Anticiper les bad buzz sur Twitter : notre prototype de scoring de sentiment, de l'IA au MLOps

Dans ce projet, nous avons travaillé comme une équipe data / IA chargée d'aider "Air Paradis", une compagnie aérienne, à mieux anticiper les bad buzz sur Twitter.
- L'objectif : construire **un prototype complet** qui va :

- Prédire le sentiment d'un tweet (positif / négatif),
- Être exposé via une **API de prédiction**,
- Être testable via une petite application **Streamlit**,
- Et être accompagné d'une **vraie démarche MLOps** (tracking, tests, CI/CD, monitoring, alertes).

---

1. Données et exploration

Comme Air Paradis ne fournit pas de données internes, nous utilisons un dataset open source de référence : composé de **1,6 million de tweets** annotés en sentiment binaire. Chaque ligne contient :

- Un label binaire (0 = négatif, 4 = positif que nous remappons en 0 / 1),
- Un identifiant de tweet,
- La date,
- Un champ "flag",
- Le nom d'utilisateur,
- Le texte du tweet.

Ce que nous avons regardé dans l'EDA

Dans le notebook d'exploration, nous avons notamment :

- Vérifié la **répartition des classes** (environ 50 % négatif / 50 % positif),
- Contrôlé la présence de **valeurs manquantes** ou de lignes corrompues,
- Analysé la **longueur des tweets** (en mots) avant et après nettoyage,
- Mis en évidence quelques exemples de tweets très bruités (URLs, mentions, caractères répétitifs, etc.).

Nous avons également construit un premier jeu de **statistiques descriptives** sur les longueurs de tweets, par type de prétraitement, pour voir concrètement l'impact des différentes stratégies de 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

---

2. Prétraitement du texte : plusieurs niveaux selon le modèle

Nous avons centralisé le nettoyage du texte avec un script Python dédié (scripts/preprocessing.py) afin de pouvoir le réutiliser :

- Dans les notebooks de modélisation,
- Dans l'API de prédiction,
- Dans les tests unitaires.

Le pipeline est construit autour de **NLTK** (tokenisation, stopwords, lemmatisation), avec trois variantes.

2.1. Prétraitement "simple" (pour le modèle TF-IDF)

Pour le modèle classique (TF-IDF + régression logistique), nous utilisons un nettoyage assez agressif :

- Passage en minuscules,
- Suppression des URLs, mentions @user, hashtags isolés,
- Suppression des caractères non alphabétiques "bruités",
- Tokenisation,
- Retrait des stopwords,
- Lemmatisation.

Nous supprimons ensuite les tweets dont la version nettoyée a une longueur trop faible (moins de 2 mots), car ils sont généralement inexploitables (spam, suites de caractères, etc.).

2.2. Prétraitement "avancé" (pour réseau de neurones + embeddings)

Pour l'approche deep learning, nous conservons davantage d'information :

- Nettoyage des URLs/mentions,

- Normalisation minimale,
- Mais moins de suppression “agressive” pour garder du contexte.

Les tweets trop courts (après nettoyage) sont également filtrés, mais avec un seuil différent. L’idée est de garder un compromis entre qualité des données et richesse sémantique.

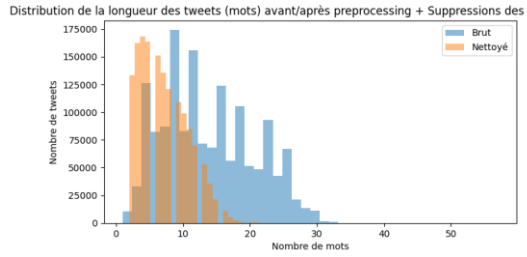
2.3. Prétraitement “BERT”

Pour le modèle basé sur ModernBERT, nous avons :

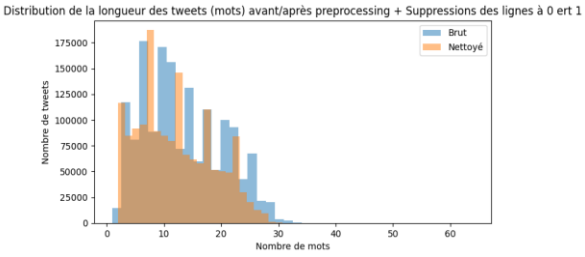
- Limité le nettoyage à ce qui est vraiment nécessaire (URLs, mentions évidentes),
- Laissé faire le tokenizer de BERT pour le reste.

Les tweets vides après nettoyage sont supprimés, mais ce cas est plus rare ici, puisque BERT sait mieux gérer des formes variées de texte.

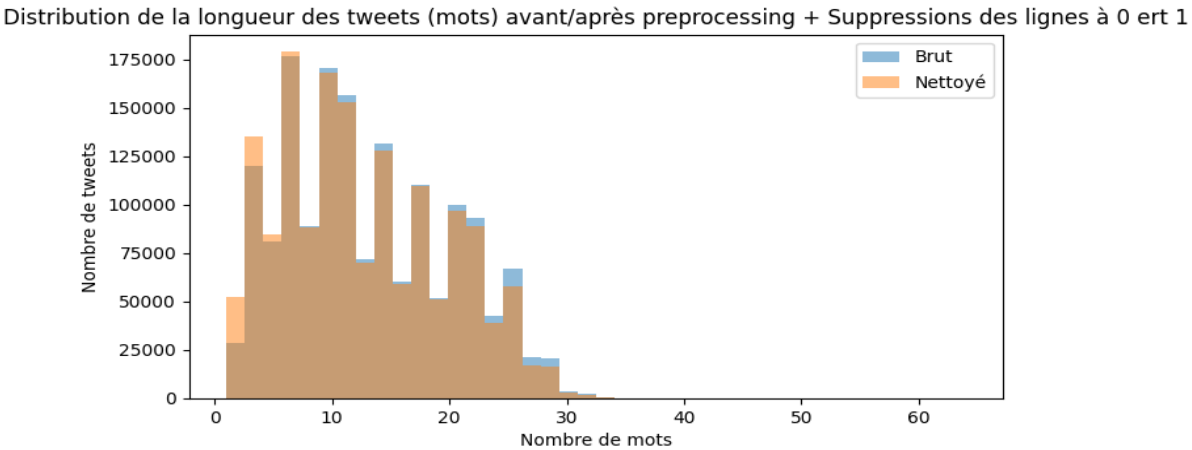
dataset	count	mean	std	min	25%	50%	75%	max
Base	1600000	13.18	6.96	1	7	12	19	64
Simple	1527316	6.97	3.60	2	4	6	10	33
Avancé	1579630	11.96	6.45	2	7	11	17	52
BERT	1597185	12.67	6.94	1	7	12	18	64



Simple



Avancer



Bert

3. Modèle 1 – Baseline TF-IDF + régression logistique

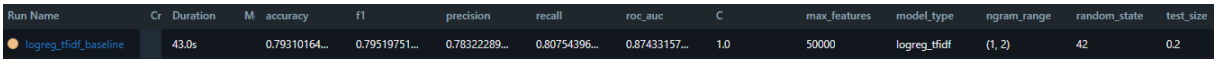
La première approche est volontairement simple, mais solide :

- Vectorisation des tweets via **TF-IDF** (unigrammes, éventuellement bigrammes),
- Modèle de **régression logistique**,
- Entraînement supervisé avec un **split train/test fixé** et réutilisé pour tous les modèles (traçé dans un fichier séparé pour garder une base de comparaison constante),
- Métriques : accuracy, precision, recall, F1-score, matrice de confusion.

Cette approche a plusieurs avantages :

- Le pipeline est facilement sérialisable avec joblib,
- Le temps d'entraînement est court,
- La taille du modèle reste raisonnable (important pour un déploiement sur la plateforme gratuite).

C'est ce modèle que nous avons finalement choisi pour le déploiement dans l'API, justement parce qu'il est léger, stable et explicable.



