Air Paradis – Analyse de sentiment

PROJET : RÉALISEZ UNE ANALYSE DE SENTIMENTS GRÂCE AU DEEP LEARNING

Sommaire :

Contexte

- Projet : prédiction du sentiment de tweets (positif / négatif)
- Client : compagnie aérienne Air Paradis
- Objectifs : anticiper les bad buzz et prototyper un produit IA industrialisable

Problématique métier & objectifs :

- Air Paradis a une mauvaise image régulière sur les réseaux

- Besoin : détecter rapidement les bad buzz potentiels

- Pas de données internes → utilisation d'un dataset open source

- Eléments clés :

    - modèle de scoring

    - API de prédiction + interface

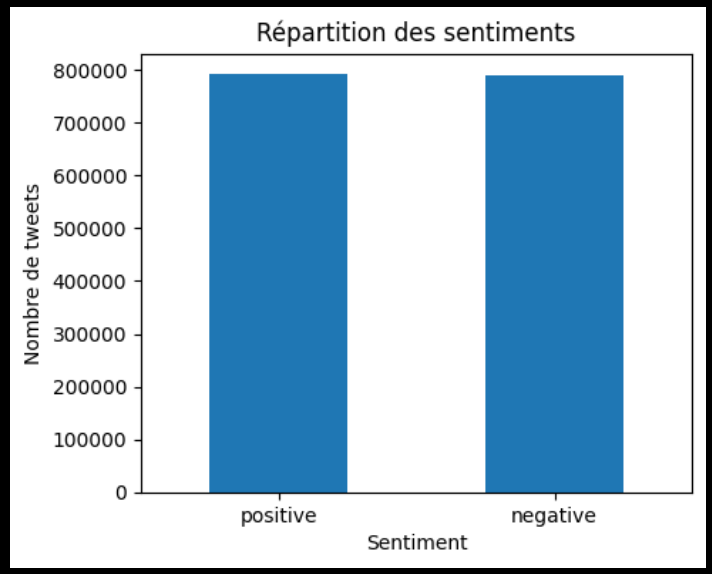
    - démarche MLOps (tracking, CI, monitoring, alertes)

Données & EDA

- Dataset : 1,6 M de tweets annotés binaire (≈ 50 % / 50 %)

- Colonnes : label, id, date, user, texte

- Contrôles :
    - équilibrage des classes
    - valeurs manquantes / lignes corrompues
    - distribution de la longueur des tweets (avant / après nettoyage)



	count	mean	std	min	25%	50%	75%	max
len_raw	1600000	13.18	6.96	1	7	12	19	64
len_simple	1600000	6.69	3.74	0	4	6	9	33
len_adv	1600000	11.82	6.53	0	6	11	17	52
len_bert	1600000	12.65	6.96	0	7	12	18	64

Prétraitement du texte :

- Nettoyage commun :
    - minuscules, suppression URLs, mentions, caractères spéciaux

- Trois variantes :
    - simple : très agressif (stopwords + lemmatisation) → TF-IDF

    - avancé : nettoyage plus léger → embeddings + LSTM

    - BERT : nettoyage minimal → tokenizer ModernBERT

Filtrage des tweets trop courts après nettoyage

dataset	count	mean	std	min	25%	50%	75%	max
Base	1600000	13.18	6.96	1	7	12	19	64
Simple	1527316	6.97	3.60	2	4	6	10	33
Avancé	1579630	11.96	6.45	2	7	11	17	52
BERT	1597185	12.67	6.94	1	7	12	18	64

Modèle 1 : baseline TF-IDF + régression logistique

- Pipeline scikit-learn :
    - preprocess_simple → TfidfVectorizer → LogisticRegression
- Split train/test fixe partagé par tous les modèles
- Métriques :
    - accuracy, F1, précision, rappel, matrice de confusion
- Points forts :
    - léger, rapide, simple à déployer

	precision	recall	f1-score	support
0	0.8035	0.7788	0.7910	153528
1	0.7832	0.8075	0.7952	151936
accuracy			0.7931	305464
macro avg	0.7934	0.7932	0.7931	305464
weighted avg	0.7934	0.7931	0.7931	305464

