# Au cœur de l'architecture : Comment sont structurées les applications d'IA ?

*Hernandez Kenny | BTS SIO – Option SLAM | Date : 05/10/2025*

#VeilleTechnologique #MLOps #Architecture #DeepLearning #SLAM

## 1. Introduction

L'Intelligence Artificielle (IA) change radicalement la structure de nos applications. Pour un développeur SLAM, ce n'est pas un concept abstrait : nous passons d'un code basé sur des règles (if/else) à une logique basée sur des « modèles » entraînés à partir de données.

Cet article de veille s'attaque à une question d'architecture logicielle : concrètement, comment sont bâties les applications modernes de Deep Learning ? Quel est le « pattern » de conception d'un logiciel capable d'analyser des images ou du texte ?

Ce sujet est au cœur de notre métier. L'enjeu technique est de comprendre que ces logiciels ne sont pas monolithiques. L'enjeu applicatif est de maîtriser les nouvelles « pipelines MLOps ». Pour illustrer cette architecture, je prendrai comme support un exemple exigeant : **l'imagerie médicale et la détection de pathologies**.

> On peut distinguer deux grandes parties dans ce type d'architecture logicielle :
>
> 1. **L'Usine d'Entraînement** : La partie « Développement », où l'on programme et entraîne le modèle.

2. **L'API d'Inférence** : La partie « Production », où le modèle (un simple fichier) est utilisé pour faire des prédictions en temps réel.

## 2. Démarche de veille (méthode et outils)

Pour analyser cette architecture, j'ai mis en place une veille ciblée sur les outils et méthodes de développement.

### Outils utilisés

- **Feedly** (agrégateur de flux RSS) : centralisation des blogs techniques des géants du secteur (Google AI, NVIDIA, Microsoft) et des documentations des frameworks clés (TensorFlow, PyTorch, FastAPI).
- **daily.dev** (agrégateur de contenu) : surveillance des articles et tutoriels populaires dans la communauté des développeurs (#mlops, #python, #fastapi).

### Critères de sélection

- Sources de moins de 18 mois, privilégiant la fiabilité et la pertinence pour un développeur.
- Documentation technique des frameworks et blogs d'ingénieurs spécialisés en MLOps.
- Priorité aux retours d'expérience (REX) d'entreprises ayant mis en production ce type d'architecture.

## 3. Analyse des tendances et innovations

La structure d'un logiciel d'IA repose sur la **séparation nette entre la création du modèle et son utilisation**.

| 🏭 L'Usine d'Entraînement | 🔌 L'API d'Inférence |
| --- | --- |
| Partie *Développement* — fabrication du | Partie *Production* — utilisation du |

| | |
|---|---|
| modèle à partir des données. | modèle pour prédire en temps réel. |

## Tendance 1 — La structure de « l'Usine d'Entraînement »

C'est là que le modèle est « fabriqué ». Il s'agit d'une véritable chaîne de montage logicielle (une pipeline) qui assemble plusieurs composants :

- **Composant 1 — Préparation des données.** Des scripts Python et OpenCV récupèrent les données brutes (ex. : connexion à un serveur d'images médicales DICOM/PACS), les nettoient et les transforment pour que le modèle puisse les comprendre.
- **Composant 2 — Programmation du modèle.** Le développeur utilise un framework (TensorFlow ou PyTorch) pour définir l'architecture du réseau de neurones. Pour l'exemple médical, on utiliserait une architecture *U-Net*, réputée pour sa capacité à « segmenter » une image (délimiter précisément une tumeur).
- **Composant 3 — Suivi des essais.** Des outils comme MLflow enregistrent automatiquement les performances, les versions des modèles et les paramètres de chaque essai.

> Le produit final de cette usine n'est pas un exécutable, mais un simple **fichier artefact** (ex. : `model.h5`) contenant les millions de « poids » du réseau entraîné.

## Tendance 2 — La structure de « l'API d'Inférence »

C'est l'environnement de production. Le fichier `model.h5` est inutile seul. Pour qu'une application l'utilise, la tendance est de l'encapsuler dans un **microservice web**.

- **Composant 1 — Le « Wrapper » d'API.** C'est une tâche 100 % SLAM. Le développeur utilise un micro-framework web Python (FastAPI ou Flask) pour créer un serveur qui charge le modèle en mémoire, expose un endpoint REST (ex. : `POST /predict`), reçoit les données d'entrée (une image), appelle le modèle (`model.predict()`) et renvoie la prédiction en JSON.

- **Composant 2 — Le « Moteur de Service » (optionnel).** Pour des besoins de très haute performance (voiture autonome), des serveurs spécialisés comme NVIDIA Triton optimisent l'exécution des modèles sur GPU à très haute vitesse.

Cette structure en API est la clé : elle **découple l'application métier** (le logiciel de l'hôpital) de la complexité de l'IA. L'application se contente d'appeler une API REST.

### Tendance 3 — Le MLOps ou la fusion des structures

La dernière tendance est de connecter ces deux structures (Usine et API) via une **pipeline CI/CD**. C'est le MLOps. La chaîne logique est la suivante : un développeur pousse du code → un outil de CI (ex. : GitLab CI) lance automatiquement l'entraînement → le nouveau modèle est testé → s'il est validé, l'outil de CD construit l'image Docker de l'API avec ce nouveau modèle et la déploie.