Run Name	Cr	Duration	M	accuracy	f1	precision	recall	roc_auc	C	max_features	model_type	ngram_range	random_state	test_size
logreg_tfidf_baseline		43.0s		0.79310164...	0.79519751...	0.78322289...	0.80754396...	0.87433157...	1.0	50000	logreg_tfidf	(1, 2)	42	0.2

	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

Chaque ligne correspond à une classe de sentiment (**0 = négatif**, **1 = positif**).

- **Precision** : parmi les tweets prédits comme *positifs/négatifs*, combien sont corrects.
- **Recall** : parmi les tweets *réellement* positifs/négatifs, combien sont retrouvés par le modèle.
- **F1-score** : équilibre entre precision et recall (plus c'est proche de **1**, mieux c'est).
- **Support** : nombre de tweets évalués dans chaque classe.
- **Accuracy** : proportion totale de prédictions correctes.
- **Macro avg** : moyenne des scores entre classes (chaque classe compte pareil).
- **Weighted avg** : moyenne pondérée par le nombre d'exemples (support).

4. Modèle 2 – Réseau de neurones avec embeddings (approche avancée)

Pour l'approche "modèle sur mesure avancé", nous avons construit un réseau de neurones utilisant des **word embeddings pré-entraînés**. L'idée était de tester l'apport d'une représentation plus riche du texte.

4.1. Embeddings

Nous avons travaillé avec :

- Un embedding pré-entraîné de type **GloVe**,
- Un deuxième embedding de Type Fastext.

Les embeddings sont chargés, puis intégrés dans une couche Embedding (avec ou sans gel des poids selon l'essai), et le texte est représenté par une **séquence d'indices de tokens**.

4.2. Architecture

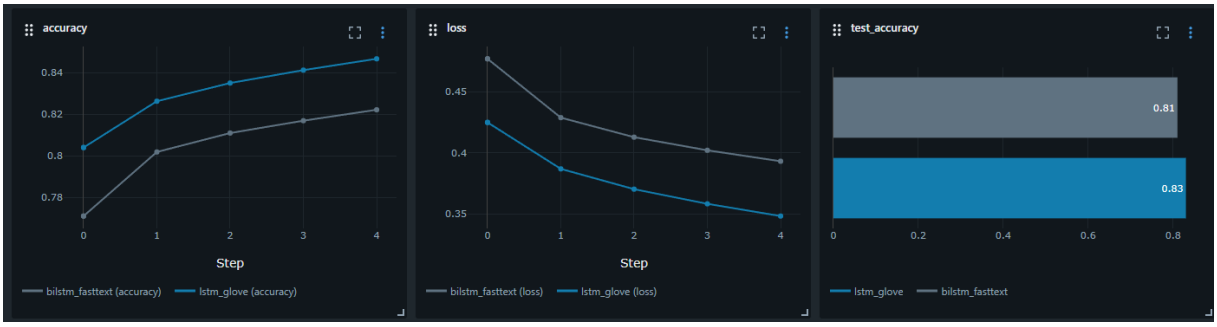
L'architecture testée suit un schéma assez classique :

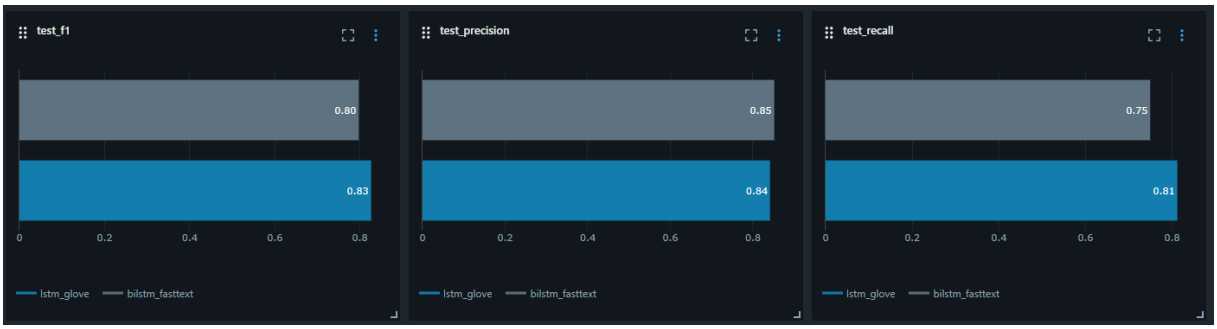
- Couche Embedding initialisée avec les poids GloVe,
- Une ou plusieurs couches LSTM,
- Éventuellement une couche de dropout pour limiter le surapprentissage,
- Couche dense finale avec sortie sigmoïde pour la classification binaire.

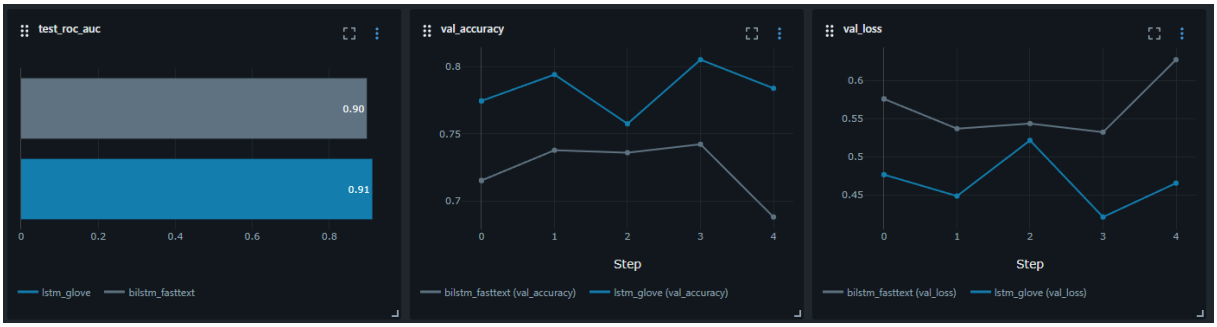
L'entraînement se fait en supervision sur le même split train/test que la baseline, avec suivi des métriques validation (accuracy, F1 … etc).

Les résultats montrent un **gain modéré** par rapport au modèle TF-IDF, au prix d'un entraînement plus exigeant en ressources et plus long. Ce type de modèle devient intéressant si l'entreprise est prête à investir dans une infrastructure un peu plus sérieuse (GPU, environnement cloud persistant).

Run Name	Cr	Duration	accuracy	loss	test_accuracy	test_f1	test_precision	test_recall	test_roc_auc	val_accuracy	val_loss	validation_accur	validation_loss	baseline	batch_size
bilstm_fasttext		16.6min	0.82215416...	0.39307889...	0.81047848...	0.79734233...	0.85162419...	0.74956560...	0.89735719...	0.68818032...	0.62765270...	0.68818032...	0.62765270...	None	256
lstm_glove		12.0min	0.84658253...	0.34818631...	0.83040554...	0.82650645...	0.84136421...	0.81216433...	0.91162285...	0.78378862...	0.46533721...	0.78378862...	0.46533721...	None	256







---

5. Modèle 3 – ModernBERT : mesurer l'apport d'un Transformer

Pour la troisième approche, nous avons utilisé un modèle de type **ModernBERT**, afin d'évaluer le niveau de performance qu'on peut attendre d'un Transformer pour ce cas d'usage.

5.1. Préparation des données

- Utilisation d'un Dataset compatible Transformers (Hugging Face),
- Tokenisation avec le tokenizer ModernBERT (troncature, padding, gestion des séquences),

- Création de DataLoaders pour l'entraînement et la validation.