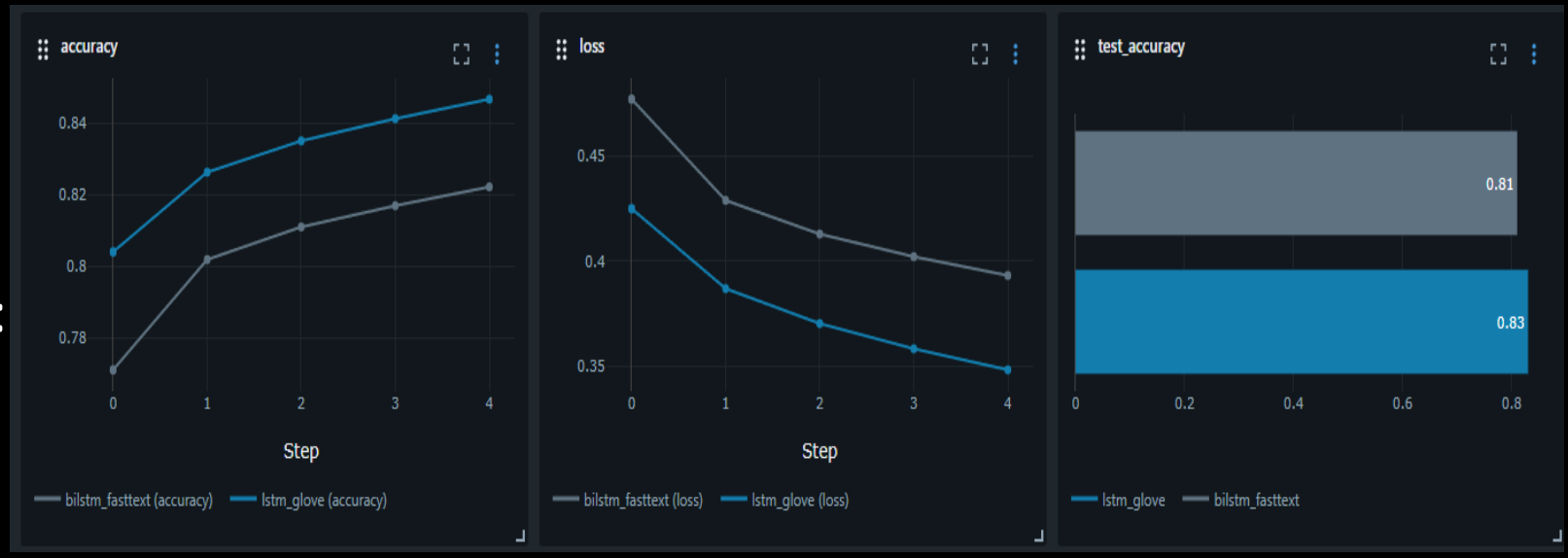
Modèle 2 : embeddings + LSTM

- Prétraitement “avancé”

- Embeddings pré-entraînés :
  - GloVe + Fastext

- Architecture :
  - Embedding → LSTM → Dense sigmoïde

- Objectif : mieux capturer le contexte et la sémantique



Run Name	Cr	Duration	accuracy	loss	test_accuracy	test_f1	test_precision	test_recall	test_roc_auc
bilstm_fasttext		16.6min	0.82215416...	0.39307889...	0.81047848...	0.79734233...	0.85162419...	0.74956560...	0.89735719...
lstm_glove		12.0min	0.84658253...	0.34818631...	0.83040554...	0.82650645...	0.84136421...	0.81216433...	0.91162285...

Modèle 3 : ModernBERT

- Tokenisation ModernBERT (troncature, padding)

- Entraînement sur un sous-échantillon (contraintes CPU / temps ou GPU / coût)

- Entraînement via Trainer (Transformers)

