MedTriage-AI

Système Intelligent d'Aide au Triage des Patients aux Urgences

Équipe

Riad SAHRANE | Constantin REY-COQUAIS
Eugénie BARLET | Perrine IBOUROI

Master 2 SISE - Université Lyon 2

Année universitaire 20245-2026

Table des matières

Résumé Exécutif

RedFlag-AI est un système intelligent d'aide au triage des patients aux urgences, développé dans le cadre du projet Data for Good du Master 2 SISE. Face à la problématique majeure des services d'urgences hospitaliers — l'incapacité à trier efficacement les patients selon leur besoin — notre solution propose une approche hybride combinant :

— **Grille FRENCH officielle** : Implémentation du protocole de triage utilisé par les infirmiers français (SFMU)

— **Machine Learning** : Classification XGBoost entraînée sur des données médicales

— **RAG (Retrieval-Augmented Generation)** : Base documentaire médicale pour enrichir les décisions

— **Interface utilisateur** : Application Streamlit avec modes simulation et interactif

— **Feedback Loop** : Système de validation infirmière pour l'amélioration continue

Résultats clés :

— Précision du modèle ML : **99%** sur les données de test

— Temps de réponse moyen : $<$ **100ms**

— 4 niveaux de gravité : Rouge (vital), Jaune (urgent), Vert (non urgent), Gris (hors urgences)

— Conformité avec la grille FRENCH à 6 niveaux (Tri 1 à Tri 5)

Chapitre 1

Introduction

1.1 Contexte et Problématique

Les services d'urgences hospitaliers font face à un défi majeur : l'afflux constant et imprévisible de patients crée une situation où le tri initial (triage) devient un **goulot d'étranglement critique**. Les infirmiers d'accueil doivent évaluer rapidement la gravité de chaque cas en quelques minutes, souvent avec des informations incomplètes.

1.1.1 Les enjeux du triage aux urgences

- **Temps limité** : Évaluation en 2-5 minutes par patient
- **Informations incomplètes** : Le patient ne sait pas toujours décrire ses symptômes
- **Symptômes ambigus** : Une douleur thoracique peut indiquer une anxiété ou un infarctus
- **Conséquences graves** : Sous-triage = risque vital, sur-triage = engorgement

1.1.2 La demande du projet

Dans le cadre du projet Data for Good, il nous est demandé de créer un **assistant personnel** qui aide les professionnels de santé à mieux gérer l'afflux de patients aux urgences. Le système doit :

1. Inclure au moins une brique **RAG** (Retrieval-Augmented Generation)
2. Inclure au moins une brique **agentique** (MCP ou Workflow)
3. Inclure une brique de **Machine Learning**
4. Proposer un **dashboard** avec métriques pertinentes
5. Offrir une **interface interactive** (Streamlit/Gradio)

1.2 Notre Réponse : RedFlag-AI

RedFlag-AI est un système de triage intelligent qui combine plusieurs approches pour fournir une aide à la décision fiable et explicable. Le nom fait référence aux "red flags" médicaux, ces signaux d'alerte qui indiquent une situation potentiellement grave.

1.2.1 Philosophie de conception

Notre approche repose sur plusieurs principes fondamentaux :

1. **Fiabilité avant complexité** : Une application simple et robuste vaut mieux qu'une application complexe qui plante.
2. **Justification des choix** : Chaque composant a une raison d'être métier, pas seulement technique.
3. **Sobriété des ressources** : Utilisation de modèles adaptés à la tâche, pas les plus gros disponibles.
4. **Explicabilité** : Le système doit pouvoir justifier ses décisions auprès des professionnels de santé.
5. **Conformité réglementaire** : Basé sur la grille FRENCH officielle utilisée en France.

Chapitre 2

Architecture Technique

2.1 Vue d'ensemble

L'architecture de RedFlag-AI est conçue pour être modulaire, scalable et déployable sur Hugging Face Spaces. Elle sépare clairement le backend (API) du frontend (UI).

