

# C.L.A.R.A.

Agentification & Analyse d'Images Médicales

## Membres du groupe:



aglae.tournois



andy.shan



lois.breant



oscar.le-dauphin

# Introduction & Objectifs

---

**Contexte :** Module “Agentification” - SCIA/Santé 3ème Année.

**Problématique :** Comment transformer une analyse d’image médicale “boîte noire” en un processus transparent, planifié et vérifiable ?

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

- **Planifier** son analyse (qualité, zone, anomalies).
- **Utiliser des outils** (Vision, Classification).
- **Contextualiser** ses actions (Mémoire à court terme).

*“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

---

## Approche Modulaire :

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

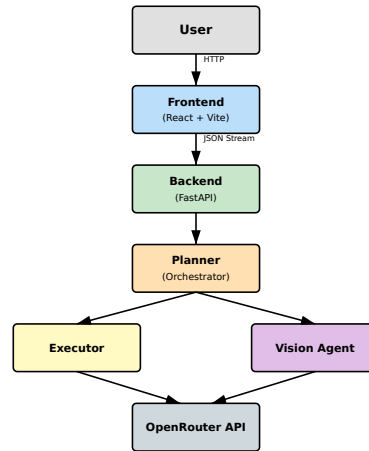


Figure 1 : Architecture Globale et Flux de Données

# Motifs Agentiques Clés

---

## 1. Planner-Executor

- **Pourquoi ?** L'analyse médicale est procédurale.
- **Comment ?** Le Planner décompose (ex: "Vérifier radio"), 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

- **Pourquoi ?** Utiliser des experts pour chaque sous-tâche.
- **Comment ?** L'Executor délègue à `vision_tool` ou `classification_tool` selon le plan.

**Note:** Ces motifs augmentent la traçabilité et la robustesse.

# Stack Technique & Streaming

---

## Backend :

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

## Frontend :

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

## Streaming Temps Réel :

Flux continu JSON (AgentResponse). L'utilisateur voit l'agent "réfléchir" étape par étape, améliorant la confiance utilisateur.

# Évaluation & Métriques

---

**Méthodologie** : Comparaison sur 50 cas cliniques variés (Fractures, Pneumonies, Normal).

## **Métriques suivies (via Telemetrics):**

- **Taux de Succès** : Pertinence du diagnostic final vs Ground Truth.
- **Latence** : Temps total de traitement (Planner + Execution).
- **Consommation** : Nombre de tokens (Input/Output) par étape.

## **Instrumentation :**

- Logging automatique dans `telemetrics.csv`.
- Suivi détaillé par `session_id` et `agent_id`.

# Coûts & Scalabilité

---

## Modèle Économique :

- **Stratégie Actuelle** : 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 + Completion).
- Latence.

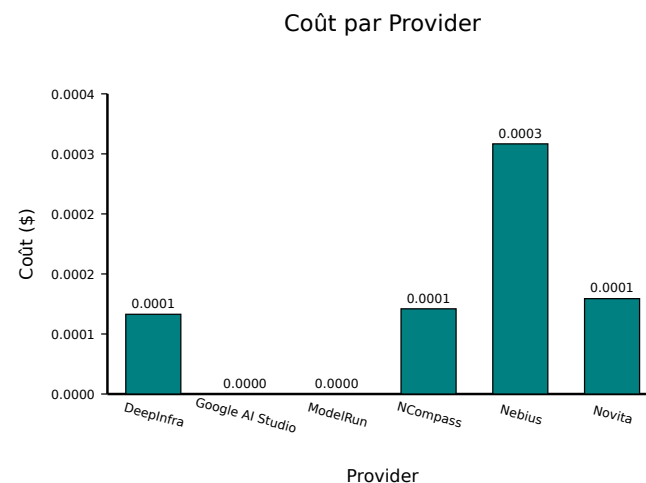


Figure 2 : Simulation des Coûts

## Optimisation :

Le Planner consomme peu (contexte court),  
l'Executor (Vision) est le poste principal.

# 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** 🔥