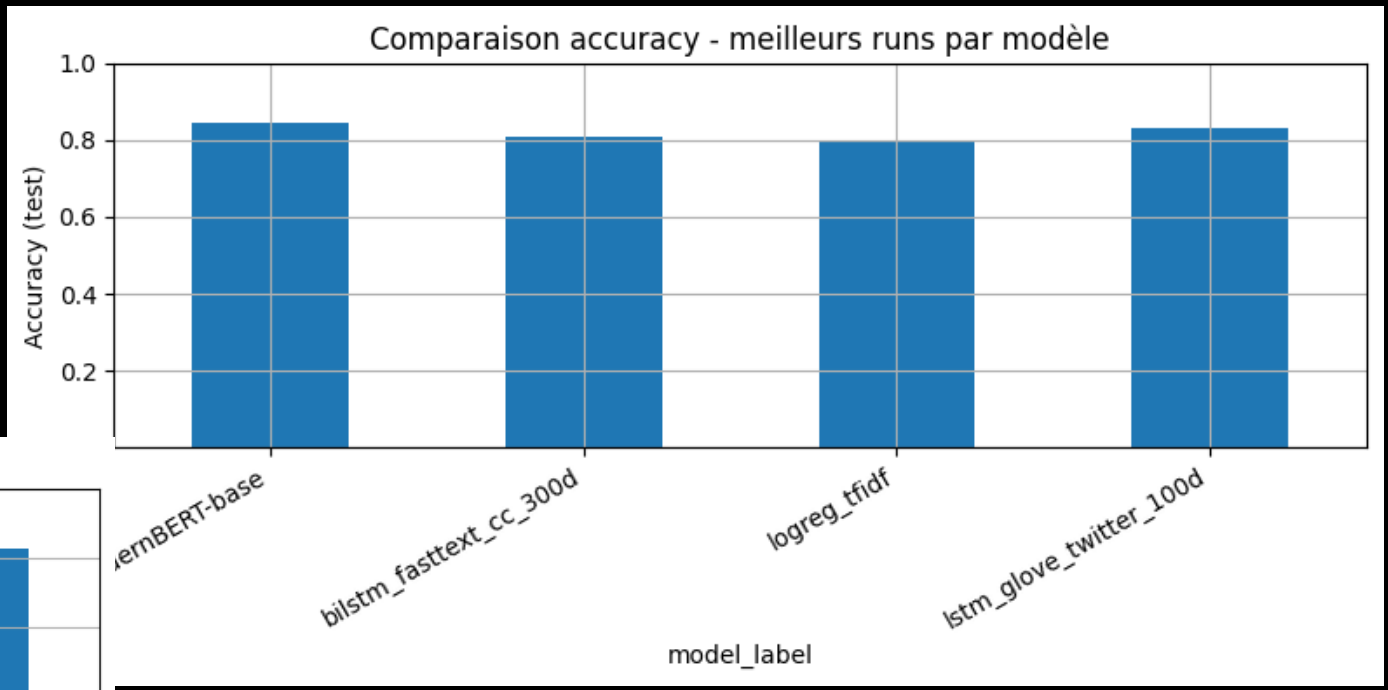
- Très bonnes performances, mais :
    - modèle lourd
    - Latence d'inférence plus élevée
    - déploiement plus coûteux