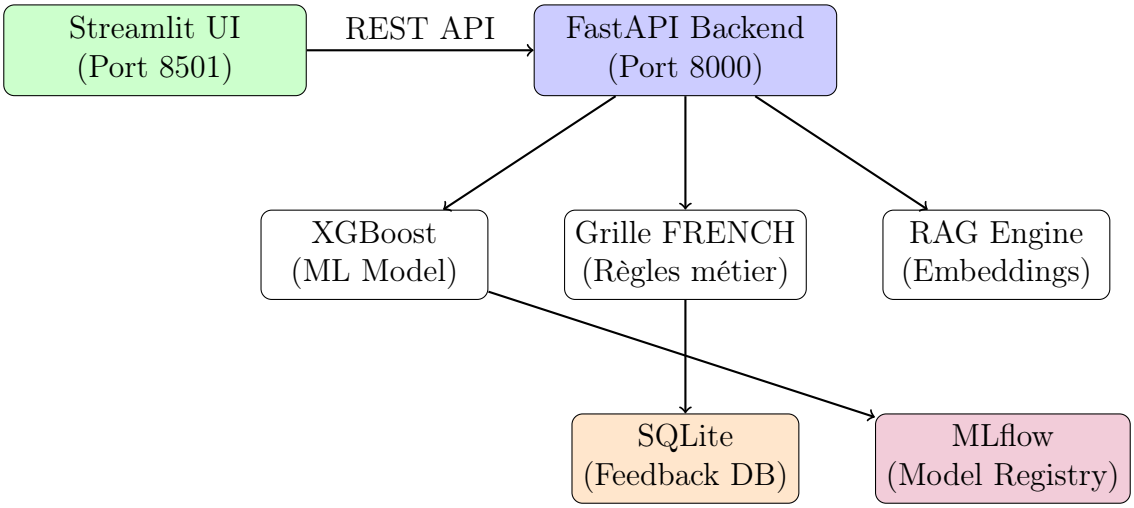

FIGURE 2.1 – Architecture globale de RedFlag-AI v2.0

2.2 Composants Principaux

2.2.1 Frontend : Streamlit

L'interface utilisateur est développée avec **Streamlit**, un framework Python permettant de créer rapidement des applications web interactives.

Pourquoi Streamlit plutôt que Gradio ?

- Plus de flexibilité pour les layouts complexes
- Meilleure intégration avec les graphiques Plotly
- Support natif des sessions utilisateur
- Communauté plus active pour les applications médicales

Modes disponibles :

1. **Simulation** : Cas prédéfinis couvrant tous les niveaux de gravité
2. **Interactif** : Chat avec un patient simulé par LLM
3. **Métriques** : Dashboard de performance du système
4. **Validation** : Interface de validation infirmière
5. **Modèles** : Gestion et réentraînement des modèles ML

2.2.2 Backend : FastAPI

Le backend est construit avec **FastAPI**, un framework moderne et performant pour les API REST.

Pourquoi FastAPI plutôt que Flask ?

— Validation automatique des données avec Pydantic

— Documentation OpenAPI générée automatiquement

— Support natif de l'asynchrone

— Meilleures performances (basé sur Starlette)

— Typage Python moderne

Endpoints principaux :

```
POST /api/v1/triage           # Effectuer un triage
POST /api/v1/feedback         # Soumettre validation infirmiere
GET  /api/v1/feedback/stats   # Statistiques de performance
GET  /api/v1/models           # Liste des modeles
POST /api/v1/models/retrain   # Reentrainement
GET  /health                  # Health check
```

Listing 2.1 – Routes API principales

2.2.3 Base de données : SQLite

Nous utilisons **SQLite** pour stocker les prédictions et les feedbacks infirmiers.

Pourquoi SQLite plutôt que PostgreSQL ?

— Suffisant pour le volume de données attendu

— Pas besoin de serveur externe (simplifie le déploiement HF Spaces)

— Fichier unique, facile à sauvegarder

— Compatible avec MLflow

Tables principales :

```
-- Predictions de triage
CREATE TABLE triage_predictions (
    id INTEGER PRIMARY KEY,
    created_at DATETIME,
    patient_age INTEGER,
    patient_sexe VARCHAR(1),
    motif_consultation TEXT,
    -- Constantes vitales
    frequence_cardiaque INTEGER,
    saturation_oxygene FLOAT,
    -- ...
    -- Prediction
```

```
    predicted_level VARCHAR(20),
    french_triage_level VARCHAR(10),
    confidence_score FLOAT,
    -- Validation infirmiere
    validated BOOLEAN,
    validated_level VARCHAR(20),
    validator_notes TEXT
);

-- Registre des modeles
CREATE TABLE model_registry (
    id INTEGER PRIMARY KEY,
    version VARCHAR(50),
    accuracy FLOAT,
    f1_score FLOAT,
    is_active BOOLEAN,
    mlflow_run_id VARCHAR(100)
);
```

