

# C.L.A.R.A.

LLM'S AGENTIC AND BIOMEDICAL - ING3 SCIA/Santé

## Membres du groupe:



aglae.tournois



andy.shan



lois.breant



oscar.le-dauphin

# Introduction & Objectifs

---

**Problématique :** Comment concevoir un agent intelligent capable d'analyser des images médicales de manière autonome et explicable ?

**Objectif :** Créer un agent capable de :

- **Planifier** une analyse.
- **Utiliser des outils pertinants** pour répondre aux attentes du patient.
- **Mémoriser et contextualiser** ses actions.
- **Minimiser les coûts** d'utilisation des LLMs.

*“L’avenir de l’IA réside dans des systèmes capables de raisonner, de planifier et d’apprendre des modèles du monde.”*

- **Yann LeCun** (Objective-Driven AI)

# Architecture du Système

- **Backend (FastAPI)** : Orchestration des agents (Planner/Executor/Reactive).
- **Frontend (React/Vite)** : Visualisation temps réel du graphe de raisonnement.

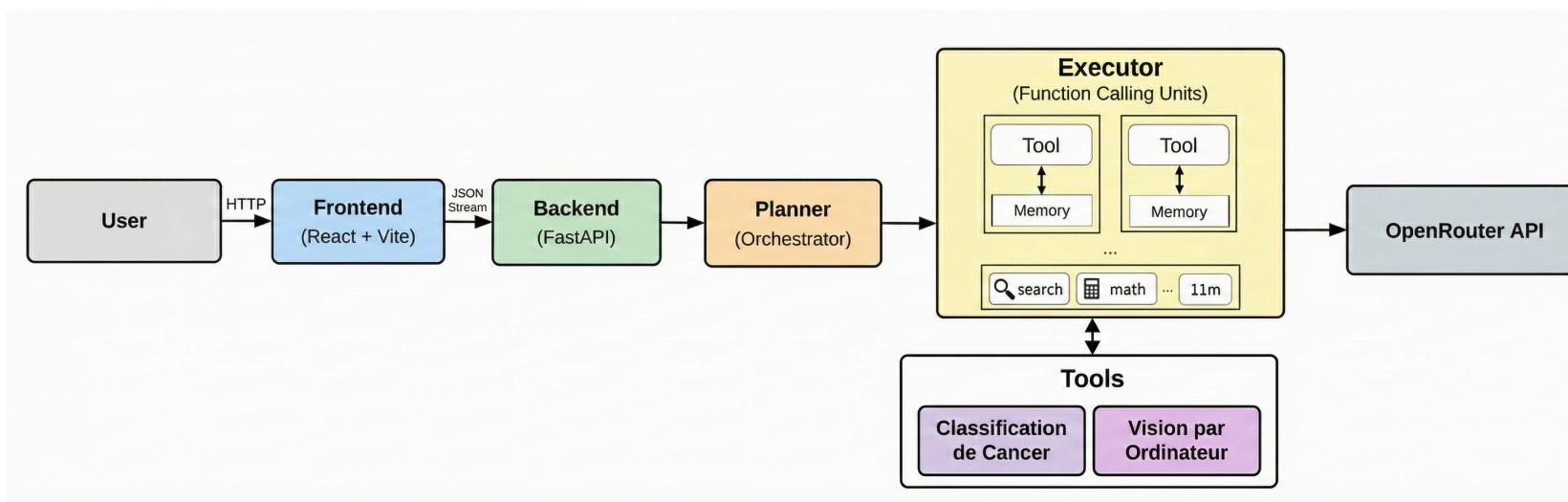


Figure 1 : Architecture Globale

# Motifs Agentiques

---

## 1. Planner-Executor

- **Pourquoi ?** L'analyse médicale est procédurale.
- **Comment ?** Le Planner décompose les actions, l'Executor agit.

## 2. Mémoire (Contextuelle)

- **Pourquoi ?** Partager les résultats entre les tâches.
- **Comment ?** Résolution de variables (`$step_id`) et passage de contexte.

**3. Outils & Délégation** L'Executor délègue les tâches perceptives :

- **Outil Classification (CNN)**
  - **Input** : Image (Mammographie/Scan).
  - **Output** : Label (Bénin/Malin) + Confiance.
- **Agent Vision (VLM)**
  - **Input** : Image + Instructions.
  - **Output** : Description textuelle et analyse visuelle.

# Stack Technique & Streaming

---

## Backend :

- Python 3.13, FastAPI.
- Modèle google/gemma-3-27b-it (Vision & Texte).
- Client openrouter.

## Frontend :

- React 19, Tailwind CSS v4.
- Graphe dynamique (Mermaid/Recharts).

## Streaming de réponse en temps réel :

Flux continu JSON (AgentResponse). L'utilisateur voit l'agent “réfléchir” et peut consulter chaque étape de la plannification.

# Monitoring & Métriques

---

**Approche :** Monitoring temps réel de la performance des agents via une instrumentation dédiée.

## Métriques loggées :

- **Latence** : Temps d'exécution par étape (Planner/Executor/Vision).
- **Consommation** : Comptage précis des tokens (Input/Output) pour l'analyse des coûts.

## Instrumentation :

- Export automatique dans `telemetrics.csv`.
- Permet d'identifier les goulots d'étranglement.

# Coûts & Scalabilité

---

## Modèle Économique :

- **Stratégie** : Utilisation de modèles “Free Tier” (Gemma, Llama) via OpenRouter pour le développement.
- **Coût réel** : Quasi-nul pour le prototype.

**Suivi des coûts** : Chaque appel API est loggé avec :

- Modèle utilisé.
- Tokens (Prompt & Complétion).
- Latence.

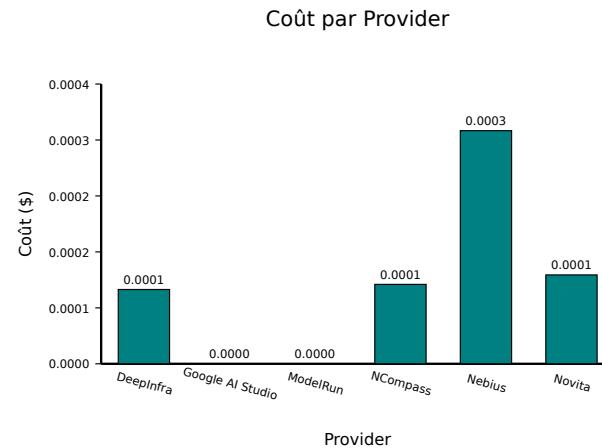


Figure 2 : Simulation des Coûts

## Optimisation :

Le Planner est léger (texte), la Vision est le poste principal de consommation (images).

# Conclusion

---

## Apports du projet :

- Architecture agentique fonctionnelle (Planner/Executor).
- Transparence accrue pour le praticien (Graphe de pensée).
- Démonstration de la viabilité des modèles Open Source.

## Perspectives :

- **Latence** : Optimiser le streaming et paralléliser.
- **Robustesse** : Ajouter un agent “Critic” pour valider les diagnostics.
- **Mémoire** : Intégrer un vrai RAG pour l'historique médical.

# Démo Time