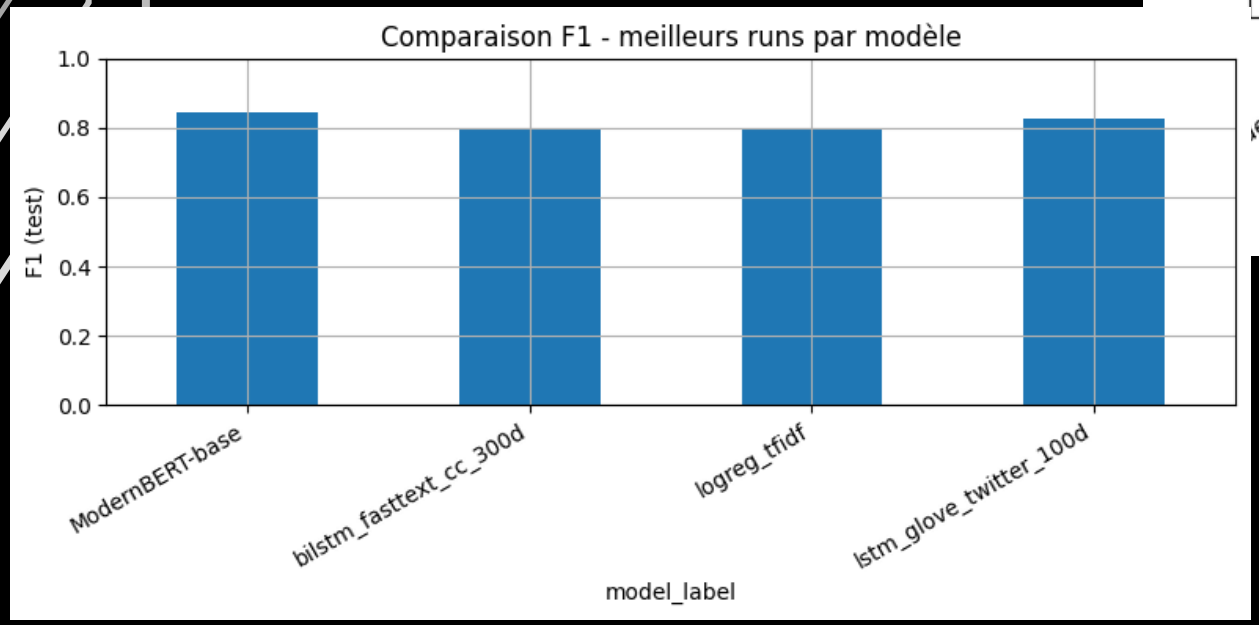
```
{'eval_loss': 0.35788649320602417,
 'eval_accuracy': 0.84474,
 'eval_precision': 0.842803865741669,
 'eval_recall': 0.8454143201930813,
 'eval_f1': 0.8441070747233769,
 'eval_roc_auc': 0.9217781821637926,
 'eval_runtime': 1961.3742,
 'eval_samples_per_second': 25.492,
 'eval_steps_per_second': 0.05,
 'epoch': 3.0}
```

Run Name	Cr	Duration	epoch	eval_accuracy	eval_f1	eval_loss	eval_precision	eval_recall	eval_roc_auc	eval_runtime	eval_samples_pe	eval_steps_per_second
modernbert_base		5.2h	3	0.84474	0.84410707...	0.35788649...	0.84280386...	0.84541432...	0.92177818...	1961.3742	25.492	0.05
modernbert_base		2.1h	3	0.81975	0.81777283...	0.40659123...	0.81649338...	0.81905629...	0.89736798...	767.0253	26.075	0.052
modernbert_base		13.3min	3	0.686	0.68662674...	0.59290075...	0.68253968...	0.69076305...	0.74725295...	80.4171	24.87	0.199

Comparaison des modèles

model_label	experiment	run_id	score_for_sort	test_accuracy	test_f1	test_precision	test_recall	test_roc_auc	train_loss	val_loss	val_accuracy
ModernBERT-base	sentiment_airparadis_bert	3c865f091fc0	0.8441070747233769	0.84474	0.8441070747	0.842803865741669	0.84541432019308	0.9217781821637926	0.387263161795479	0.3578864932060241	*Missing value*
bilstm_fasttext_cc_300d	sentiment_airparadis_modele_avan	c765b13efe2b	0.7973423322504761	0.8104784851897441	0.7973423322	0.851624192390524	0.74956560657118	0.8973571987590773	0.393078893423080	0.6276527047157288	0.6881803274154663
logreg_tfidf	sentiment_airparadis_modele_simp	5c675ade24f8	0.7951975112608963	0.7931016420920305	0.7951975112	0.7832228988726748	0.80754396588037(	0.8743315727541261	*Missing value*	*Missing value*	*Missing value*
lstm_glove_twitter_100d	sentiment_airparadis_modele_avan	57ddba0ab02	0.8265064517965566	0.8304055469711652	0.8265064517	0.8413642159235799	0.81216433235046	0.9116228564071932	0.348186314105987	0.4653372168540954(	0.7837886214256287





Principes MLOps (1/2)

- Séparer :
    - exploration vs code industrialisable

- Rendre les expériences reproductibles :
    - versionning du code et des données
    - tracking des métriques, hyperparamètres, modèles

- Automatiser :
    - tests, déploiement, monitoring

- Surveiller :
    - dérive des performances

    - incidents / alertes

MLOps appliqué au projet (2/2)

- MLflow :
    - tracking des runs (params, métriques, figures)
    - enregistrement des modèles