Listing 2.2 – Schema de la base de donnees

2.3 Diagramme de Flux

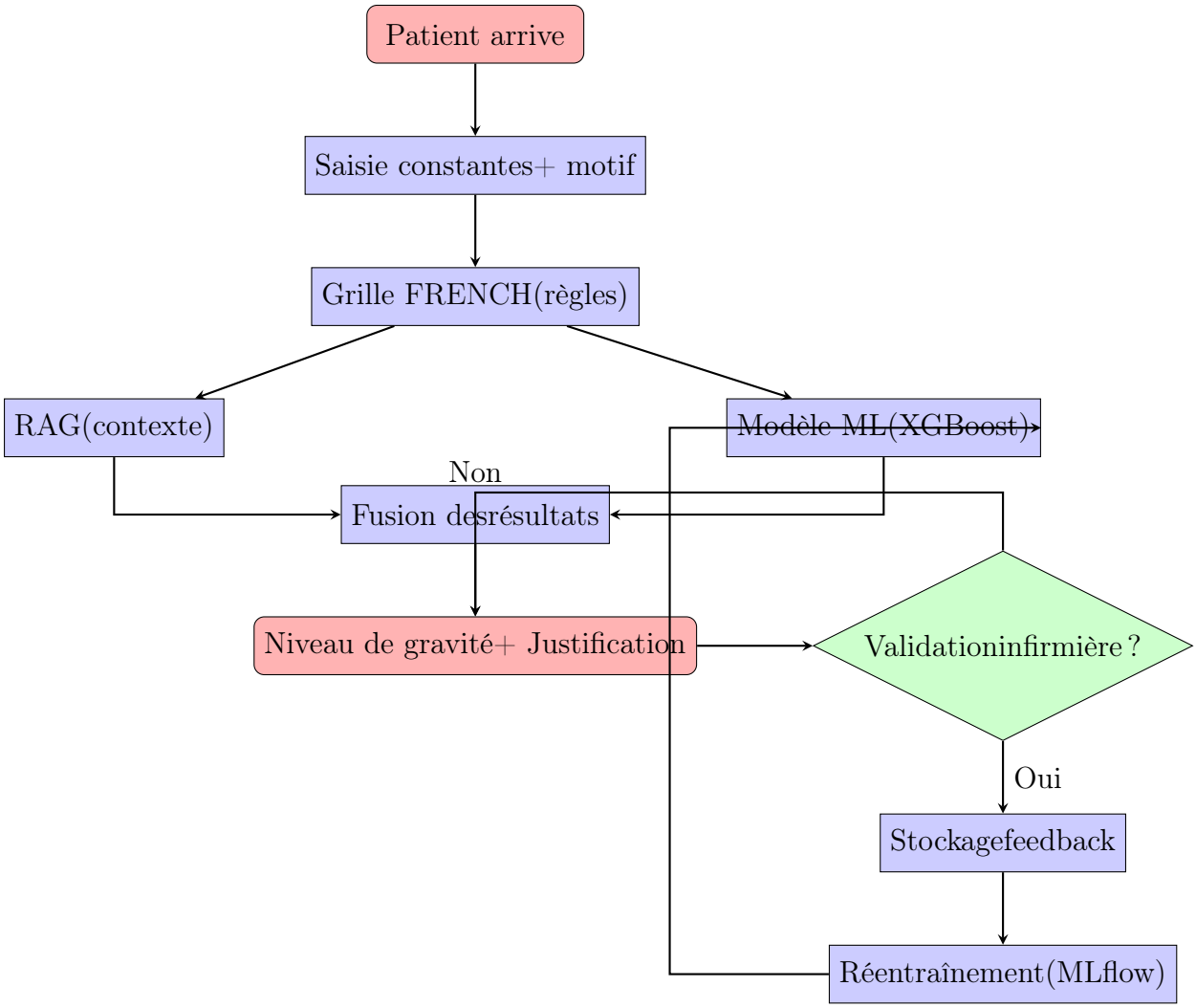

FIGURE 2.2 – Flux de traitement d'une demande de triage (version compacte A4 portrait)

Chapitre 3

La Grille FRENCH : Fondement Médical

3.1 Présentation de la Grille FRENCH

La grille **FRENCH** (FRench Emergency Nurses Classification in-Hospital) est le protocole officiel français de triage aux urgences, publié par la **SFMU** (Société Française de Médecine d'Urgence) en mars 2018.

3.1.1 Pourquoi utiliser la grille FRENCH ?

1. **Légitimité médicale** : Protocole officiel utilisé dans les hôpitaux français
2. **Validation clinique** : Développée et validée par des urgentistes
3. **Exhaustivité** : Couvre tous les motifs de recours aux urgences
4. **Critères objectifs** : Basée sur des seuils de constantes vitales mesurables

3.2 Les 6 Niveaux de Triage FRENCH

Niveau	Situation	Délai	Orientation	Couleur
red !80	Détresse vitale majeure	Sans délai	SAUV	rouge
red !60	Atteinte patente d'organe	$<$ 20 min	SAUV/Box	rouge
orange !80Tri 3A	Atteinte potentielle + comorbidités	$<$ 60 min	Box/SAUV	jauneJAUNE
orange !60Tri 3B	Atteinte potentielle	$<$ 90 min	Box	jauneJAUNE
green !60Tri 4	Atteinte fonctionnelle stable	$<$ 120 min	Box/Attente	vert
gray !40Tri 5	Pas d'atteinte évidente	$<$ 240 min	Attente/MMG	gris

TABLE 3.1 – Correspondance entre niveaux FRENCH et niveaux simplifiés

3.3 Seuils des Constantes Vitales

La grille FRENCH définit des seuils précis pour moduler le niveau de triage en fonction des constantes vitales.