Par contrainte de temps et de ressources (CPU uniquement et budget limité sur l'environnement), nous avons entraîné le modèle sur un **sous-échantillon contrôlé** du dataset, plutôt que sur les 1,6 million de tweets. (3 Tests effectués de 1000 à 50 000, une augmentation de l'échantillon accroit exponentiellement toutes les valeurs en positif, mais consomme beaucoup plus de ressource et durent plus longtemps)

5.2. Entraînement et résultats

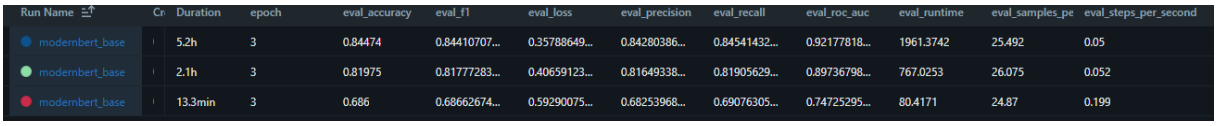
L'entraînement se fait via Trainer / TrainingArguments :

- Batch size adaptée à la RAM disponible,
- Nombre d'époch limité,
- Suivi de la loss et des métriques sur le jeu de validation.

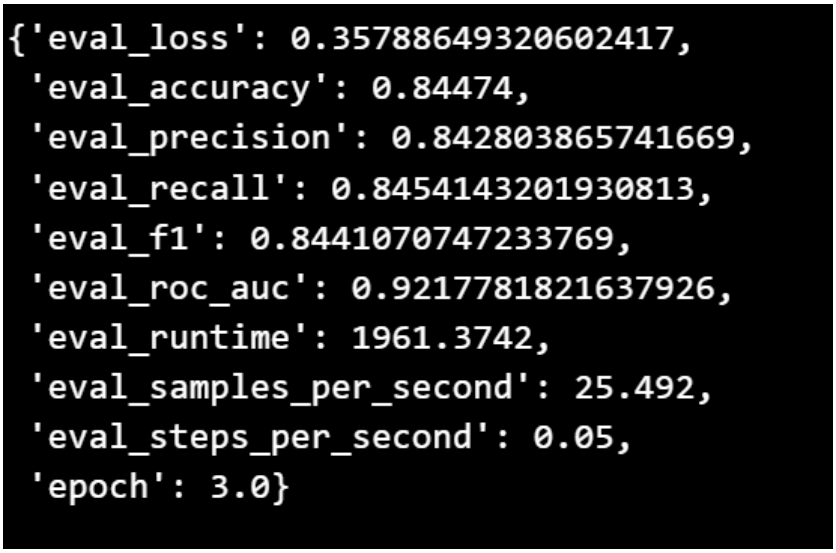
Sans surprise, ce modèle offre des **performances très bonnes** sur le dataset, mais :

- Il est beaucoup plus lourd,
- La latence d'inférence est plus élevée,
- Le déploiement nécessite un environnement plus musclé.

Dans le cadre du prototype, nous utilisons donc cette approche comme **référence de performance**, mais nous ne la déployons pas dans l'API finale pour rester conformes aux contraintes de coût.

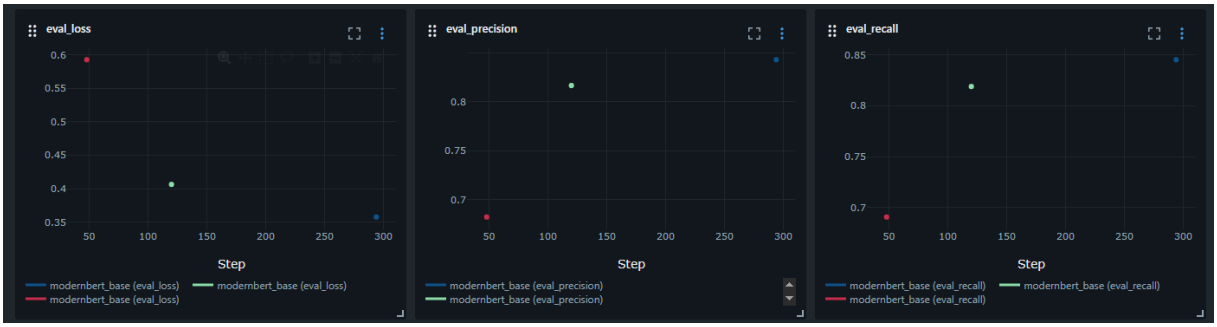


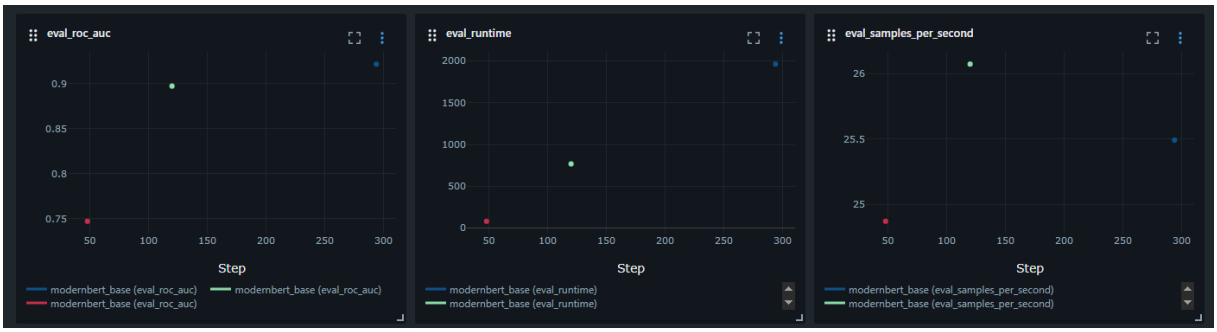
Run Name	Cr	Duration	epoch	eval_accuracy	eval_f1	eval_loss	eval_precision	eval_recall	eval_roc_auc	eval_runtime	eval_samples_pe	eval_steps_per_second
modernbert_base		5.2h	3	0.84474	0.84410707...	0.35788649...	0.84280386...	0.84541432...	0.92177818...	1961.3742	25.492	0.05
modernbert_base		2.1h	3	0.81975	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



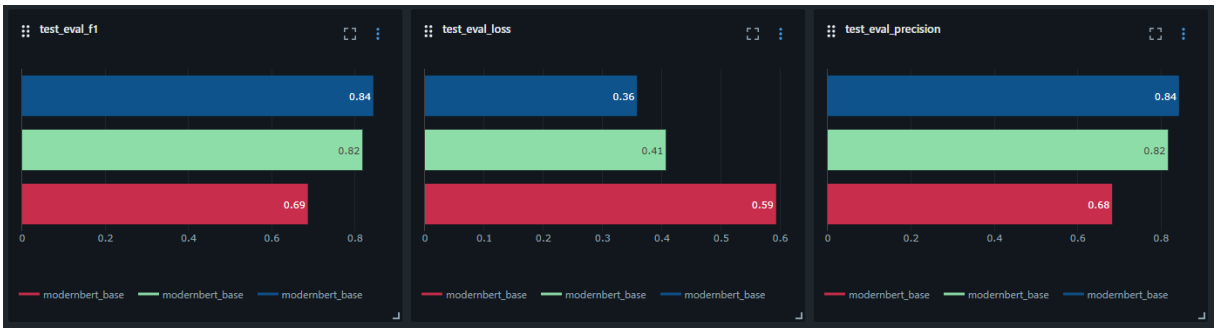
```
{'eval_loss': 0.35788649320602417,
 'eval_accuracy': 0.84474,
 'eval_precision': 0.8428038657416 69,
 'eval_recall': 0.8454143201930813,
 'eval_f1': 0.8441070747233769,
 'eval_roc_auc': 0.9217781821637926,
 'eval_runtime': 1961.3742,
 'eval_samples_per_second': 25.492,
 'eval_steps_per_second': 0.05,
 'epoch': 3.0}
```