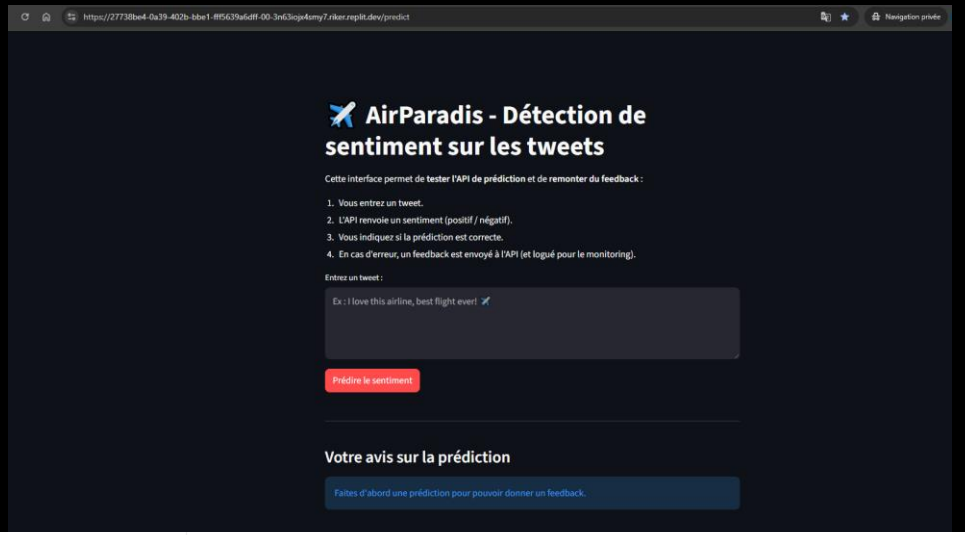
- Git + GitHub :
    - versionning du code
    - workflow GitHub Actions → pytest

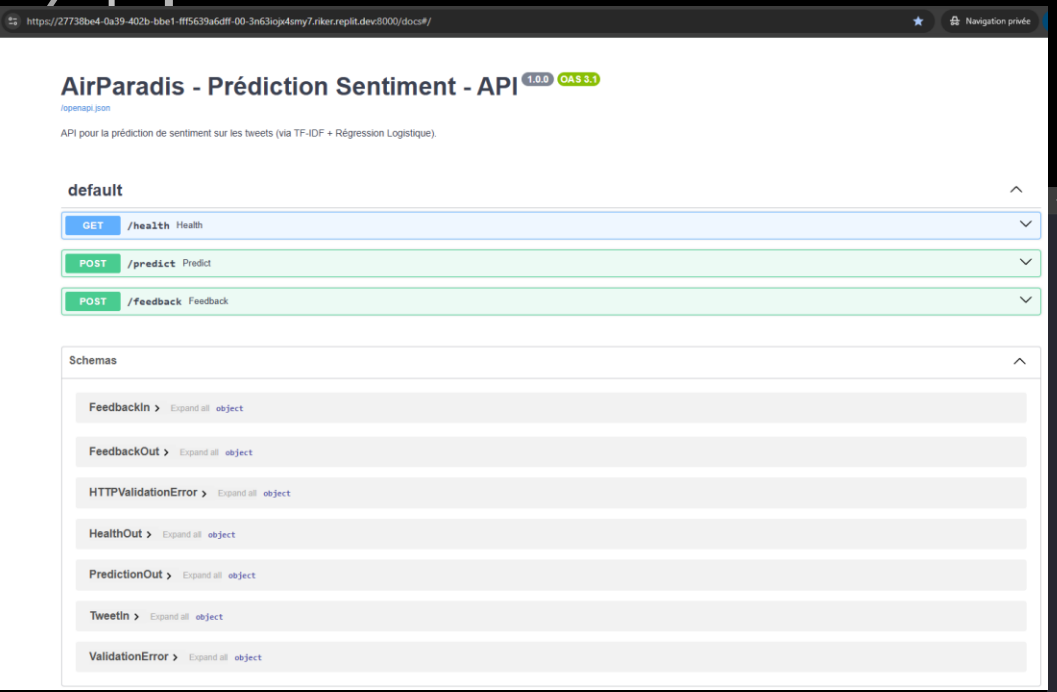
- API FastAPI :
    - modèle chargé au démarrage
    - schémas Pydantic typés