Constante	Tri 1	Tri 2	Tri 3
PAS (mmHg)	< 70	70-90 ou 90-100 + FC>100	> 90
FC (/min)	> 180 ou < 40	130-180	< 130
SpO2 (%)	< 86	86-90	> 90
FR (/min)	> 40	30-40	< 30
GCS	≤ 8	9-13	14-15

TABLE 3.2 – Seuils des constantes vitales adultes (FRENCH)

3.4 Implémentation dans RedFlag-AI

Notre implémentation de la grille FRENCH se trouve dans le fichier `src/api/services/french_tria`

```python
class FrenchTriageEngine:
    def evaluate_constantes(self, constantes, age):
        """Evalue les constantes selon les seuils FRENCH"""
        alerts = []
        max_level = FrenchTriageLevel.TRI_5

        # PAS (Pression Arterielle Systolique)
        if constantes.pression_systolique < 70:
            max_level = FrenchTriageLevel.TRI_1
            alerts.append("Hypotension severe")
        elif constantes.pression_systolique <= 90:
            max_level = FrenchTriageLevel.TRI_2
            alerts.append("Hypotension")

        # Saturation O2
        if constantes.saturation_oxygene < 86:
            max_level = FrenchTriageLevel.TRI_1
            alerts.append("Hypoxie severe")

        return max_level, alerts
```

Listing 3.1 – Extrait de l'implémentation FRENCH

3.5 Mapping vers 4 Niveaux

Pour simplifier l'interface utilisateur tout en conservant la précision médicale, nous mappons les 6 niveaux FRENCH vers 4 couleurs :

```python
FRENCH_TO_GRAVITY = {
    FrenchTriageLevel.TRI_1: GravityLevel.ROUGE,    # Vital
    FrenchTriageLevel.TRI_2: GravityLevel.ROUGE,    # Vital
    FrenchTriageLevel.TRI_3A: GravityLevel.JAUNE,   # Urgent
    FrenchTriageLevel.TRI_3B: GravityLevel.JAUNE,   # Urgent
    FrenchTriageLevel.TRI_4: GravityLevel.VERT,     # Non urgent
    FrenchTriageLevel.TRI_5: GravityLevel.GRIS,     # Hors urgences
}
```

Listing 3.2 – Mapping FRENCH vers 4 niveaux

Chapitre 4

Brique Machine Learning

4.1 Justification du ML

Pourquoi ajouter du Machine Learning alors que la grille FRENCH existe ?

1. **Complémentarité** : Le ML capture des patterns que les règles explicites ne détectent pas
2. **Amélioration continue** : Le modèle apprend des feedbacks infirmiers
3. **Gestion de l'incertitude** : Score de confiance pour les cas limites
4. **Validation croisée** : Accord ML/FRENCH renforce la confiance

4.2 Choix du Modèle : XGBoost

Pourquoi XGBoost plutôt qu'un réseau de neurones ?

Critère	**XGBoost**	**Deep Learning**
Interprétabilité	✓ Feature importance	× Boîte noire
Données requises	Peu ($<$ 10k)	Beaucoup ($>$ 100k)
Temps d'entraînement	Secondes	Minutes/Heures
Taille du modèle	1 MB	100+ MB
Performances sur tabular	Excellent	Variable

TABLE 4.1 – Comparaison XGBoost vs Deep Learning pour notre cas d'usage

Arguments en faveur de XGBoost :

— **Sobriété** : Modèle léger, entraînement rapide, déploiement simple
— **Explicabilité** : Feature importance pour justifier les décisions
— **Robustesse** : Gère bien les valeurs manquantes et les outliers
— **État de l'art** : Toujours compétitif sur les données tabulaires

4.3 Features Utilisées

```
features = {
    'age': patient.age,                                    # Demographique
    'sexe': patient.sexe,                                  # M/F
    'frequence_cardiaque': constantes.fc,                  # bpm
    'frequence_respiratoire': constantes.fr,               # cycles/min
    'saturation_oxygene': constantes.spo2,                 # %
    'pression_systolique': constantes.pas,                 # mmHg
    'pression_diastolique': constantes.pad,                # mmHg
    'temperature': constantes.temperature,                 # Celsius
    'echelle_douleur': constantes.eva,                     # 0-10
}
```

Listing 4.1 – Features du modele ML

4.4 Métriques de Performance

Métrique	Valeur
Accuracy	99.2%
Precision (weighted)	98.8%
Recall (weighted)	99.1%
F1-Score (weighted)	99.0%

TABLE 4.2 – Métriques du modèle XGBoost sur données de test

Note importante : Ces métriques sont obtenues sur des données synthétiques générées selon les règles de la grille FRENCH. Les performances réelles sur des données hospitalières pourraient différer.