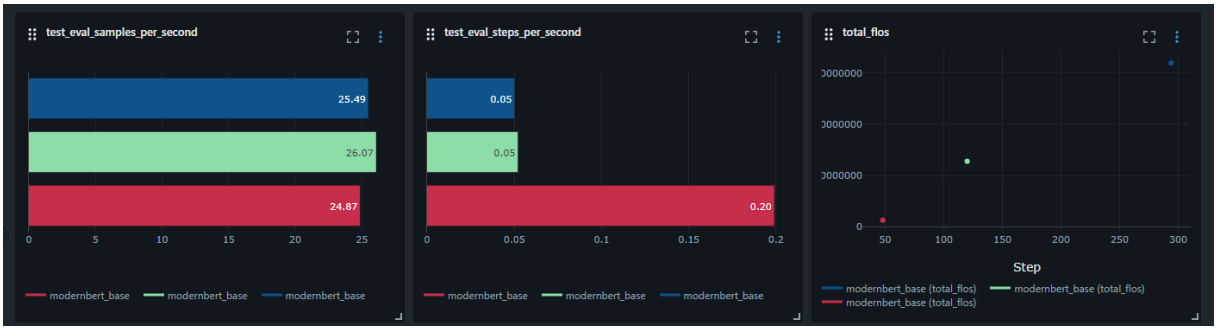

epoch
Step
modernbert_base (epoch)
modernbert_base (epoch)
modernbert_base (epoch)
eval_accuracy
Step
modernbert_base (eval_accuracy)
modernbert_base (eval_accuracy)
eval_f1
Step
modernbert_base (eval_f1)
modernbert_base (eval_f1)
modernbert_base (eval_f1)

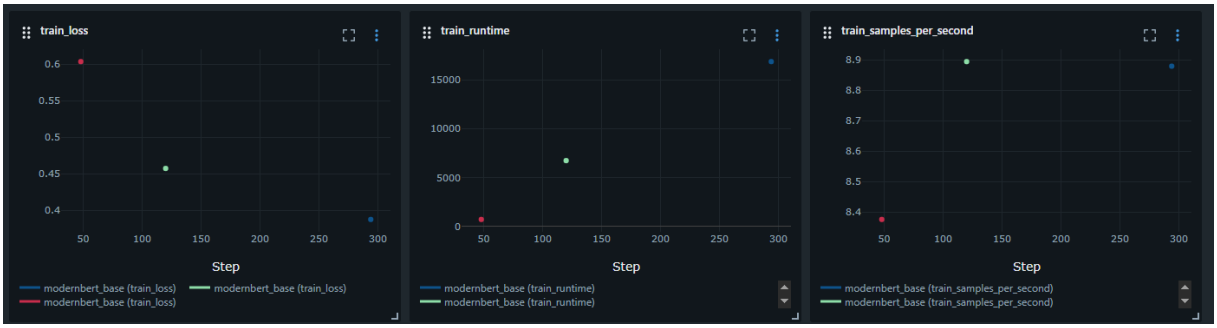

eval_loss
Step
modernbert_base (eval_loss)
modernbert_base (eval_loss)
modernbert_base (eval_loss)
eval_precision
Step
modernbert_base (eval_precision)
modernbert_base (eval_precision)
eval_recall
Step
modernbert_base (eval_recall)
modernbert_base (eval_recall)
modernbert_base (eval_recall)

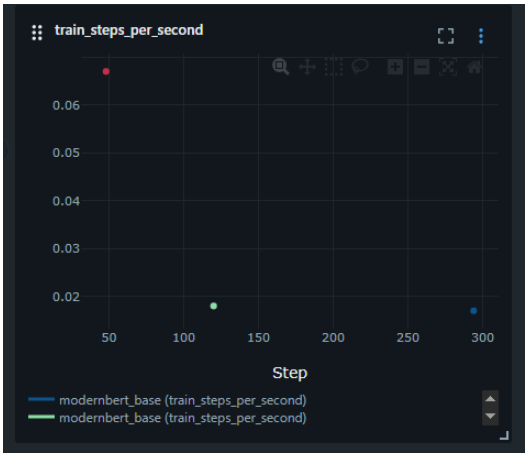

eval_roc_auc
Step
modernbert_base (eval_roc_auc)
modernbert_base (eval_roc_auc)
modernbert_base (eval_roc_auc)
eval_runtime
Step
modernbert_base (eval_runtime)
modernbert_base (eval_runtime)
eval_samples_per_second
Step
modernbert_base (eval_samples_per_second)
modernbert_base (eval_samples_per_second)


eval_steps_per_second
Step
modernbert_base (eval_steps_per_second)
modernbert_base (eval_steps_per_second)
test_epoch
3.00
3.00
3.00
modernbert_base
modernbert_base
modernbert_base
test_eval_accuracy
0.84
0.82
0.69
modernbert_base
modernbert_base
modernbert_base


test_eval_f1
0.84
0.82
0.69
modernbert_base
modernbert_base
modernbert_base
test_eval_loss
0.36
0.41
0.59
modernbert_base
modernbert_base
modernbert_base
test_eval_precision
0.84
0.82
0.68
modernbert_base
modernbert_base
modernbert_base







---

6. Comparaison des approches et choix du modèle en production

Pour comparer les modèles, nous avons réuni les résultats dans un tableau de synthèse (dans un notebook dédié), avec :

- accuracy,
- precision, recall, F1,
- Taille du modèle,
- Temps d'entraînement approximatif,
- Facilité de déploiement.

Les grandes tendances :

- **TF-IDF + régression logistique** :
  – Performances déjà très correctes,
  – Modèle compact,
  – Très simple à déployer,
  – Aucun besoin de GPU.
- **Embeddings + LSTM** :
  – Meilleure capture du sens,
  – Gain de performance possible selon la configuration,
  – Mais coût d'entraînement et de déploiement plus élevé.
- **ModernBERT** :
  – Meilleur candidat en termes de qualité brute,
  – Mais beaucoup plus exigeant (RAM, temps, taille du modèle),
  – Peu adapté à un déploiement "low cost" sur une petite infrastructure gratuite.

Pour un prototype destiné à tourner sur des ressources limitées, **nous avons donc choisi de déployer le modèle TF-IDF + régression logistique**.
Les deux autres approches servent de **référence** et de base de réflexion pour une éventuelle montée en puissance.

model_label	experiment	run_id	score_for_sort	test_accuracy	test_f1
ModernBERT-base	sentiment_airparadis_bert	3c865f091fc0	0.8441070747233769	0.84474	0.8441070747
bilstm_fasttext_cc_300d	sentiment_airparadis_modele_avan	c765b13efe2b	0.7973423322504761	0.8104784851897441	0.7973423322
logreg_tfidf	sentiment_airparadis_modele_simp	5c675ade24f8	0.7951975112608963	0.7931016420920305	0.7951975112
lstm_glove_twitter_100d	sentiment_airparadis_modele_avan	57ddba0ab02	0.8265064517965566	0.8304055469711652	0.8265064517

test_precision	test_recall	test_roc_auc	train_loss	val_loss	val_accuracy
0.842803865741669	0.84541432019308	0.9217781821637926	0.387263161795479	0.3578864932060241	*Missing value*
0.851624192390524	0.74956560657118	0.8973571987590773	0.393078893423080	0.6276527047157288	0.6881803274154663
0.7832228988726748	0.80754396588037(	0.8743315727541261	*Missing value*	*Missing value*	*Missing value*
0.8413642159235799	0.81216433235046	0.9116228564071932	0.348186314105987	0.4653721685409540	0.7837886214256287

7. Mise en place d'une démarche MLOps

Au-delà des modèles, nous avons pris le projet comme une occasion de mettre en place une **démarche MLOps structurée**.

