# Comparaison des approches de modélisation

Dans cet article, nous explorerons trois approches différentes de modélisation : le modèle sur-mesure simple basé sur TF-IDF et la régression logistique, le modèle sur-mesure avancé à l'aide d'un réseau de neurones avec embedding et convolution 1D, ainsi que le modèle avancé BERT. Chaque méthode est adaptée à différents scénarios et exigences. Nous verrons comment les intégrer dans une démarche MLOps, depuis le suivi des performances jusqu'à l'amélioration continue des modèles en production via des outils comme Azure Application Insights.

## Modèle simple : TF-IDF avec régression logistique

Le modèle sur mesure simple que nous avons utilisé repose sur TF-IDF (Term Frequency-Inverse Document Frequency) pour représenter les textes sous forme de vecteurs et une régression logistique pour effectuer la classification. Le TF-IDF est une méthode classique de NLP qui pondère l'importance des mots dans un corpus, tandis que la régression logistique effectue une classification linéaire.

Avantages :

* Simplicité : mise en œuvre rapide et facile avec des bibliothèques standards.
* Interprétabilité : il est simple d'identifier quels mots influencent le plus la décision du modèle.
* Efficace pour des jeux de données textuels modérés ou des tâches basiques de classification.

Limites :

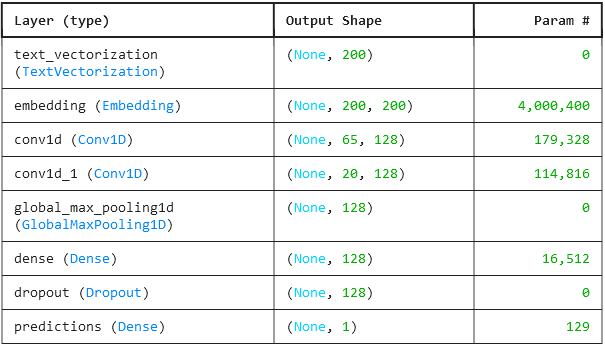
* Performance limitée : sur des corpus de grande taille ou des tâches complexes, la capacité du modèle diminue rapidement.
* Modèle linéaire : les relations complexes entre les mots ne sont pas capturées.

## Modèle intermédiaire : Réseau de neurones avec embedding et convolution 1D

Le modèle sur-mesure avancé repose sur un réseau de neurones comprenant plusieurs couches, dont une couche de vectorisation de texte suivie d’une couche d'embedding pour encoder les mots en vecteurs de 200 dimensions. Ensuite, deux couches de convolution 1D (Conv1D) permettent d’extraire les motifs locaux dans le texte, suivies par une couche de Global Max Pooling pour réduire la dimensionnalité. La dernière couche dense avec dropout aide à mieux généraliser les prédictions, avant d'aboutir à une couche de sortie pour la prédiction finale.

La structure du modèle est composée de :

* TextVectorization : conversion du texte en vecteurs numériques.
* Embedding (100 dimensions) : projection des mots dans un espace de 100 dimensions.
* Conv1D (2 couches) : détection des motifs de relations entre les mots.
* GlobalMaxPooling1D : réduction des dimensions après la convolution.
* Dense (128 neurones) + Dropout : affinage des caractéristiques et réduction du surapprentissage.
* Sortie finale (Dense) : génèration d’une prédiction de sortie binaire.



Avantages :

* Performance accrue pour des tâches textuelles complexes.
* Capture des relations locales à travers les couches de convolution.
* Modèle adaptable : possibilité de personnaliser la profondeur et la complexité du réseau.

Limites :

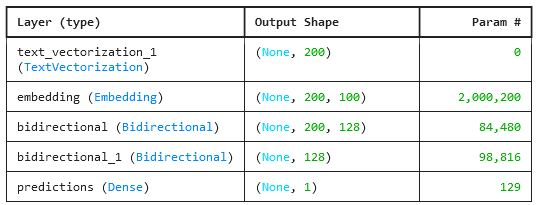
* Consommation élevée de ressources pour l'entraînement, nécessitant plus de calculs que des modèles simples.

## Modèle avancé LSTM

Ce modèle fait référence à un modèle Long Short Term Memory. Il permet la détection fine du contexte et du sens des phrases en balayant les tokens dans deux directions. Ce modèle est inspiré du cerveau humain qui peut oublier une partie des informations non nécessaire à l’apprentissage au cours du traitement ce qui permet de ne pas sur-apprendre et résoudre le problème du gradient évanescent.