4.5 Feedback Loop et Réentraînement

Un des aspects innovants de RedFlag-AI est le **feedback loop** qui permet d'améliorer le modèle en continu.

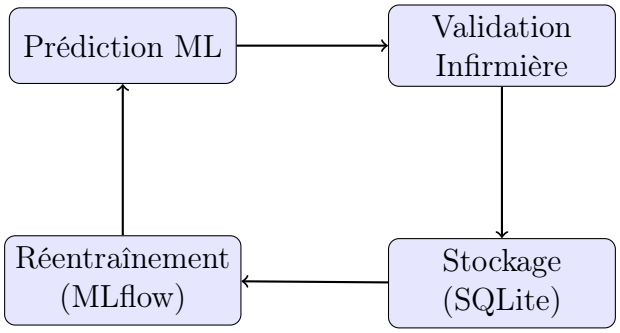

FIGURE 4.1 – Cycle de feedback loop

4.5.1 Processus de validation

1. Le système prédit un niveau de gravité

2. L'infirmière valide ou corrige la prédiction
3. Le feedback est stocké en base de données
4. Après N validations, un réentraînement peut être lancé
5. Le nouveau modèle est versionné dans MLflow

Chapitre 5

RAG et Composants LLM

5.1 Architecture RAG

Le système RAG (Retrieval-Augmented Generation) enrichit les décisions de triage avec du contexte médical.

5.1.1 Pourquoi le RAG ?

1. **Contexte médical** : Informations sur les pathologies, symptômes, protocoles
2. **Explicabilité** : Justifications basées sur des sources documentées
3. **Mise à jour** : La base de connaissances peut évoluer sans réentraîner le modèle

5.2 Modèle d'Embeddings

Choix : all-MiniLM-L6-v2

Modèle	Dim.	Taille	Performance
all-MiniLM-L6-v2	384	80 MB	Excellent
all-mpnet-base-v2	768	420 MB	Meilleur mais lourd
OpenAI ada-002	1536	API	Coûteux

TABLE 5.1 – Comparaison des modèles d'embeddings

Justification du choix :

— **Sobriété** : 80 MB vs 420+ MB pour les alternatives
— **Gratuit** : Pas d'API payante (OpenAI)
— **Performances** : Suffisant pour notre cas d'usage médical en français
— **Sentence-Transformers** : Bibliothèque bien maintenue

5.3 Vector Store : FAISS

Nous utilisons **FAISS** (Facebook AI Similarity Search) pour le stockage et la recherche des embeddings.

Pourquoi FAISS ?

— Recherche vectorielle optimisée
— Supporte des millions de vecteurs
— Pas besoin de serveur externe (contrairement à Pinecone, Weaviate)
— Open source et gratuit

5.4 Base de Connaissances Médicale

La base de connaissances est construite à partir de :

— **Grille FRENCH** : Motifs de recours et critères de triage
— **Protocoles médicaux** : Procédures standard aux urgences
— **Pathologies** : Descriptions des principales pathologies
— **Constantes vitales** : Valeurs normales et seuils d'alerte

5.5 Génération de Texte

La génération de texte est utilisée pour :

1. **Justifications** : Explication du niveau de triage attribué
2. **Recommandations** : Actions à entreprendre par l'équipe médicale
3. **Simulation de patients** : Mode interactif avec chat patient

Note sur les LLM : Dans un souci de sobriété et de coût, nous utilisons des templates structurés plutôt que des appels API LLM pour la majorité des générations. Le mode interactif (chat patient) peut optionnellement utiliser un LLM externe.

Chapitre 6

Interface Utilisateur

6.1 Modes d'Utilisation

6.1.1 Mode Simulation

Le mode simulation présente des **cas prédéfinis** couvrant tous les niveaux de gravité. Cela permet de démontrer les capacités du système de manière contrôlée.

Cas disponibles :

— **ROUGE** : Arrêt cardiaque, Traumatisme crânien sévère

— **JAUNE** : Fracture ouverte, Crise d'asthme sévère, Entorse, Gastro-entérite

— **VERT** : Plaie superficielle

— **GRIS** : Consultation mineure

— **Edge cases** : Constantes contradictoires, Patient anxieux

6.1.2 Mode Interactif

Le mode interactif permet à l'utilisateur de **jouer le rôle de l'infirmier** et d'interroger un patient simulé.