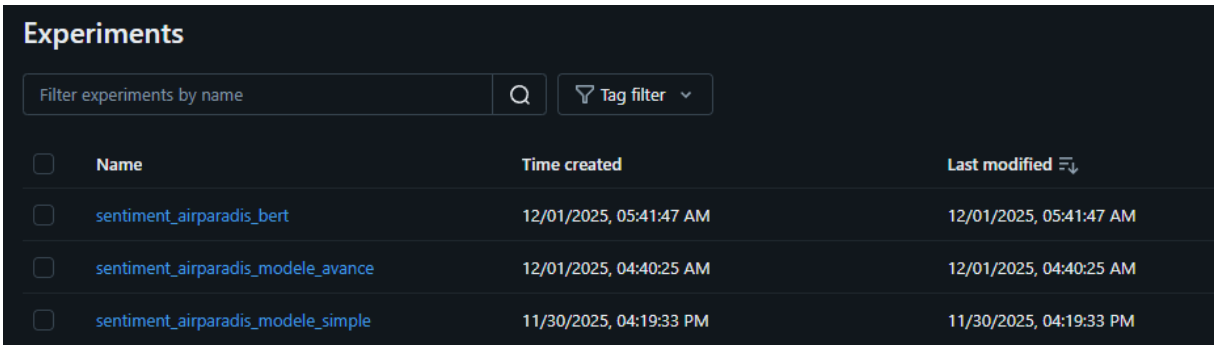
7.1. Tracking et gestion des modèles avec MLflow

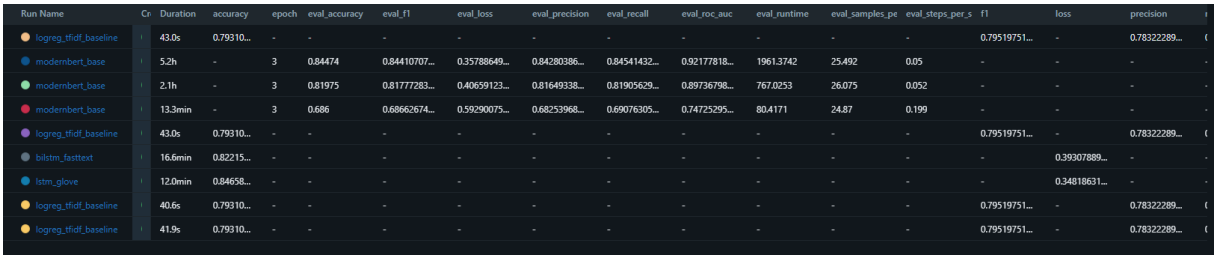
Nous utilisons MLflow pour :

- Tracer chaque expérience (paramètres, métriques, tags),
- Comparer rapidement les runs (baseline vs embeddings vs BERT),
- Sauvegarder les modèles comme "artifacts",
- Garder un historique propre des évolutions.

Un simple coup d'œil à l'interface MLflow permet de :

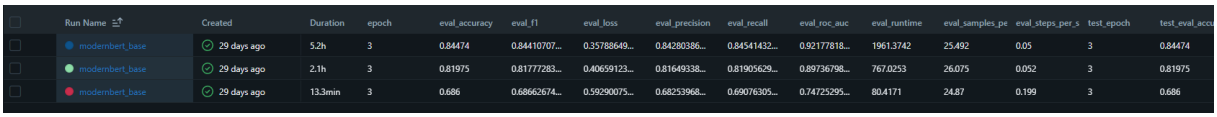
- Voir quel modèle est le meilleur sur le jeu de test,
- Retrouver les hyperparamètres associés,
- Documenter les décisions (par exemple : pourquoi nous avons choisi la baseline pour le déploiement).



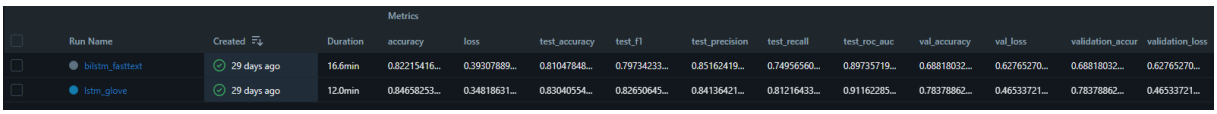


Avec Trie des Datas :

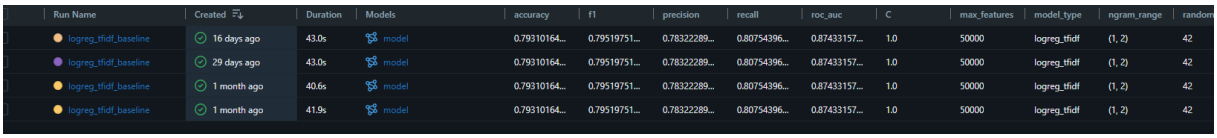
model_label	experiment	run_id	score_for_sort	test_accuracy	test_f1	test_precision
ModernBERT-base	sentiment_airparadis_bert	3c865f091fc0	0.8441070747233769	0.84474	0.8441070747	0.84280386574166
bilstm_fasttext_cc_300d	sentiment_airparadis_modele_avan	c765b13efe2b	0.7973423322504761	0.8104784851897441	0.7973423322	0.85162419239052
logreg_tfidf	sentiment_airparadis_modele_simp	5c675ade24f8	0.7951975112608963	0.7931016420920305	0.7951975112	0.783222898872674
lstm_glove_twitter_100d	sentiment_airparadis_modele_avan	57ddba0ab02	0.8265064517965566	0.8304055469711652	0.8265064517	0.841364215923579



Run Name	Created	Duration	epoch	eval_accuracy	eval_f1	eval_loss	eval_precision	eval_recall	eval_roc_auc	eval_runtime	eval_samples_pe	eval_steps_per_s	test_epoch	test_eval_accu
modernbert_base	29 days ago	5.2h	3	0.84474	0.84410707...	0.35788649...	0.84280386...	0.84541432...	0.92177818...	1961.3742	25.492	0.05	3	0.84474
modernbert_base	29 days ago	2.1h	3	0.81975	0.81777283...	0.40659123...	0.81649338...	0.81905629...	0.89736798...	767.0253	26.075	0.052	3	0.81975
modernbert_base	29 days ago	13.3min	3	0.686	0.68662674...	0.59290075...	0.68253968...	0.69076305...	0.74725295...	80.4171	24.87	0.199	3	0.686



Run Name	Created	Duration	Metrics									
			accuracy	loss	test_accuracy	test_f1	test_precision	test_recall	test_roc_auc	val_accuracy	val_loss	validation_accur	validation_loss
bilstm_fasttext	29 days ago	16.6min	0.82215416...	0.39307889...	0.81047848...	0.79734233...	0.85162419...	0.74956560...	0.89735719...	0.68818032...	0.6276527...	0.68818032...	0.62765270...
lstm_glove	29 days ago	12.0min	0.84658253...	0.34818631...	0.83040554...	0.82650645...	0.84136421...	0.81216433...	0.91162285...	0.78378862...	0.46533721...	0.78378862...	0.46533721...



