Maintenance
29. RAG Production: The Complete Guide - Kairntech, accessed January 22, 2026, https://kairntech.com/blog/articles/rag-production-the-complete-guide-to-building-and-deploying-retrieval-augmented-generation-applications/
30. Industrial RAG: cut MTTR and downtime costs with AI - Baobab Soluciones, accessed January 22, 2026, https://baobabsoluciones.es/en/noticias/industry/mttr-and-downtime-costs-the-rag-model-that-turns-manuals-into-actionable-knowledge/
31. Prescriptive Agents based on RAG for Automated Maintenance (PARAM) - arXiv, accessed January 22, 2026, https://arxiv.org/html/2508.04714v2

32. Generating Troubleshooting Trees for Industrial Equipment using Large Language Models (LLM) - IEEE Xplore, accessed January 22, 2026, https://ieeexplore.ieee.org/document/10626823/
33. Generating Troubleshooting Trees for Industrial Equipment using Large Language Models (LLM) | Request PDF - ResearchGate, accessed January 22, 2026, https://www.researchgate.net/publication/383108809_Generating_Troubleshooting_Trees_for_Industrial_Equipment_using_Large_Language_Models_LLM
34. Building AI Workflows with LangGraph: Practical Use Cases and Examples - Scalable Path, accessed January 22, 2026, https://www.scalablepath.com/machine-learning/langgraph
35. Thinking in LangGraph - Docs by LangChain, accessed January 22, 2026, https://docs.langchain.com/oss/python/langgraph/thinking-in-langgraph
36. LangGraph: Building Self-Correcting RAG Agent for Code Generation, accessed January 22, 2026, https://learnopencv.com/langgraph-self-correcting-agent-code-generation/
37. vLLM or llama.cpp: Choosing the right LLM inference engine for your use case, accessed January 22, 2026, https://developers.redhat.com/articles/2025/09/30/vllm-or-llamacpp-choosing-right-llm-inference-engine-your-use-case
38. Running Local LLMs with Ollama: 3 Levels from Laptop to Cluster-Scale Distributed Inference - BentoML, accessed January 22, 2026, https://www.bentoml.com/blog/running-local-llms-with-ollama-3-levels-from-local-to-distributed-inference
39. Agile V-Model - digitalplaybook.org, accessed January 22, 2026, https://aiotplaybook.org/index.php?title=Agile_V-Model
40. Local LLM inference - by Amir Zohrenejad - Medium, accessed January 22, 2026, https://medium.com/@aazo11/local-llm-inference-897a06cc17a2