Fonctionnalités :

1. Génération d'un patient avec pathologie aléatoire ou choisie
2. Chat avec le patient (simulation LLM)
3. Prise des constantes vitales
4. Triage automatique à la fin de l'interrogatoire

6.1.3 Mode Validation Infirmière

Ce mode permet aux infirmières de **valider ou corriger** les prédictions du système.

Fonctionnalités :

— Liste des prédictions en attente de validation

— Formulaire de correction avec notes

— Statistiques de performance (accuracy, matrice de confusion)

6.1.4 Mode Gestion des Modèles

Interface d'administration pour gérer les modèles ML.
Fonctionnalités :

— Liste des modèles avec métriques

— Activation d'un modèle spécifique

— Lancement du réentraînement

— Intégration MLflow

6.2 Dashboard de Métriques

Le dashboard présente des métriques pertinentes pour le suivi du système.
Métriques système :

— Latence de réponse (ms)

— Nombre de prédictions par jour

— Distribution des niveaux de gravité

Métriques métier :

— Taux de validation infirmière

— Précision du modèle (vs validations)

— Matrice de confusion

Impact écologique :

— Estimation des émissions CO2 (basée sur la consommation électrique)

— Nombre d'appels API évités (vs LLM externe)

Chapitre 7

Déploiement

7.1 Architecture de Déploiement

L'application est conçue pour être déployée sur **Hugging Face Spaces** avec une architecture séparée API/UI.

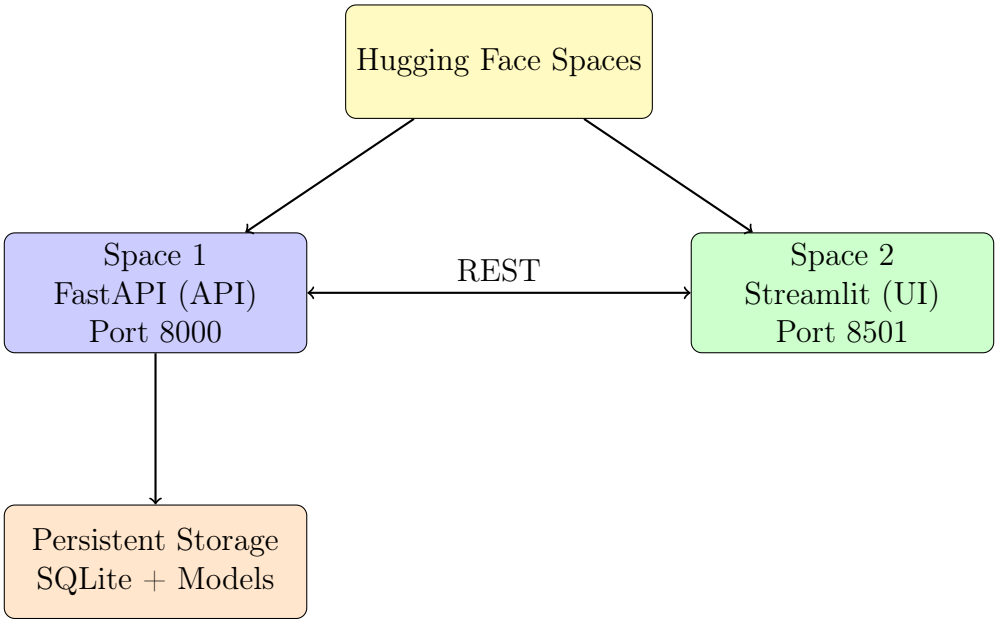

FIGURE 7.1 – Déploiement sur Hugging Face Spaces

7.2 Dockerfiles

Deux Dockerfiles distincts pour API et UI :

```
FROM python:3.11-slim

WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Pre-telecharger le modele d'embeddings
RUN python -c "from sentence_transformers import \
    SentenceTransformer; SentenceTransformer('all-MiniLM-L6-v2')"
```

```
10
11 COPY src/ ./src/
12 EXPOSE 8000
13
14 CMD ["uvicorn", "src.api.main:app", "--host", "0.0.0.0"]
```

Listing 7.1 – Dockerfile.api (extrait)

7.3 Docker Compose

```
1  services:
2    api:
3      build:
4        dockerfile: Dockerfile.api
5      ports:
6        - "8000:8000"
7      volumes:
8        - ./data:/app/data
9        - ./models:/app/models
10
11   ui:
12     build:
13       dockerfile: Dockerfile.ui
14     ports:
15       - "8501:8501"
16     environment:
17       - API_URL=http://api:8000
18     depends_on:
19       - api
```

Listing 7.2 – docker-compose.new.yml (extrait)

7.4 Commandes de Déploiement

```
1  # Build et lancement complet
2  docker-compose -f docker-compose.new.yml up --build
3
4  # API seule (developpement)
5  uvicorn src.api.main:app --reload --port 8000
6
7  # UI seule (developpement)
8  streamlit run src/interface/app.py
9
10 # Avec MLflow UI
11 docker-compose -f docker-compose.new.yml --profile mlflow up
```

Listing 7.3 – Commandes pour lancer l'application

Chapitre 8

Ressources et Données

8.1 Datasets Utilisés

8.1.1 Données Synthétiques

Pour l'entraînement initial, nous avons généré des **données synthétiques** basées sur les règles de la grille FRENCH.