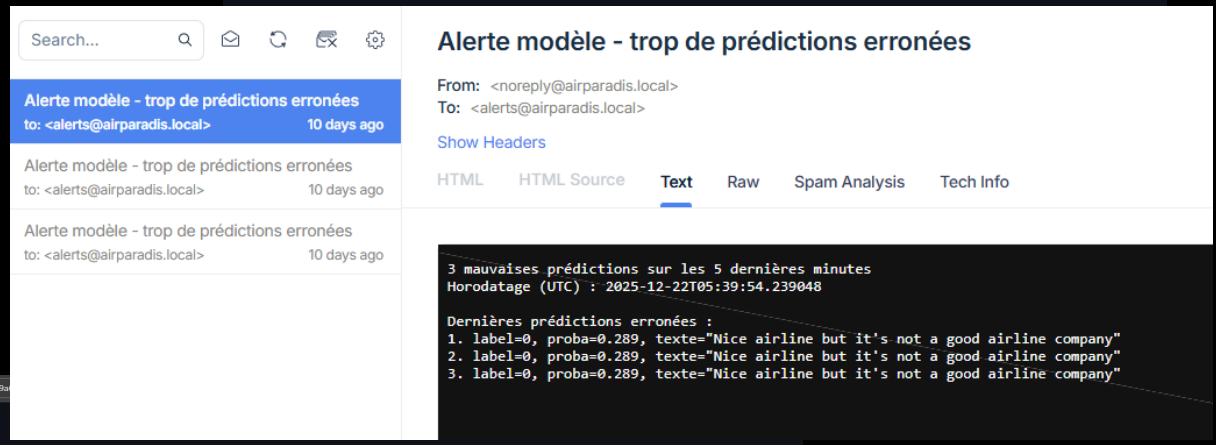
- Monitoring + alertes :
    - logs JSON feedback.log
    - stats exposées via endpoints
    - envoi d’email en cas de trop nombreuses erreurs

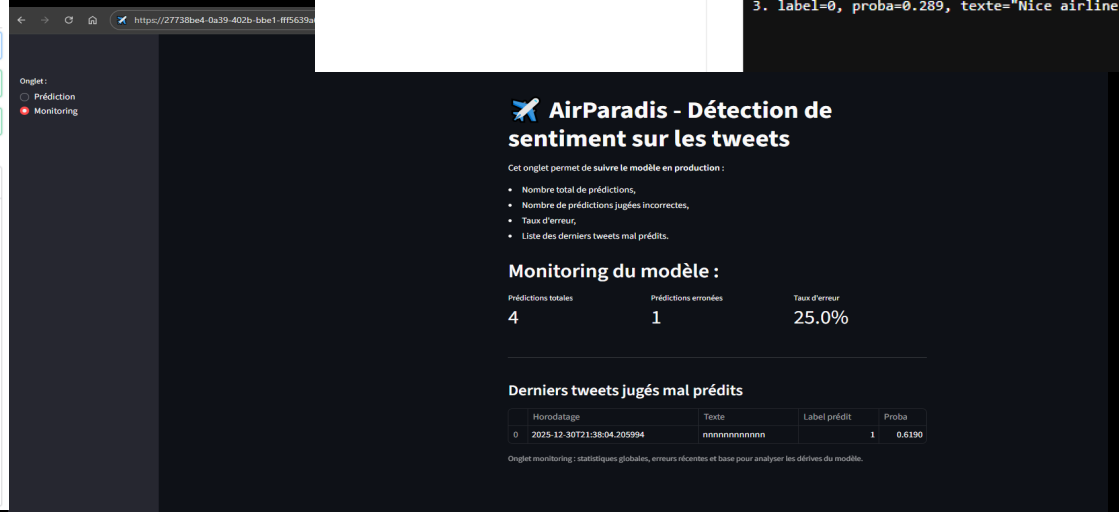
Architecture technique

- Streamlit (interface utilisateur pour prédiction et monitoring)
- API FastAPI (/predict, /feedback, /stats etc)
- Modèle TF-IDF + LogReg chargé depuis models/
- Logs écrits dans logs/feedback.log
- Email / Mailtrap pour les alertes