### Impact sur le métier de développeur SLAM

- **Python devient le langage « full-stack »** : utilisé pour les données, le modèle, l'API web et le pipeline CI/CD.
- **L'API REST est notre point de liaison.** Nos compétences en API (vues en BTS) deviennent le pont indispensable entre la Data Science et les applications métier.
- **Du DevOps au MLOps.** On ne gère plus seulement le cycle de vie du code, mais aussi celui des données et des modèles, bien plus complexe à versionner.

## 4. Applications pratiques et exemples concrets

### Le framework MONAI de NVIDIA

Le projet open-source **MONAI** (Medical Open Network for AI) est un framework permettant de construire une Usine d'Entraînement pour l'imagerie médicale. Il fournit des composants prêts à l'emploi pour lire les formats médicaux (DICOM) et des architectures de modèles (U-Net). En tant que développeur SLAM, on utilise MONAI pour assembler un pipeline d'entraînement plus rapidement.

### Mini cas pratique : Architecture d'un service de triage de biopsies

**Contexte :** Un laboratoire veut prioriser les biopsies les plus suspectes pour une analyse médicale.

- **Structure 1 (Usine) :** Une équipe entraîne un modèle sur des milliers d'images de biopsies. Le résultat est le fichier `biopsy_model.v1.h5`.
- **Structure 2 (API) :** Un développeur SLAM code une API REST avec FastAPI. Le logiciel du laboratoire envoie l'image à l'endpoint `POST /triage` et reçoit en réponse un JSON : `{"id_lame": "XYZ", "status": "Suspect", "score": 0.94}`.

> **La force du découplage :** le logiciel du labo ne sait pas comment fonctionne l'IA, il consomme simplement un service web. L'équipe IA peut redéployer l'API avec un `v2.h5` sans que le logiciel du labo ne s'en aperçoive.

## 5. Analyse critique

### Limites et points de vigilance techniques

- **Complexité :** cette architecture à deux têtes (Usine/API) plus le MLOps est bien plus compliquée à maintenir qu'une application web classique.
- **Vieillissement du modèle (Data Drift) :** si le monde réel change (ex. : un nouveau scanner à l'hôpital), le modèle devient « périmé » sans que le code de l'API ne bouge. Un monitoring constant des performances est indispensable.
- **Sécurité de l'API :** l'endpoint `/predict` est une nouvelle porte d'entrée pour des « attaques adverses » (images modifiées pour tromper l'IA). Sa sécurisation est critique.

### Analyse concurrentielle

| Approche | Exemples | Forces | Faiblesses |
|---|---|---|---|
| **Solutions propriétaires** | | Certification médicale | Coût élevé, personnalisation |

| Approche | Exemples | Forces | Faiblesses |
|---|---|---|---|
| | Philips, Siemens, PathAI | incluse, support, intégration facile | impossible, dépendance fournisseur |
| **Frameworks Open-Source** | MONAI, TensorFlow, PyTorch | Gratuit, transparent, personnalisable | Coût humain (équipe d'experts), maintenance et certifications à la charge de l'entreprise |

## 6. Synthèse et conclusion

Programmer une IA, ce n'est pas coder un algorithme magique. C'est concevoir une architecture logicielle en deux parties : une « **usine** » pour l'entraînement et une « **API** » pour la production. Le lien entre les deux est le **MLOps**, l'évolution logique de nos pratiques DevOps.

Pour le développeur SLAM, l'impact est clair : notre rôle grandit. Nous devenons les architectes qui conçoivent ces API robustes et les intégrateurs qui automatisent tout le cycle de vie du modèle. L'imagerie médicale, avec ses contraintes fortes (RGPD, HDS), est un excellent exemple qui nous pousse à adopter les structures les plus solides.

> **Perspective :** La prochaine étape sera la **multimodalité** — des API qui n'accepteront plus seulement une image, mais une combinaison d'entrées (image, compte-rendu texte, données génomiques) pour produire une seule prédiction. Un défi d'architecture passionnant pour un développeur SLAM.

## 7. Sources et références


- NVIDIA Blogs. (14/03/2024). *What Is Federated Learning?* blogs.nvidia.com. Consulté le 21/10/2025.
- MONAI Project. (2024). *MONAI : Medical Open Network for AI.* monai.io. Consulté le 12/10/2025.

- Haute Autorité de Santé (HAS). (12/07/2024). *IA en santé : guide pour l'évaluation des algorithmes d'IA en imagerie médicale.* has-sante.fr. Consulté le 18/10/2025.
- FastAPI. (2024). *Documentation officielle.* fastapi.tiangolo.com. Consulté le 05/10/2025.
- MLflow. (2024). *An open source platform for the machine learning lifecycle.* mlflow.org. Consulté le 20/10/2025.
- TensorFlow. (2024). *Documentation officielle TensorFlow.* tensorflow.org. Consulté le 09/10/2025.
- PyTorch. (2024). *Documentation officielle PyTorch.* pytorch.org. Consulté le 14/10/2025.
- Microsoft Azure Blog. (22/02/2024). *Qu'est-ce que MLOps (Machine Learning Operations) ?* azure.microsoft.com. Consulté le 17/10/2025.