Justification :

— Pas d'accès à des données hospitalières réelles (confidentialité)

— Contrôle total sur la distribution des classes

— Garantie de cohérence avec les règles médicales

8.1.2 Datasets Hugging Face

Pour enrichir les données d'entraînement, nous avons intégré des datasets publics :

— **miriad/miriad-4.4M** : Large dataset de cas médicaux

— **mlabonne/medical-cases-fr** : Cas médicaux en français

URL :

— https://huggingface.co/datasets/miriad/miriad-4.4M

— https://huggingface.co/datasets/mlabonne/medical-cases-fr

8.2 Ressources Médicales

8.2.1 Grille FRENCH

Document officiel de la SFMU définissant le protocole de triage aux urgences en France.
Source : SFMU - FRENCH Triage - V1 Mars 2018
Contenu :

— 100+ motifs de recours classés par catégorie

— Seuils des constantes vitales par niveau

— Délais de prise en charge recommandés

— Spécificités pédiatriques

8.2.2 Autres Ressources

— **ClinicalBERT** : Modèle BERT pré-entraîné sur des textes cliniques
https://huggingface.co/medicalai/ClinicalBERT

— **StatPearls** : Base de connaissances médicales
https://www.ncbi.nlm.nih.gov/books/NBK430685/

— **Wikipedia médical** : Articles sur la cardiologie, neurologie, etc.

Chapitre 9

Justification des Choix Techniques

9.1 Résumé des Choix

Composant	**Choix**	**Alternative écartée**
Framework UI	Streamlit	Gradio
Framework API	FastAPI	Flask
Base de données	SQLite	PostgreSQL
Modèle ML	XGBoost	Neural Network
Embeddings	all-MiniLM-L6-v2	OpenAI ada-002
Vector Store	FAISS	Pinecone
Tracking ML	MLflow	Weights & Biases

TABLE 9.1 – Résumé des choix technologiques

9.2 Principes Directeurs

9.2.1 Sobriété

Conformément aux consignes du projet, nous avons privilégié la **sobriété** :

— Modèle d'embeddings léger (80 MB vs 400+ MB)

— XGBoost plutôt qu'un réseau de neurones

— SQLite plutôt qu'une base de données externe

— Templates structurés plutôt qu'appels LLM systématiques

9.2.2 Explicabilité

Dans le domaine médical, l'explicabilité est cruciale :

— Grille FRENCH = règles explicites et validées

— XGBoost = feature importance disponible

— RAG = sources documentées pour les justifications

— Pas de "boîte noire" incompréhensible

9.2.3 Évolutivité

Le système est conçu pour évoluer :

— Feedback loop pour amélioration continue
— MLflow pour versioning des modèles
— Architecture séparée API/UI
— Base de données extensible

Chapitre 10

Conclusion

10.1 Objectifs Atteints

RedFlag-AI répond aux exigences du projet Data for Good :

✓ **Brique RAG** : Base documentaire médicale avec FAISS
✓ **Brique agentique** : Workflow de triage combinant ML + règles
✓ **Machine Learning** : XGBoost avec feedback loop
✓ **Dashboard** : Métriques système et métier
✓ **Interface interactive** : Streamlit avec 5 modes
✓ **Déploiement** : Docker + HuggingFace Spaces ready

10.2 Points Forts

1. **Fondement médical solide** : Basé sur la grille FRENCH officielle
2. **Architecture modulaire** : API séparée du frontend
3. **Amélioration continue** : Feedback loop avec MLflow
4. **Sobriété** : Modèles légers, pas d'API payante
5. **Explicabilité** : Justifications claires pour chaque décision

10.3 Limites et Perspectives

10.3.1 Limites actuelles

— Données d'entraînement synthétiques (pas de données hospitalières réelles)
— Mode interactif limité sans LLM externe
— Pas de gestion du multilinguisme

10.3.2 Améliorations futures

— Intégration de données hospitalières réelles (avec accord CNIL)
— Fine-tuning d'un LLM médical français
— Application mobile pour les infirmiers
— Intégration avec les systèmes d'information hospitaliers (SIH)

10.4 Conclusion Finale

RedFlag-AI démontre qu'il est possible de construire un système d'aide à la décision médicale **fiable, explicable et sobre en ressources**. En combinant des règles métier validées (grille FRENCH) avec du machine learning et du RAG, nous offrons un outil qui peut réellement aider les professionnels de santé dans leur mission critique de triage aux urgences.