Run Name	Created	Duration	Models	accuracy	f1	precision	recall	roc_auc	C	max_features	model_type	ngram_range	random
logreg_tfidf_baseline	16 days ago	43.0s	model	0.79310164...	0.79519751...	0.78322289...	0.80754396...	0.87433157...	1.0	50000	logreg_tfidf	(1, 2)	42
logreg_tfidf_baseline	29 days ago	43.0s	model	0.79310164...	0.79519751...	0.78322289...	0.80754396...	0.87433157...	1.0	50000	logreg_tfidf	(1, 2)	42
logreg_tfidf_baseline	1 month ago	40.6s	model	0.79310164...	0.79519751...	0.78322289...	0.80754396...	0.87433157...	1.0	50000	logreg_tfidf	(1, 2)	42
logreg_tfidf_baseline	1 month ago	41.9s	model	0.79310164...	0.79519751...	0.78322289...	0.80754396...	0.87433157...	1.0	50000	logreg_tfidf	(1, 2)	42

7.2. Versionning du code et structure du dépôt

Tout le projet est versionné sur GitHub, avec une arborescence claire :

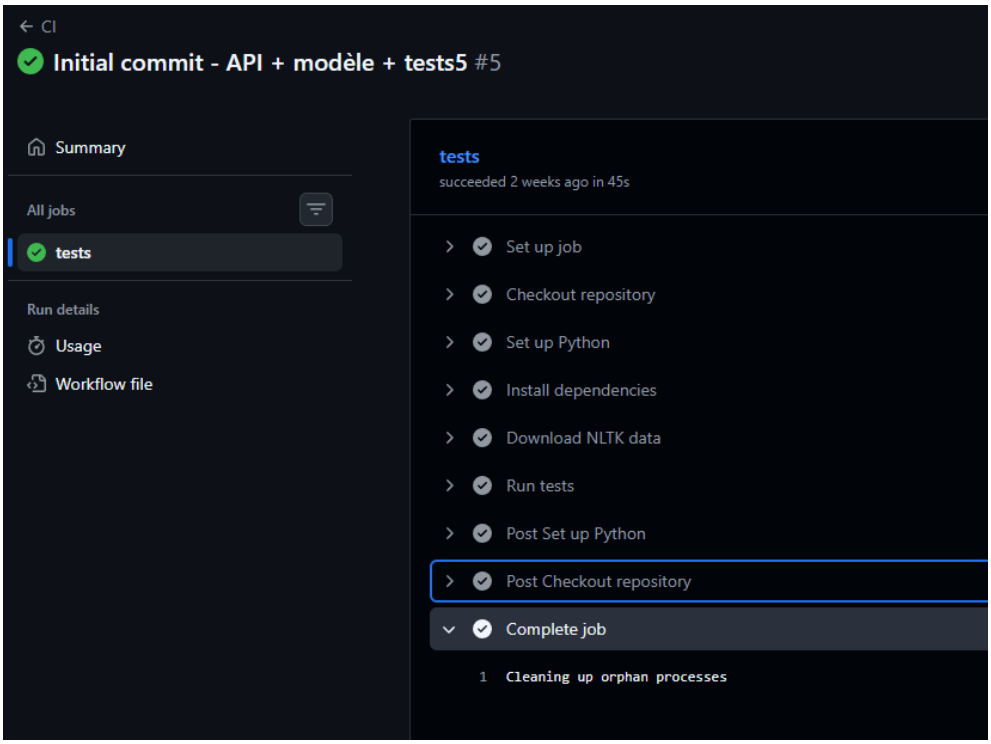
- data/ (données locales, embeddings et data csv des 1.6 millions de tweets),
- notebooks/ (EDA, modèles simple/avancé/BERT, comparaison),
- scripts/ (prétraitement réutilisable),
- api/ (FastAPI, chargement du modèle, schémas Pydantic),
- app/ (interface Streamlit),
- tests/ (tests unitaires),
- logs/ (logs structurés pour le monitoring),
- fichiers de configuration (requirements.txt, .replit, .env, etc …).
- models/ (artifact joblib tfidf)
- out/ (split csv)

7.3. Tests unitaires et CI/CD

Nous avons ajouté des tests pytest pour :

- Vérifier que l'API répond correctement (/health, /predict, /feedback),
- Valider le chargement du modèle sérialisé,
- S'assurer que le prétraitement fait bien ce qu'on attend (nettoyage d'URL, de mentions, etc.).

Un workflow GitHub Actions lance automatiquement ces tests à chaque push.
Si un test échoue, le pipeline passe en rouge, ce qui évite de déployer une version cassée.

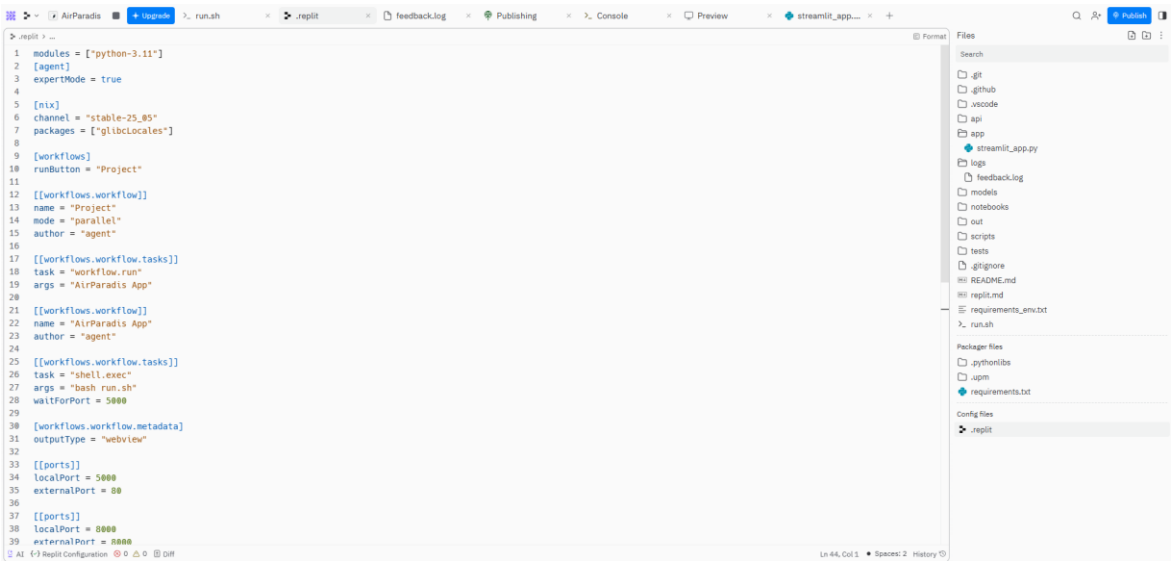


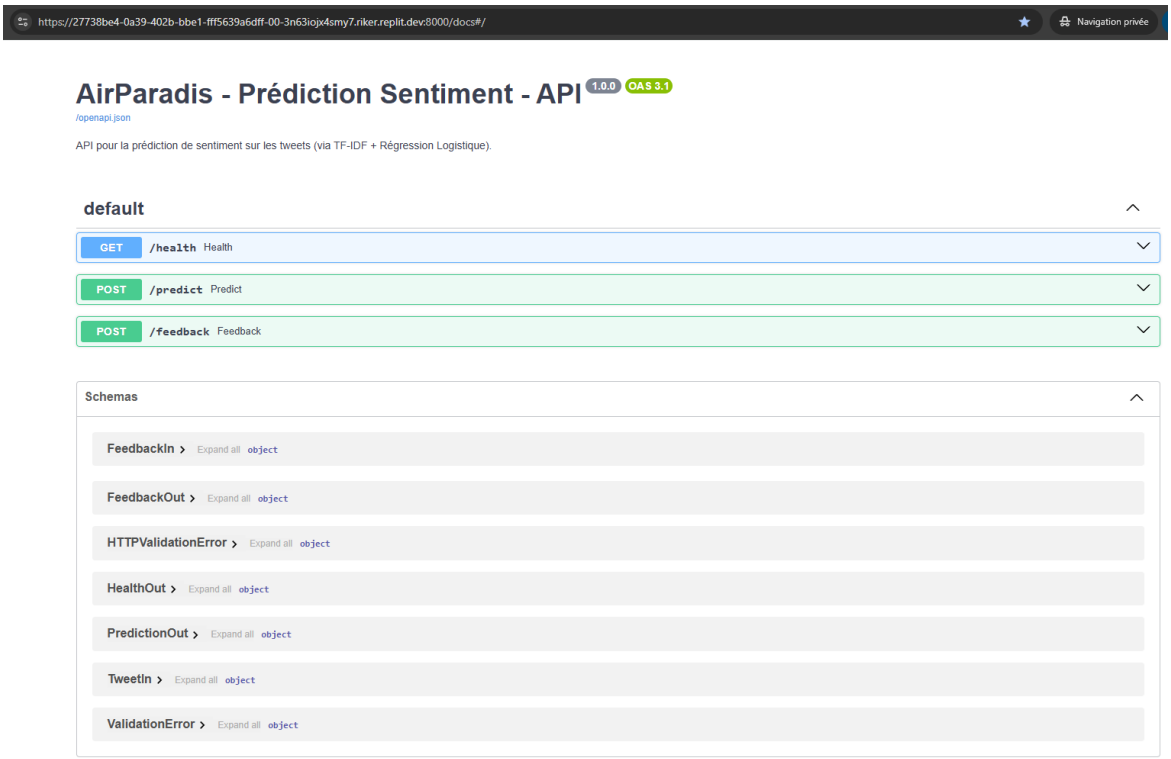
7.4. Déploiement de l'API et de l'interface

