

End-to-End Latency-Optimized Architecture for Semantic Veracity Detection

Brahim Chahba*

brahim.chahba@etu.uae.ac.ma

Taha Lmhandi*

taha.lamhandi@etu.uae.ac.ma,

Abstract

Ce rapport détaille l'ingénierie et le déploiement d'un système de classification binaire conçu pour détecter la désinformation dans les discours politiques courts (dataset LIAR). L'objectif principal était d'établir un pipeline d'inférence à haut débit et à faible latence capable d'analyse en temps réel. Nous avons utilisé **DistilBERT**, une version distillée de BERT, pour équilibrer la compréhension sémantique et l'efficacité computationnelle. Le modèle a été affiné (fine-tuned) sur $\approx 10,000$ échantillons, atteignant un score F1 de validation stable de **0.63** avant l'apparition de dynamiques de surapprentissage. Le système de production est conteneurisé sur **Azure ML**, servant les prédictions via une instance **Standard_D2as_v4** avec une latence moyenne de **100 – 160ms**. L'application côté client, construite avec **React**, **Vite** et **Tailwind**, consomme l'API d'inférence pour présenter à la fois les étiquettes de classification et les scores de confiance probabilistes, fournissant aux utilisateurs finaux des métriques de véracité interprétables.

1 Introduction

Les fausses nouvelles (fake news) et la désinformation sont des problèmes critiques pour les écosystèmes d'information modernes. Le but de ce projet est de construire un classificateur automatisé qui distingue les courtes déclarations politiques probablement fausses (“Fake”) de celles probablement véridiques (“Real”) en utilisant des modèles NLP modernes basés sur des transformateurs. L'objectif de conception est pragmatique : produire un pipeline reproductible et prêt pour la production qui équilibre qualité, latence, coût et simplicité opérationnelle.

Ce rapport documente l'ensemble du cycle de vie de l'ingénierie : préparation du jeu de données (LIAR), entraînement de DistilBERT à l'aide d'AzureML, évaluation et analyse des erreurs, et déploiement en tant qu'endpoint géré Azure avec une interface frontale React + Vite.

[Accès à la Démonstration](#)

2 Travaux connexes

Les modèles Transformers (Vaswani et al., 2017) ont considérablement fait progresser le NLP et sont efficaces pour les tâches de classification de texte. DistilBERT (Sanh et al., 2019) offre une alternative compacte et plus rapide à BERT tout en préservant une grande partie de ses performances. Le dataset LIAR (W. Y. Wang, ACL 2017) est un benchmark couramment utilisé pour la détection de véracité des déclarations courtes et est utilisé ici comme jeu de données principal. Le rapport fait référence à ces travaux fondateurs là où cela est approprié.

3 Données

3.1 Source

Nous utilisons le dataset LIAR (W. Yang Wang, ACL 2017) obtenu via Kaggle. Le jeu de données contient de courtes déclarations politiques, des métadonnées riches et des étiquettes à granularité fine (pants-fire, false, barely-true, half-true, mostly-true, true).

3.2 Nettoyage et mappage

Pour rendre la tâche traitable et robuste, nous avons converti les étiquettes originales à 6 niveaux en un mappage binaire :

- **Fake (0)**: pants-fire, false, barely-true
- **Real (1)**: half-true, mostly-true, true

Étapes de nettoyage :

1. Lecture du TSV avec pandas (ignorer les lignes mal formées).
2. Conservation uniquement des champs `statement` et `label`.
3. Suppression des valeurs manquantes et des doublons.
4. Suppression des espaces et rejet des déclarations de moins de 10 caractères pour éliminer le bruit.
5. Mappage des étiquettes en entiers et conversion (cast) en `int`.

3.3 Tailles des jeux de données

- Total du jeu de données brut : 13,227
- Split de Test : 1,282
- Split de Validation : 1,276

3.4 Équilibre des classes

Après le mappage binaire et le nettoyage, l'ensemble de validation montre un support quasi équilibré : Fake=615, Real=665 (total validation 1280). L'équilibre des classes

est acceptable mais pas parfait ; par conséquent, le F1 et les métriques par classe sont importants.

4 Modèle et méthodologie d’entraînement

4.1 Architecture du modèle

- **Encodeur de base** : DistilBERT (6 couches, 12 têtes, dim cachée 768), poids pré-entraînés `distilbert-base-uncased`.
- **Tête (Head)** : tête de classification de séquence avec `num_labels=2`.
- **Tokenizer** : DistilBertTokenizerFast (taille vocabulaire 30,522).