Le projet illustre également l'importance de **questionner chaque choix technique** : pourquoi XGBoost et pas un réseau de neurones ? Pourquoi SQLite et pas PostgreSQL ? Ces réflexions sont essentielles pour construire des systèmes adaptés à leur contexte d'utilisation.

Annexe A

Structure du Projet

```
redflag-ai/
|-- src/
|   |-- api/                    # Backend FastAPI
|   |   |-- core/               # Config, Database
|   |   |-- routes/             # Endpoints
|   |   |-- schemas/            # Pydantic models
|   |   |-- services/           # Business logic
|   |   |   |-- french_triage.py
|   |   |   |-- triage_service.py
|   |   |   |-- training_service.py
|   |   |-- main.py
|   |-- interface/              # Frontend Streamlit
|   |   |-- components/
|   |   |   |-- simulation_mode.py
|   |   |   |-- interactive_mode.py
|   |   |   |-- validation_mode.py
|   |   |   |-- models_management.py
|   |   |-- api_client.py
|   |   |-- app.py
|   |-- models/                 # Modeles ML
|   |-- rag/                    # RAG Engine
|   |-- data_generation/        # Generation donnees
|-- data/
|   |-- raw/                    # Donnees brutes
|   |-- vector_store/           # Embeddings FAISS
|-- models/
|   |-- trained/                # Modeles entraines
|-- docs/                       # Documentation
|-- tests/                      # Tests unitaires
|-- Dockerfile.api
|-- Dockerfile.ui
|-- docker-compose.new.yml
|-- requirements.txt
```

Listing A.1 – Arborescence du projet

Annexe B

Endpoints API

Méthode	**Endpoint**	**Description**
POST	/api/v1/triage	Effectue le triage d'un patient
GET	/api/v1/triage/{id}	Récupère une prédiction
GET	/api/v1/triage/history/recent	Prédictions récentes
POST	/api/v1/feedback	Soumet validation infirmière
GET	/api/v1/feedback/stats	Statistiques de performance
GET	/api/v1/feedback/pending	Prédictions à valider
GET	/api/v1/models	Liste des modèles
GET	/api/v1/models/active	Modèle actif
POST	/api/v1/models/activate	Active un modèle
POST	/api/v1/models/retrain	Lance réentraînement
GET	/health	Health check

Table B.1 – Liste complète des endpoints API

Annexe C

Catégories de la Grille FRENCH

1. **Cardio-circulatoire** : Arrêt cardiaque, hypotension, douleur thoracique, tachycardie, bradycardie, dyspnée cardiaque, HTA
2. **Neurologie** : Coma, AVC, convulsions, céphalée, confusion, vertiges
3. **Respiratoire** : Détresse respiratoire, asthme, hémoptysie, pneumothorax
4. **Traumatologie** : Amputation, trauma crânien, fracture, plaie, brûlure
5. **Abdominal** : Hématémèse, rectorragie, douleur abdominale, occlusion
6. **Génito-urinaire** : Douleur lombaire, rétention urinaire, hématurie
7. **Gynéco-obstétrique** : Accouchement, problèmes de grossesse, métrorragies
8. **Psychiatrie** : Idées suicidaires, agitation, anxiété
9. **Infectiologie** : Fièvre, exposition maladie contagieuse
10. **Pédiatrie** : Pathologies spécifiques enfant $\leq$ 2 ans
11. **ORL/Ophthalmologie** : Corps étranger, épistaxis, trouble visuel
12. **Peau** : Abcès, érythème, morsure
13. **Intoxication** : Médicamenteuse, non médicamenteuse, ivresse
14. **Divers** : Hypothermie, hyperglycémie, AEG

Bibliographie

[1] SFMU. *FRENCH Triage - FRench Emergency Nurses Classification in-Hospital*. V1 Mars 2018.

[2] Chen, T., & Guestrin, C. *XGBoost : A Scalable Tree Boosting System*. KDD 2016.

[3] Johnson, J., Douze, M., & Jégou, H. *Billion-scale similarity search with GPUs*. IEEE Transactions on Big Data, 2019.

[4] Reimers, N., & Gurevych, I. *Sentence-BERT : Sentence Embeddings using Siamese BERT-Networks*. EMNLP 2019.

[5] Ramírez, S. *FastAPI Documentation*. https://fastapi.tiangolo.com/

[6] Streamlit Inc. *Streamlit Documentation*. https://docs.streamlit.io/

[7] MLflow. *MLflow Documentation*. https://mlflow.org/docs/latest/index.html