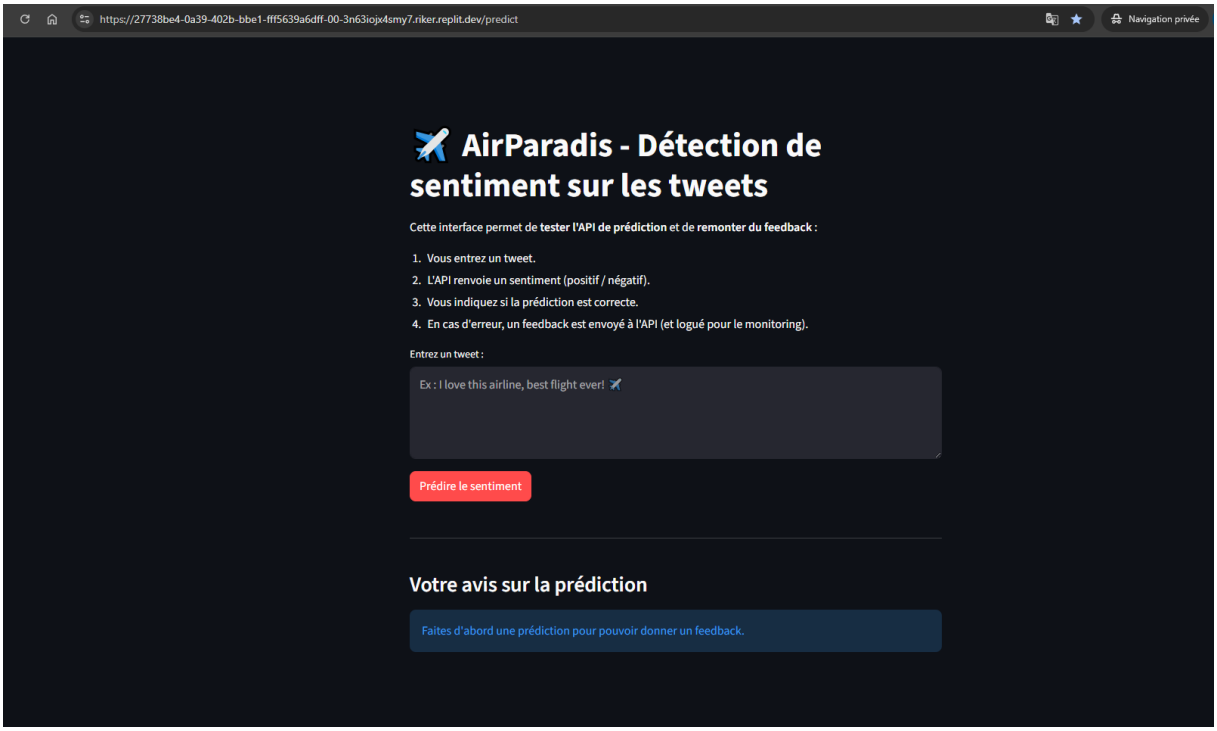
L'API FastAPI et l'interface Streamlit sont démarrées ensemble via un script de lancement :

- FastAPI expose les endpoints métier,
- Streamlit sert d'interface utilisateur pour tester le modèle et remonter du feedback,
- Le modèle TF-IDF est chargé au démarrage pour éviter le coût de chargement à chaque requête.

Même si nous utilisons une plateforme type Replit dans ce projet, la même logique pourrait être appliquée sur un cloud classique (Azure Web App, AWS, etc.) avec très peu de modifications.







---

8. Monitoring et alertes : fermer la boucle MLOps

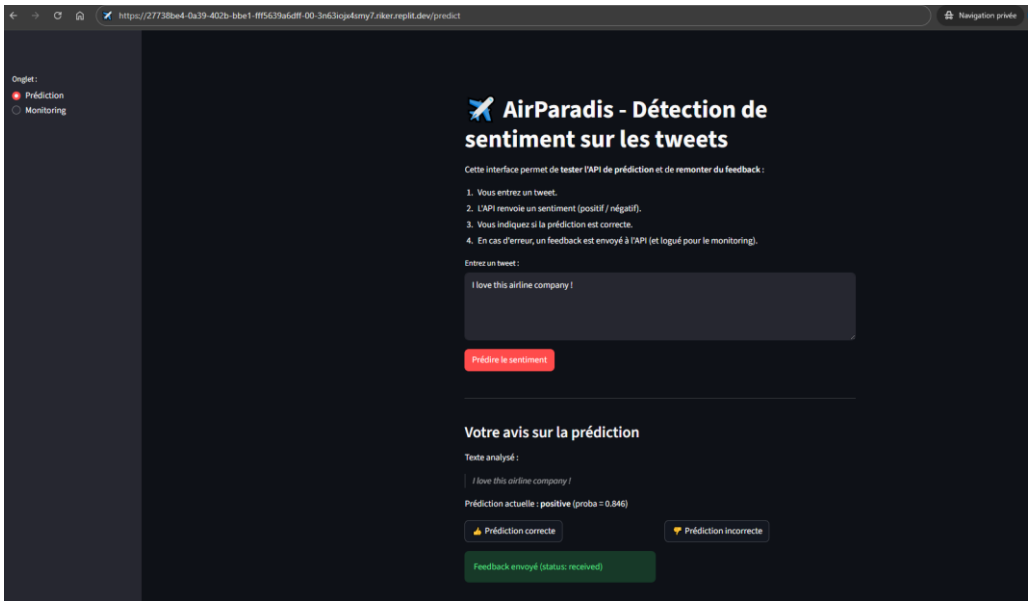
La dernière brique importante du projet concerne le **suivi du modèle en production**. Le cahier des charges parlait d'Azure Application Insights ; faute de pouvoir l'utiliser directement, nous avons mis en place un équivalent léger :

8.1. Collecte de feedback utilisateur

Chaque fois qu'un utilisateur teste un tweet dans Streamlit :

