# RAPPORT TECHNIQUE - PROJET P10 DSML

**Nom : Abdourahamane LY**

## 1. Introduction

Ce projet s'inscrit dans le cadre du parcours **Data Scientist – OpenClassrooms**.
L'objectif était de concevoir un **assistant IA complet**, capable de répondre à des questions en langage naturel concernant le basketball, en s'appuyant sur deux sources d'information distinctes :

1. **Des documents PDF** contenant des analyses, discussions et contenus textuels.
2. **Une base SQLite** contenant des statistiques NBA structurées.

Le défi principal consistait à créer un système capable de :

- comprendre la question de l'utilisateur,
- choisir automatiquement la bonne stratégie (RAG ou SQL),
- extraire l'information pertinente,
- et formuler une réponse claire et fiable.

Le code complet du projet est disponible sur GitHub :
**https://github.com/LyAbdourahmane/nba-ai-assistant-rag-sql**

## 2. Objectifs du projet

Les objectifs techniques étaient les suivants :

- Construire un pipeline d'ingestion robuste pour les documents PDF.
- Mettre en place un moteur RAG basé sur FAISS et les embeddings Mistral.
- Développer un SQL Tool capable de générer, valider et exécuter des requêtes SQL.
- Créer un routeur intelligent pour distinguer les questions SQL des questions RAG.
- Concevoir une interface utilisateur simple et intuitive avec Streamlit.
- Ajouter une couche d'observabilité complète avec Logfire.
- Garantir une architecture modulaire, maintenable et extensible.

## 3. Architecture générale du système

L'architecture repose sur six modules principaux :

1. **Ingestion & OCR**
2. **Vectorisation & FAISS**
3. **Moteur RAG**
4. **SQL Tool**
5. **Router (classification SQL/RAG)**
6. **Interface Streamlit**

Chaque module est indépendant et communique avec les autres via des interfaces claires.

### Schéma simplifié :

`PDF → Ingestion → Chunking → Embeddings → FAISS → RAG Engine`

`SQL DB → SQL Tool → Réponse SQL`

`Question → Router → (RAG ou SQL) → Réponse finale`

`UI Streamlit → Interaction utilisateur`

`Logfire → Observabilité`

# 4. Ingestion des données

## 4.1 Sources utilisées

- **PDF** dans `inputs/pdf/`
- **Base SQLite** alimentée par un fichier Excel (non vectorisé)

## 4.2 Extraction du texte

Le pipeline d'ingestion suit une logique robuste :

1. Extraction du texte via PyPDF2.
2. Si le texte est vide → fallback OCR via **EasyOCR**.
3. Nettoyage et normalisation du texte.
4. Construction d'un dictionnaire standardisé.

# 5. Vectorisation & Index FAISS

## 5.1 Chunking

Les documents sont découpés en segments de :

- **1500 caractères**
- **150 caractères de chevauchement**

Ce découpage permet :

- une meilleure granularité,
- une recherche plus précise,
- une réduction des pertes d'information.

## 5.2 Embeddings Mistral

Chaque chunk est transformé en vecteur via :

`mistral-embed`

Les embeddings sont normalisés pour optimiser la similarité cosinus.

## 5.3 Construction FAISS

L'index FAISS est construit avec :

- **IndexFlatIP** (produit scalaire)
- Vecteurs normalisés L2

Les fichiers générés :

`vector_db/faiss_index.idx`

`vector_db/document_chunks.pkl`

### 5.4 Reconstruction automatique

Si FAISS n'existe pas au lancement de l'application :

- l'ingestion est relancée,
- les embeddings sont régénérés,
- l'index est reconstruit automatiquement.

# 6. Moteur RAG

Le moteur RAG suit les étapes suivantes :

1. Embedding de la question utilisateur.
2. Recherche des chunks les plus proches dans FAISS.
3. Sélection des passages pertinents.
4. Construction d'un prompt contenant :
    - la question,
    - les passages trouvés.
5. Appel au modèle Mistral pour générer la réponse.
6. Retour d'une réponse contextualisée.

Le RAG garantit que les réponses sont basées sur des sources réelles.

# 7. SQL Tool

Le SQL Tool permet d'interroger la base SQLite en langage naturel.

### Fonctionnement :

1. Génération d'une requête SQL via Mistral.
2. Validation syntaxique et sécuritaire.
3. Exécution sur la base SQLite.
4. Reformulation du résultat en langage naturel.

### Exemples de questions :

- "Quels sont les meilleurs marqueurs ?"
- "Classe les équipes par net rating."

Ce module rend les statistiques accessibles à tous.

# 8. Router : classification SQL / RAG

Le routeur analyse la question et demande au modèle :

> “Cette question nécessite-t-elle une requête SQL ou une recherche documentaire ?”

Le modèle répond par :

- `"SQL"`
- `"RAG"`

Exemples :

- “Quels sont les meilleurs marqueurs ?” → SQL
- “Que disent les analyses sur les Lakers ?” → RAG

Ce mécanisme rend l’expérience utilisateur fluide et naturelle.

# 9. Interface utilisateur Streamlit

L’interface Streamlit offre :

- un chat interactif,
- un historique des messages,
- un message d’attente pendant le traitement,
- une reconstruction automatique de FAISS au premier lancement.

L’utilisateur n’a rien à configurer : il pose une question, l’assistant s’occupe du reste.

# 10. Observabilité avec Logfire

Logfire permet de tracer :

- les appels au routeur,
- les recherches FAISS,
- les requêtes SQL,
- les appels Mistral,
- les erreurs éventuelles.

Chaque étape du pipeline est visible dans un tableau de bord clair et structuré.

# 11. Résultats obtenus

Le système final est capable de :

- répondre à des questions textuelles basées sur les PDF,
- répondre à des questions statistiques basées sur SQLite,
- choisir automatiquement la bonne stratégie,
- fournir des réponses fiables et contextualisées,

- offrir une interface simple et agréable,
- tracer toutes les étapes du traitement.

# 12. Difficultés rencontrées

- **OCR très lent** sur CPU (EasyOCR).
- **Gestion des chemins FAISS** (nécessité d'un rechargement après reconstruction).
- **Ambiguïté entre PDF et Excel** (résolue par exclusion explicite).
- **Synchronisation entre modules** (résolue par une architecture modulaire).

# 13. Améliorations possibles

- Remplacer EasyOCR par Tesseract pour accélérer l'OCR.
- Ajouter un bouton "Reconstruire FAISS" dans l'UI.
- Ajouter un mode debug affichant les chunks utilisés.
- Étendre l'assistant à d'autres sports.
- Ajouter un cache d'embeddings pour réduire les coûts API.

# 14. Conclusion

Ce projet m'a permis de construire un assistant IA complet, combinant :

- ingestion intelligente,
- vectorisation,
- moteur RAG,
- SQL Tool,
- classification automatique,
- interface utilisateur,
- observabilité professionnelle.

Il constitue une base solide pour un assistant IA spécialisé, extensible et maintenable.

Le code complet est disponible sur GitHub : **https://github.com/LyAbdourahmane/nba-ai-assistant-rag-sql**