La structure du modèle est composée de :

* **TextVectorization** : conversion du texte en séquences d'entiers.
* **Embedding (100 dimensions)** : projection des tokens dans un espace de 100 dimensions.
* **Bidirectional LSTM (128 unités, 1ère couche)** : capture les dépendances contextuelles de bas niveau dans les deux directions.
* **Bidirectional LSTM (128 unités, 2ème couche)** : affine la compréhension des relations plus abstraites
* **Dense**  : génère une prédiction binaire finale.



Avantages :

* **Gestion des dépendances à long terme** : Les LSTM capturent mieux les relations dans les séquences, contrairement aux CNN, qui se concentrent sur les motifs locaux.
* **Mémoire contextuelle** : Les LSTM utilisent des portes pour sélectionner quelles informations retenir ou oublier, offrant une meilleure gestion des séquences complexes.

Limites :

* Les LSTM sont plus lents que les CNN en raison de leur traitement séquentiel.

## Modèle complexe BERT

Le modèle BERT (Bidirectional Encoder Representations from Transformers) est une approche de deep learning qui repose sur des transformers bidirectionnels pour capter le contexte complet des mots dans un texte. C’est un modèle pré-entraîné qui excelle dans les tâches de traitement du langage naturel (NLP).

Avantages :

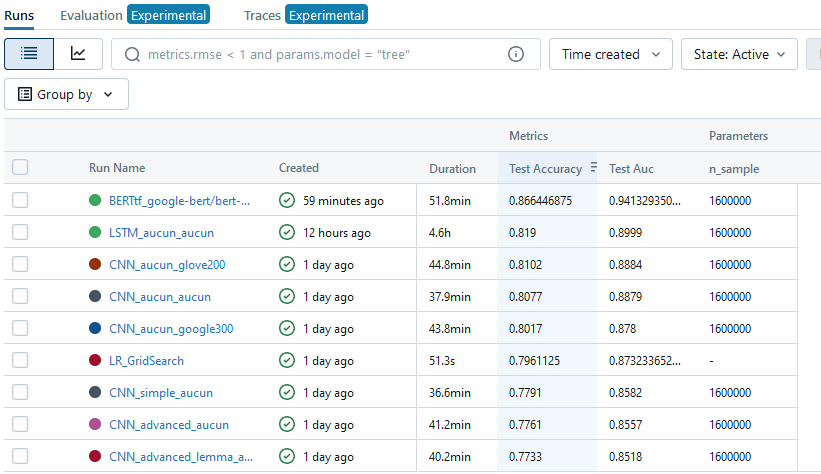
* Compréhension contextuelle profonde des relations sémantiques.
* Performance inégalée pour des tâches de NLP comme la classification, l'analyse de sentiment ou l'extraction d'entités.

Limites :

* Nécessite des ressources matérielles importantes, notamment pour l'entraînement.
* Complexité d'interprétation : BERT est souvent qualifié de "boîte noire" en raison de la difficulté à expliquer son fonctionnement interne.

## Bilan des résultats obtenus

Les résultats sont présentés dans la capture d’écran ci-dessous :



Le modèle simple logistic regression a un accuracy score de 0,796.

Le modèle CNN a un accuracy score de 0,810 . Le CNN avec les meilleurs résultats d’accuracy est obtenu avec l’utilisation d’un plongement de mot de type glove200.

On observe également que les prétraitements utilisés dégradent systématiquement les résultats :

* simple (0,779)
* avancé et lemmatisation (0,773).

Le modèle LSTM a un accuracy score de 0,819.

Le modèle BERT a un accuracy score de 0,866.

Le meilleur résultat en accuracy et en auc revient au modèle BERT. Toutefois, l’utilisation de ce modèle a été complexe à mettre en œuvre. En effet, il a été nécessaire de mettre en place une configuration spécifique permettant les calculs sur GPU alors que les autres modèles fonctionnent bien en CPU.

Le temps de traitement du modèle de BERT varie fortement selon la configuration de calcul choisie :

* Sur CPU, 8e4 tweets traités en 8 h,
* Sur GPU 1.6e6 tweets traités en 51 minutes.