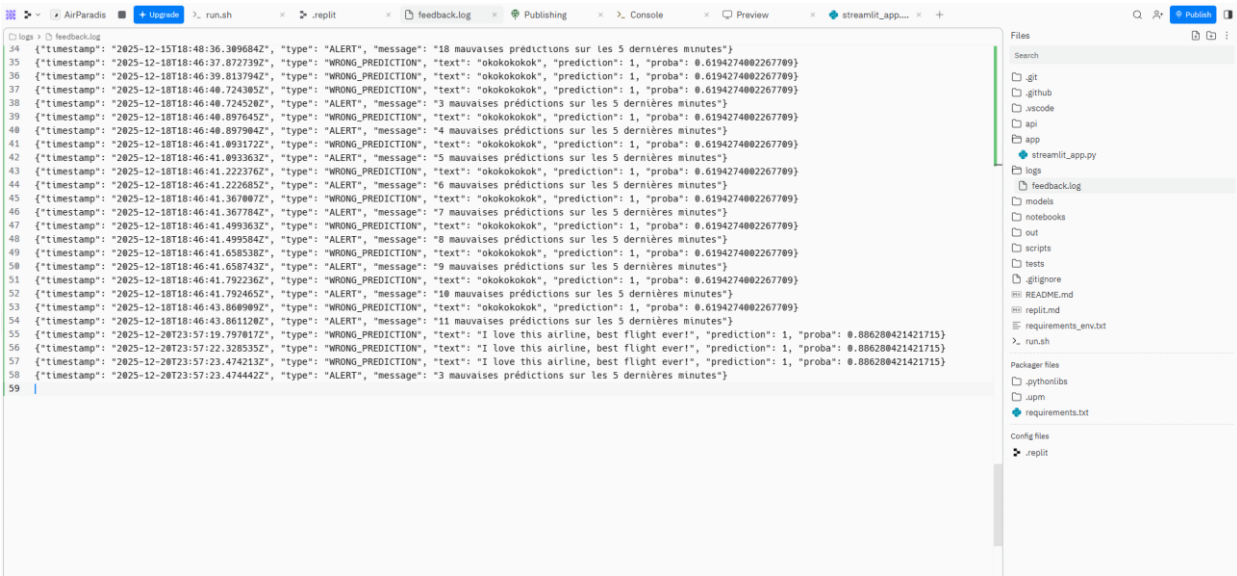
1. L'API renvoie une prédiction (label + probabilité).
2. L'interface affiche la réponse.
3. L'utilisateur peut ensuite cliquer sur :

➔ "Prédiction correcte" ou "Prédiction incorrecte".



En cas d'erreur signalée :

- Un appel est envoyé à l'endpoint /feedback,
- L'API logue un événement de type WRONG_PREDICTION dans un fichier feedback.log,
- Le tweet, la prédiction et la probabilité sont sauvegardés.



8.2. Statistiques et visualisation

L'API maintien des compteurs :

- Nombre total de prédictions,
- Nombre total de prédictions incorrectes (d'après le feedback),
- Taux d'erreur global.

Elle expose aussi la liste des derniers tweets mal prédits.

Dans Streamlit, un onglet "Monitoring" permet de :

- Afficher ces statistiques en temps réel,
- Lister les exemples récents jugés incorrects.

C'est l'équivalent d'un petit tableau de bord de production, mais autour de notre API.

8.3. Alerte email en cas de dérive

Nous avons ajouté une règle simple :

- Si au moins **3 mauvaises prédictions** sont signalées dans une fenêtre de **5 minutes**,
- L'API génère une entrée d'alerte dans feedback.log,
- Puis déclenche un **envoi de mail** via un serveur SMTP de test (Mailtrap).

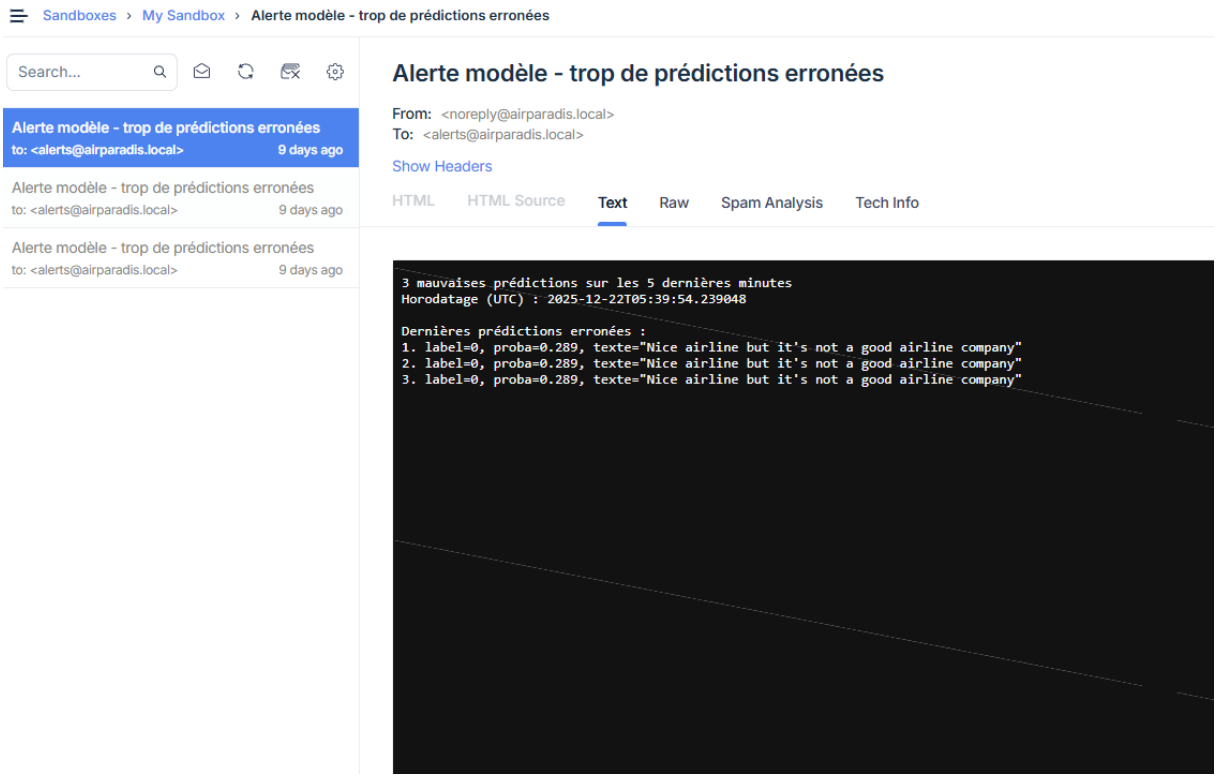
Le mail contient :

- Un résumé de l'alerte,

- L’horodatage,
- Un petit récapitulatif des derniers tweets concernés (label, probabilité, extrait du texte).

Dans un contexte réel, ce mécanisme pourrait être branché :

- Soit sur une boîte mail dédiée (“monitoring@airparadis.com”),
- Soit sur un canal Teams / Slack via webhook,
- Soit intégré à un système de supervision plus large.



---

9. Vers l’amélioration continue du modèle

Grâce aux logs et au feedback utilisateur, nous avons une base concrète pour améliorer le modèle dans le temps :

1. **Identifier les cas difficiles** : tweets souvent mal classés, nouveau vocabulaire, scénarios spécifiques (retards, bagages, service client).
2. **Labeliser ces exemples** correctement (en interne chez Air Paradis).
3. **Réentraîner** le modèle en enrichissant le dataset, en particulier sur ces zones “difficiles”.
4. Comparer les nouvelles versions dans MLflow (scores, courbes, temps d’entraînement).

5. **Déployer** une nouvelle version uniquement si elle apporte un gain réel (meilleure F1, stabilité, temps de réponse acceptable).

Cette boucle “observabilité → analyse → réentraînement → déploiement” est au cœur de la démarche MLOps.

Même dans un projet pédagogique, mettre en place ces briques permet de montrer que l’on sait dépasser le simple entraînement d’un modèle pour aller vers une solution exploitable.

Dans ce projet, nous avons :

- Construit et comparé trois familles de modèles (TF-IDF, embeddings + réseau de neurones, ModernBERT),
- Utilisé MLflow pour suivre et documenter les expériences,
- Sélectionné un modèle simple mais robuste pour le déploiement,
- Exposé ce modèle via FastAPI,
- Créé une interface Streamlit pour tester et collecter du feedback,
- Mis en place des tests unitaires et une CI pour sécuriser le code,
- Ajouté un mécanisme de monitoring et d’alerte, inspiré de ce qu’on ferait avec des outils comme Azure Application Insights.

L’ensemble constitue un prototype complet (conforme au cahier des charges du projet), qui montre qu’on peut aller de la donnée brute jusqu’à un service de scoring en ligne, tout en gardant la démarchede façon structurée, reproductible et orientée MLOps.