Un extrait de la configuration du modèle (artefact enregistré) :

```
"architectures": ["DistilBertForSequenceClassification"],  
"dim": 768,  
"n_layers": 6,  
"n_heads": 12,  
"seq_classif_dropout": 0.2,  
"transformers_version": "4.41.2"
```

4.2 Recette d’entraînement

- **Environnement** : AzureML (Standard_D4s_v3 : 4 coeurs, 16 Go de RAM).
- **Logiciel** : Python 3.10, PyTorch 2.2.2, Transformers 4.41.2.
- **Hyperparamètres** : `epochs = 5`, `batch_size = 16`, `learning_rate = 2e-5`, `max_len = 128`.
- **Optimiseur** : AdamW (HuggingFace/transformers), planificateur de taux d’apprentissage linéaire avec chauffe (warmup).
- **Journalisation** : MLflow utilisé pour le suivi des paramètres/métriques/artefacts ; meilleurs modèles par F1 de validation sauvegardés dans `outputs/best_model` et enregistrés comme artefacts.

4.3 Durée d’entraînement et calcul

Un seul travail d’entraînement AzureML a été exécuté pendant **5.5 heures** sur Standard_D4s_v3. Cela correspond à un fine-tuning de DistilBERT de taille modeste sur 7–8k échantillons d’entraînement.

5 Analyse des performances et résultats

Les journaux d’entraînement indiquent qu’un checkpoint optimal a été atteint à l’**Époque 2** avant qu’un surapprentissage significatif ne dégrade les performances de généralisation.

Table 1: Résumé de la dynamique d’entraînement (5 Époques)

Époque	Perte Train	Perte Val	Précision Val	F1 (Pondéré)	Statut
1	0.6656	0.6660	60.12%	0.5740	Sous-apprentissage
2	0.6094	0.6422	63.79%	0.6309	Optimal
3	0.4867	0.7518	63.40%	0.6297	Début surapprentissage
5	0.2378	1.0089	62.31%	0.6207	Divergence

5.1 Dynamique d’entraînement

Le meilleur F1 de validation a eu lieu à l’époque 2 et a été enregistré comme le meilleur artefact de modèle : `best_model_binary_epoch_1_ts...`.

5.2 Observations

- Le grand écart entre la perte d’entraînement (0.2378) et la perte de validation (1.0089) à l’époque 5 indique un **surapprentissage** — le modèle s’adapte plus étroitement aux données d’entraînement tandis que la généralisation se détériore après le point d’arrêt précoce (meilleur F1 à l’époque 2).
- Le meilleur modèle pratique était à l’époque 2 (val F1 = 0.6309, val acc ≈ 0.638). La production devrait utiliser ce checkpoint ou un checkpoint ajusté avec un arrêt précoce (early stopping) / décroissance de poids (weight decay).

5.3 Métriques d’évaluation (Époque optimale)

Les métriques du modèle déployé (checkpoint Époque 2) sur l’ensemble de validation ($N = 1284$) sont :

$$\text{Précision} = \frac{TP + TN}{TP + TN + FP + FN} \approx 0.6379 \quad (1)$$

Pour la classe **Real (1)**, les métriques clés sont :

$$\text{Précision}_{\text{real}} = \frac{TP}{TP + FP} \approx 0.62 \quad (2)$$

$$\text{Rappel}_{\text{real}} = \frac{TP}{TP + FN} \approx 0.76 \quad (3)$$

Le rappel plus élevé pour la classe ‘Real’ suggère une légère réticence du modèle à classer les déclarations ambiguës comme ‘Fake’, favorisant la classe de support plus large.

6 Analyse des erreurs et améliorations pratiques

6.1 Causes probables du plafond de performance

- **Étiquettes bruitées et textes courts** : Les déclarations LIAR sont courtes et parfois ambiguës — le bruit des étiquettes est intrinsèque.

- **Données limitées pour le fine-tuning** : après nettoyage, la taille d'entraînement $\approx 7.6k$ est modeste pour des transformateurs.
- **Surapprentissage dû à un fine-tuning agressif** : la perte d'entraînement a diminué rapidement mais la perte de validation a augmenté après l'époque 2.

6.2 Améliorations pratiques à haute valeur ajoutée

1. **Validation croisée stratifiée ou exécutions répétées avec différentes graines (seeds)** : réduit la variance et donne des estimations stables.
2. **Pondération des classes / Focal Loss** : si les faux négatifs pour une classe sont plus coûteux, pondérer la perte en conséquence.
3. **Augmentation de données** : rétro-traduction, paraphrase ou transformations syntaxiques pour étendre le jeu de données (ROI élevé).
4. **Mise à l'échelle du modèle** : tester BERT-base ou RoBERTa dans une petite expérience ; compromis coût vs précision.
5. **Utilisation des métadonnées** : les orateurs, le sujet et le contexte peuvent être prédictifs lorsqu'ils sont fusionnés avec le texte — essayer des modèles multi-entrées ou la concaténation de caractéristiques.