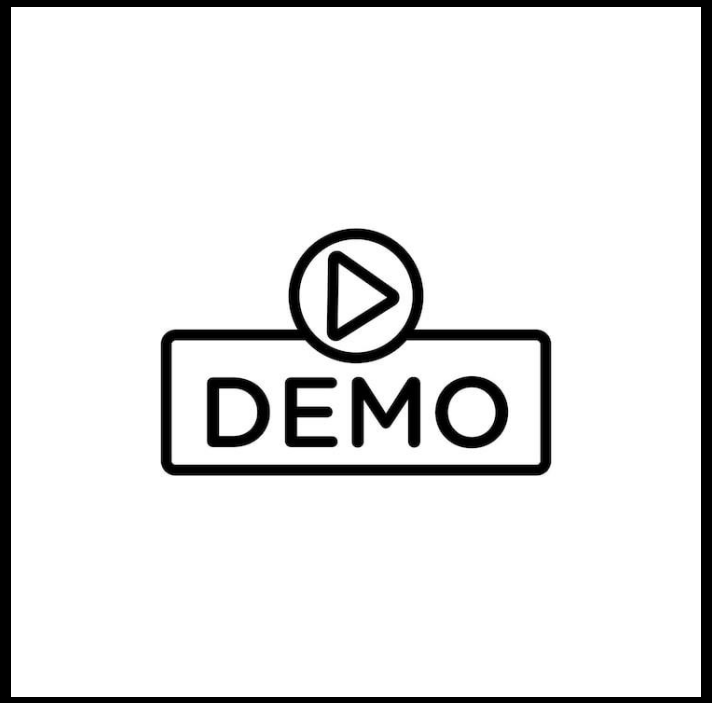




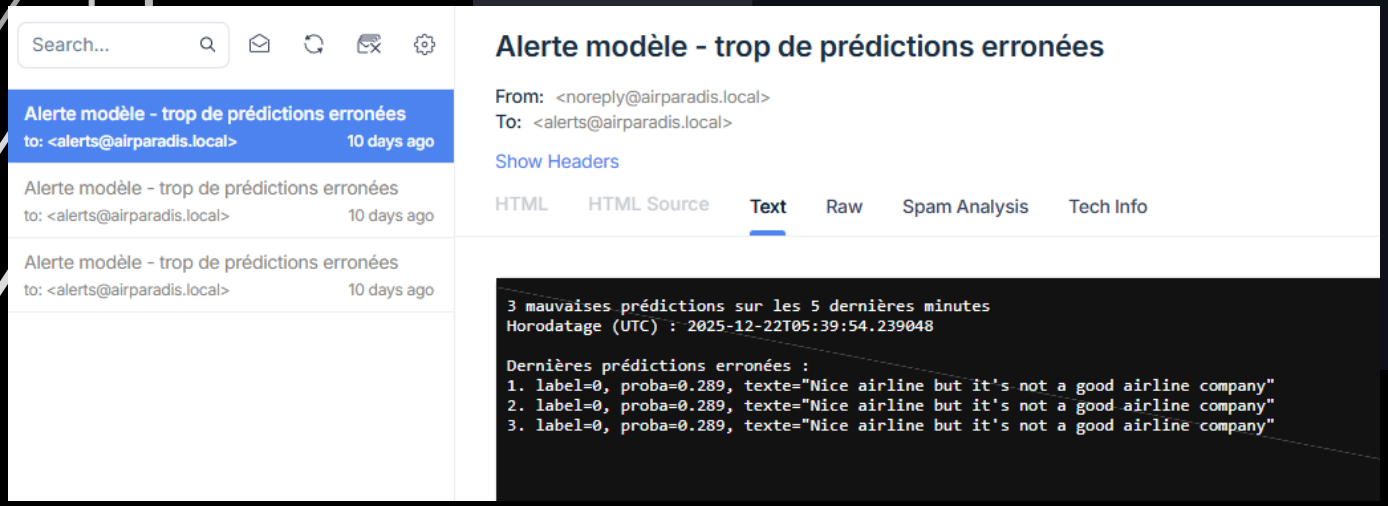


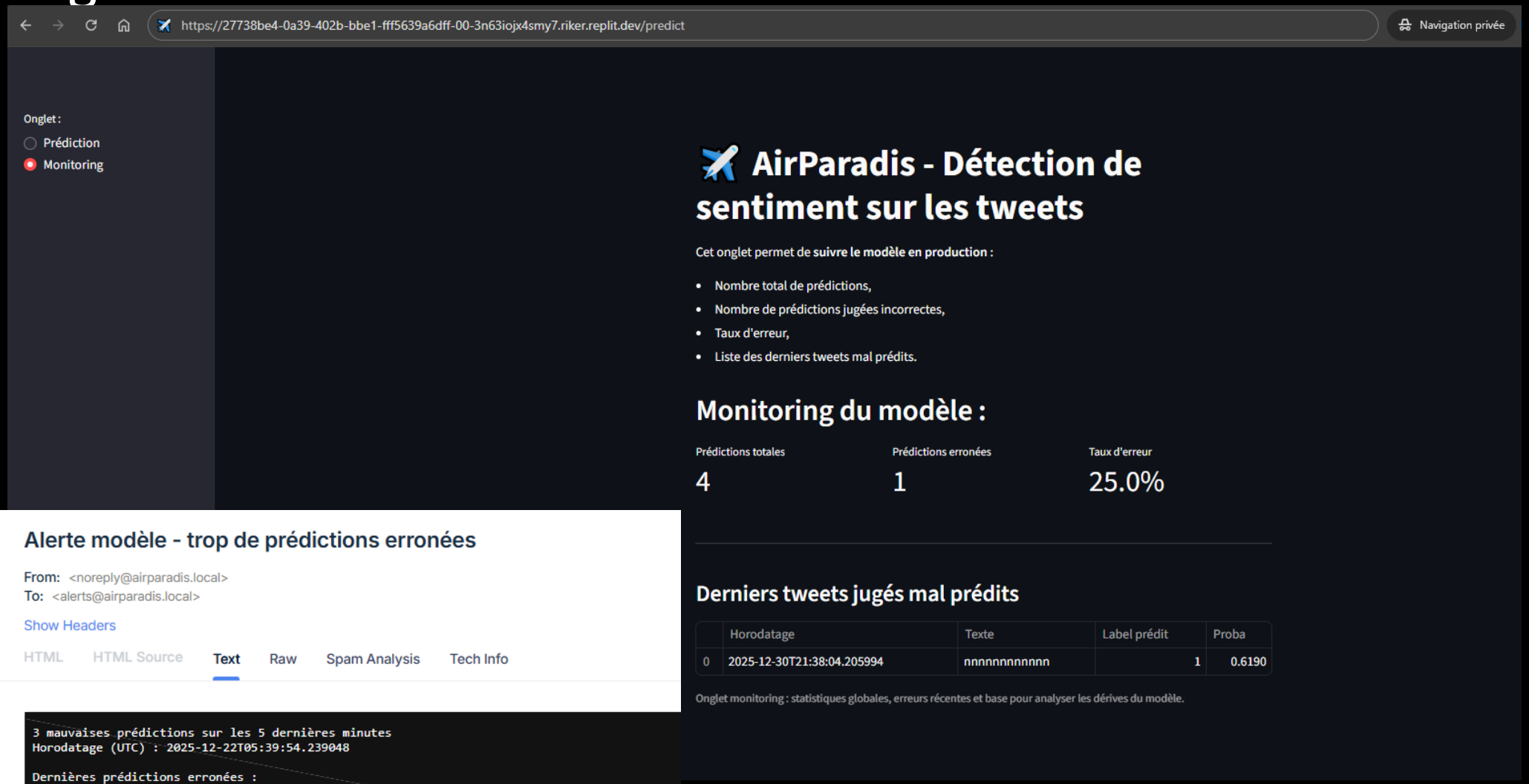


Démo live (API + Streamlit)



Monitoring & amélioration continue





Limites actuelles

- Entraînement BERT sur sous-échantillon (contraintes de ressources)

- Monitoring basique sur logs fichiers, et view via streamlit

- Modèle entraîné sur données de langue anglaises → pas encore multilingue

Merci