Deux propositions sont viables suite à l’analyse des résultats :

* Afin de choisir le meilleur compromis, complexité – précision, nous avons décidé de déployer le modèle LSTM.
* Si le client souhaite un modèle simple et facile d’utilisation, le tf-idf avec logistic regression, présente également de bons résultats pour des temps de calculs très faibles (1 minute).

# Démarche orientée MLOps

## Gestion des versions et déploiement

L’approche MLOps (Machine Learning Operations) est cruciale pour assurer l’automatisation, la gestion et la maintenance des modèles machine-learning tout au long de leur cycle de vie ; de la formation initiale à la mise à jour en production.

La démarche repose sur les points clés suivants :

* Gestion des versions pour maintenir un historique clair de chaque modèle déployé.
* Intégration continue (CI) et déploiement continu (CD) pour la gestion des versions des modèles.
* Suivi des performances avec des outils de monitoring.

Nous avons utilisé MLflow sur un serveur local pour gérer le stockage des modèles et les différentes métriques. La performance des modèles est évaluée par les métriques « accuracy » et « auc score ».

Une fois les essais terminés un modèle est sélectionné et « push » sur github pour déploiement.

Le déploiement du code est réalisé par github actions afin de permettre l’intégration en continu du projet.

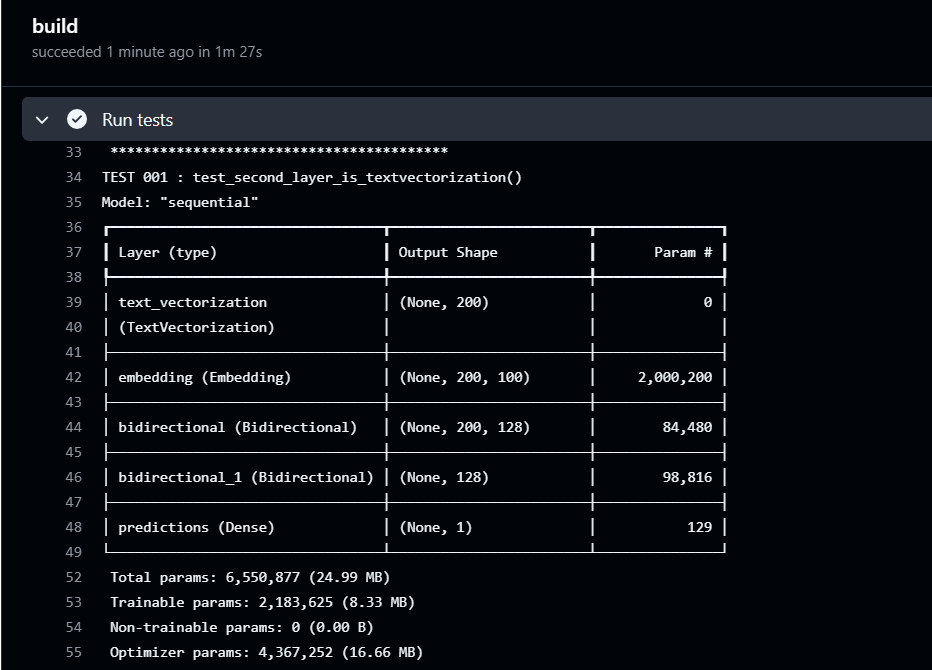
Le workflow GitHub Actions est conçu pour automatiser le processus de construction et de déploiement d'une application Python sur Azure Web App. Ce workflow est configuré pour se déclencher automatiquement lors des pushs réalisés sur la branche « main » du dossier.

Dans un premier temps, le script récupère le code source du dépôt à l'aide de l'action checkout. Il configure ensuite la version de Python spécifiée, crée un environnement virtuel pour l'application sur ubuntu et installe les dépendances nécessaires. Des tests sont exécutés pour assurer la qualité du code.