7 Déploiement en production et architecture

7.1 Architecture de haut niveau

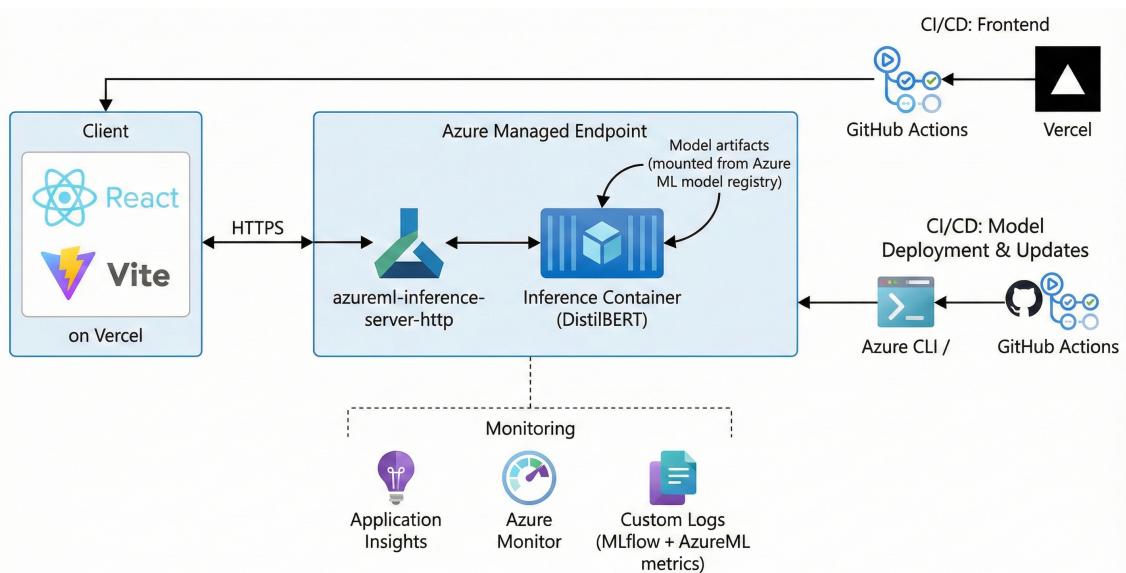


Figure 1: Composants de déploiement

8 Monitoring

Le monitoring opérationnel est essentiel pour garantir la stabilité du système déployé. Dans notre cas, plusieurs indicateurs sont suivis en continu via AzureML :

- **Latence P50/P95/P99**, afin de détecter rapidement tout ralentissement dû à la charge, aux mises à jour ou à une dérive de performance.
- **Taux d'erreurs HTTP**, permettant d'identifier les problèmes d'infrastructure ou d'intégration côté client.
- **Distribution des scores de probabilités**, utile pour surveiller une éventuelle dérive de données (data drift) ou une perte de calibration.

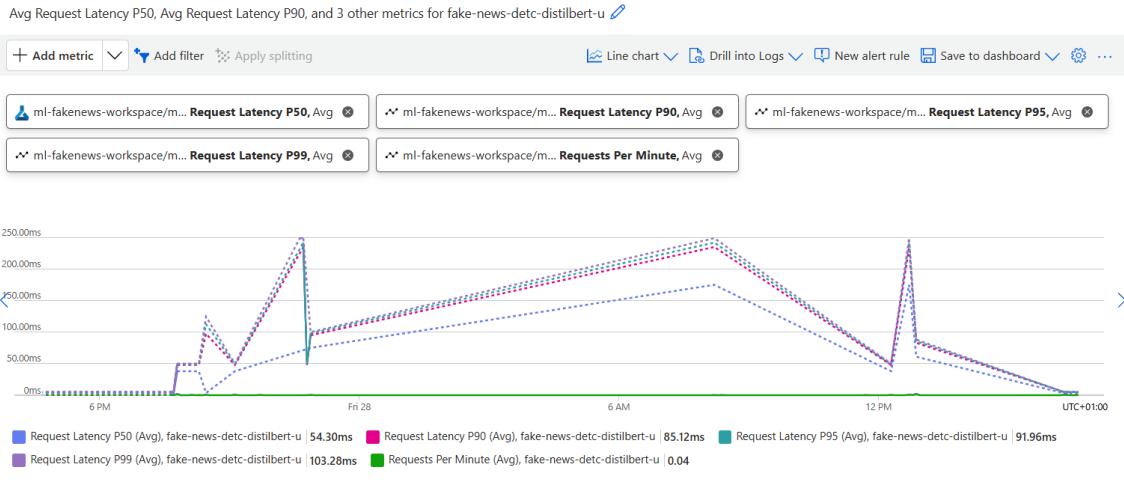


Figure 2: Monitoring de la latence des requêtes P50

9 Références

1. W. Y. Wang. “Liar, Liar Pants on Fire”: A New Benchmark Dataset for Fake News Detection. ACL, 2017.
2. V. Sanh, L. Debut, J. Chaumond, T. Wolf. “DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter.” arXiv:1910.01108, 2019.
3. Ashish Vaswani et al. “Attention Is All You Need.” NeurIPS 2017.
4. Documentation Hugging Face Transformers.
5. Documentation Azure Machine Learning ([déploiement de modèle, serveur d’inférence](#))