# Architecture d'IA Hybride Auto-Programmable: Le Modèle Maître d'Œuvre et Compagnons Experts Industrialisé

## 1. Le Principe de l'Hybridation

Notre modèle repose sur la dissociation fonctionnelle et l'optimisation des ressources :

- **Le Maître d'Œuvre (Stratégie):** LLM Cloud (Gemini, GPT-4). Se concentre sur la décomposition des missions et l'orchestration (raisonnement le plus complexe).
- **Les Compagnons Experts (Tactique):** Agents locaux (Llama 3, Qwen). Exécutent les tâches concrètes (codage, audit, gestion de système) sur l'infrastructure Promethee.

Cette approche, gérée par **LangGraph**, assure la maîtrise des coûts et la souveraineté.

---

## 2. Les Piliers de l'Architecture Cognitive et d'Orchestration

### A. L'Orchestration par Graphe (LangGraph)

Le moteur d'orchestration utilise LangGraph pour : 1. **Contrôle de Flux :** Définir des workflows complexes et conditionnels (`check_queue.py` pour le routage). 2. **Persistance :** Gérer l'état de mission complet (`ToolingFlowState`) de manière sérialisable et persistante. 3. **Modularité :** Chaque étape de raisonnement est un Nœud indépendant, facilitant la modification et l'ajout de nouvelles compétences.

### B. Le Cerveau Hybride (Double Mémoire)

Le système évite les hallucinations en s'ancrant dans : * La **Mémoire Sémantique** (Weaviate). * La **Mémoire Logique** (Neo4j) pour le raisonnement structurel (ex: liens de dépendance du code généré).

---

## 3. Stratégie RAG Avancée : Compréhension Structurelle du Code (R&D)

Cette section décrit une **prochaine évolution majeure** visant à améliorer la performance du RAG sur des bases de code volumineuses. **Cette fonctionnalité est actuellement en phase de recherche et de conception architecturale (R&D) et non implémentée.**

L'objectif est de surmonter le **"Problème de l'Échelle"** et d'éviter la saturation du contexte lors du travail sur l'architecture du projet lui-même.

### A. L'Intelligence Code par Graphe (AST & Call Graph)

La Mémoire Logique (Neo4j) sera enrichie par la représentation abstraite du code source : * **Analyse AST (Abstract Syntax Tree) :** Le code est d'abord transformé en un arbre syntaxique. * **Création du Call Graph :** Les relations entre les fonctions, les classes, et les fichiers (dépendances, héritages, appels) sont extraites de cet AST et stockées dans Neo4j. * **RAG Structurel :** Lorsqu'un agent a besoin de modifier une fonction, il interroge Neo4j pour obtenir le **"squelette structurel"** (le *Call Graph*) et les **interfaces critiques** des fonctions adjacentes, réduisant drastiquement le bruit du contexte et améliorant la précision du raisonnement sur l'impact de la modification.

### B. Squelettisation du Contexte

Cette approche permettra au système de naviguer dans l'architecture par étapes : 1. **Vue d'Oiseau :** L'agent obtiendra un aperçu de l'arborescence enrichie de résumés de fichiers. 2. **Vue Détaillée :** L'agent sélectionnera les nœuds du graphe pertinents, puis demandera uniquement le code source des fonctions spécifiques pour le contexte de génération.

---

## 4. Maîtrise et Audit du Système (Implémenté)

L'autonomie est contrebalancée par une auditabilité totale :

- **Logging de Nœud Granulaire :** Le système de `node_logger.py` assure une trace complète de l'exécution de l'IA (Input/Output/Erreurs/Durée) pour chaque étape du graphe.
- **Quarantaine :** Le code généré est systématiquement placé dans un état de quarantaine (`quarantined_tools` dans le state) avant toute exécution sur la plateforme publique Pollux, nécessitant une validation humaine finale.