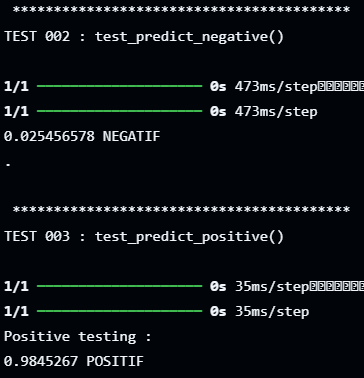
Nous avons utilisé pytest pour vérifier différents aspects du modèle. Voici les principaux tests implémentés :

* Vérification de la couche TextVectorization : Ce test permet de s'assurer que la première couche du modèle est bien une couche TextVectorization. Cela garantit que le modèle effectue correctement le prétraitement du texte avant de passer aux étapes suivantes.
* Test de prédiction pour une entrée négative : Ce test envoie un texte négatif ("terrible") à l'API et vérifie que la prédiction du modèle est bien inférieure à 0,1 ; ce qui indique une prédiction correcte pour une entrée négative.
* Test de prédiction pour une entrée positive : Ce test envoie un texte positif ("awesome") et vérifie que la prédiction est supérieure à 0,9, ce qui correspond à une prédiction correcte pour une entrée positive.

Ces tests permettent de valider à la fois la structure du modèle (en vérifiant la présence de la couche TextVectorization) et son comportement (en testant les prédictions pour des textes positifs et négatifs). Cela assure que chaque nouvelle version du modèle fonctionne, comme prévu, avant d'être déployée en production.



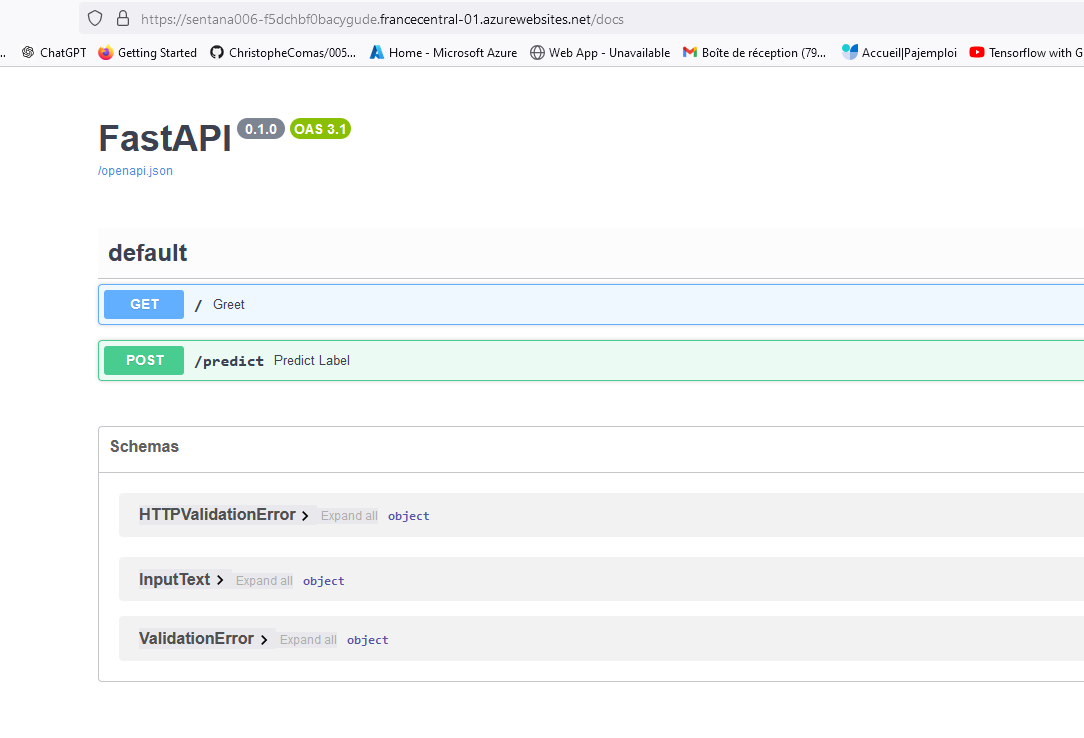
Visualisation du test 1 sur github actions



Visualisation des tests 2 et 3 sur github actions

Les fichiers de l'application sont ensuite compressés en un fichier ZIP et cet artefact est transmis à Azure pour être utilisé dans le processus de déploiement.

Le job de déploiement commence par le téléchargement de l'artefact ZIP. Cet artefact est ensuite décompressé et déployé sur Azure Web App en utilisant le profil de publication sécurisé fourni.



Visualisation de l’API

## Suivi des performances en production

Une fois les modèles déployés, il est essentiel de les surveiller en continu pour détecter les dérives ou les baisses de performance.

Avec Azure Application Insights, il est possible de :

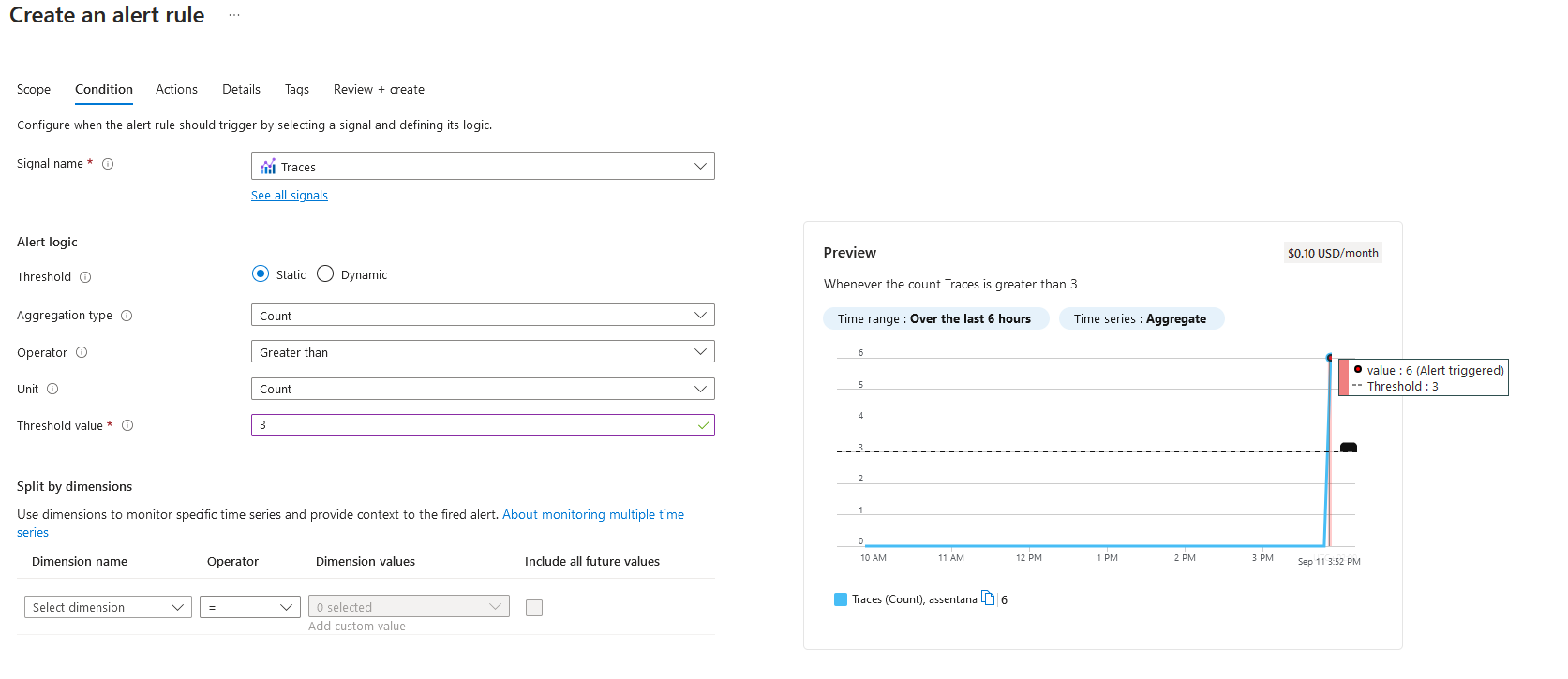
* Suivre les erreurs et anomalies dans les prédictions du modèle.
* Configurer des alertes pour signaler des dérives ou des chutes de performance.

Pour configurer une alerte il est nécessaire de :

* Choisir la data à remonter grâce à une query sur les différents logs disponibles,

*Nota :* Dans notre cas, nous remontons une trace avec opentelemetry permettant de visualiser la phrase de l’utilisateur dont la prédiction n’est pas bonne.

* Choisir le temps d’analyse,
* Choisir le nombre d’occurrence requis,
* Définir un mode de communication de l’alerte.



Visualisation des collections de traces et du seuil d’alerte

Il pourrait être intéressant d’utiliser ces données collectées en production afin de réajuster le modèle au fil du temps. Cela pourrait inclure le réentraînement périodique du modèle avec de nouvelles données pour maintenir sa performance.

Les alertes ne font plus parties des services gratuits d’Azure Insight.