



عهد الوطني للبريد والمواصلات  
የኢትዮጵያ ፖስታና ቴሌኮሙኒኬሽን ቢሮ  
Institut National des Postes et Télécommunications

# LLM FOR FAKE NEWS DETECTION

PROJET DE FIN D'ANNÉE

Réalisé par:  
ABAALI Sanae  
BENSALIM Kaoutar

Encadré par:  
Mme. EL ASRI Ikram  
Mme. AYACHE Meryem



# Table des Matières

1. Introduction
  - a. Contexte du Projet
  - b. Objectif du Projet
2. Revue de la Littérature
  - a. Définition et Importance de la Détection des Fausses Nouvelles
  - b. Techniques Existantes pour la Détection des Fausses Nouvelles
  - c. Présentation du Modèle DeBERTa
3. Données Utilisées
  - a. Description du Jeu de Données
  - b. Prétraitement des Données
4. Méthodologie
  - a. Téléchargement du Modèle
  - b. Processus d'Entraînement du Modèle
  - c. Évaluation du Modèle
5. Résultats
  - a. UI
  - b. Test et Validation de l'Application
6. Conclusion
7. Références

# 1. Introduction

## a. Contexte du Projet

La prolifération des fausses nouvelles est devenue un problème majeur dans notre société moderne, où l'information est accessible à tous instantanément via internet et les réseaux sociaux. Ces fausses nouvelles peuvent avoir des conséquences graves, notamment en influençant l'opinion publique, en affectant les résultats des élections, et en créant des tensions sociales. La détection efficace des fausses nouvelles est donc cruciale pour maintenir l'intégrité de l'information et protéger la société contre la désinformation.

## b. Objectif du Projet

L'objectif de ce projet est de développer un modèle basé sur l'apprentissage automatique capable de détecter les fausses nouvelles avec une grande précision. Pour ce faire, nous utilisons le modèle DeBERTa (Decoding-enhanced BERT with disentangled attention) de Microsoft, qui est reconnu pour ses performances exceptionnelles en traitement du langage naturel. Nous visons à entraîner et à évaluer ce modèle sur un jeu de données spécifique aux fausses nouvelles, puis à créer une Interface utilisateur permettant de tester le modèle.

# 2. Revue de la Littérature

## a. Définition et Importance de la Détection des Fausses Nouvelles

Les fausses nouvelles, également connues sous le nom de "fake news", sont des informations délibérément fausses ou trompeuses diffusées via divers canaux de communication, principalement en ligne. La détection des fausses nouvelles est essentielle pour plusieurs raisons. Elle permet de maintenir l'intégrité de l'information, de protéger les individus et les institutions contre les conséquences négatives de la désinformation, et de promouvoir une société bien informée et résiliente. La propagation rapide des fausses nouvelles peut avoir des impacts dévastateurs sur la société, influençant les élections, alimentant la polarisation et même déclenchant des violences.

## b. Techniques Existantes pour la Détection des Fausses Nouvelles

La détection des fausses nouvelles a évolué avec l'avènement des technologies de l'information et de l'intelligence artificielle. Voici un aperçu des techniques couramment utilisées :

**Vérification Manuelle des Faits :** Cette méthode repose sur l'expertise humaine où des journalistes ou des experts vérifient les faits manuellement en comparant les informations suspectes avec des sources fiables. Bien que cette méthode offre une précision et une crédibilité élevées, elle est souvent longue, coûteuse et difficile à généraliser à grande échelle.

**Analyse de Contenu :** L'analyse de contenu implique l'examen linguistique et stylistique du texte pour détecter des indices de tromperie ou de partialité. Les techniques utilisées incluent l'analyse de la syntaxe, la détection de mots-clés suspects, ainsi que l'analyse de la polarité et du ton. Cette approche peut fournir des indications sur l'intention et la qualité du contenu, mais elle peut manquer de précision sans validation humaine.

**Apprentissage Automatique (Machine Learning) :** L'apprentissage automatique utilise des algorithmes pour entraîner des modèles sur des ensembles de données étiquetés afin de prédire la véracité des nouvelles informations. Parmi les techniques courantes, on trouve la régression logistique, les arbres de décision, les forêts aléatoires et les support vector machines (SVM). Cette méthode est capable de traiter de grandes quantités de données et d'apprendre des modèles complexes, bien qu'elle nécessite des données étiquetées de haute qualité pour l'entraînement.

**Traitement Automatique du Langage Naturel (NLP) :** Les techniques avancées de NLP sont utilisées pour analyser et comprendre le contenu textuel à un niveau plus profond. Les modèles de réseaux de neurones récurrents (RNN), les réseaux de neurones convolutifs (CNN) et les transformers (comme BERT, GPT et DeBERTa) sont couramment employés. Cette approche est capable de capturer les nuances contextuelles et syntaxiques, offrant une haute précision dans les tâches de classification de texte, bien que sa mise en œuvre soit complexe et demandeuse en ressources de calcul.

**Détection Basée sur le Réseau Social :** Cette méthode analyse les interactions sur les réseaux sociaux pour identifier les sources et les patterns de diffusion des fausses nouvelles. Les techniques incluent les graphes de diffusion, l'analyse de sentiment des utilisateurs et la détection des bots. Cette approche est efficace pour détecter les fausses nouvelles dès leur propagation, mais elle dépend fortement des données des réseaux sociaux et soulève des enjeux de confidentialité.

En conclusion, ces techniques montrent une progression claire vers des méthodes plus sophistiquées et automatisées, ouvrant la voie à l'utilisation de modèles de langage de grande taille comme DeBERTa, que nous mettons en œuvre dans ce projet pour une détection plus efficace et précise des fausses nouvelles.

## c. Présentation du Modèle DeBERTa

### **Introduction aux Modèles de Langage de Grande Taille (LLM):**

Un grand modèle de langage (LLM) est un type d'intelligence artificielle (IA) capable de générer un texte de type humain et d'effectuer diverses tâches de traitement du langage naturel . Les LLM sont formés sur d'énormes quantités de données textuelles, ce qui leur permet d'apprendre les modèles et les relations qui existent dans le langage. Cela les rend capables de générer un texte souvent impossible à distinguer de celui écrit par des humains.

Voici quelques exemples concrets de LLM en action :

- **Google Translate :** Google Translate utilise un LLM pour traduire du texte d'une langue à une autre. Le LLM est formé sur un ensemble massif de données de texte et de code, ce qui lui permet d'apprendre les modèles et les relations qui existent entre différentes langues. Cela fait de Google Translate l'un des outils de traduction les plus précis disponibles.
- **GPT-3 :** GPT-3 est un grand modèle de langage développé par OpenAI. GPT-3 est capable de générer du texte de type humain, de traduire des langues et d'écrire différents types de contenu créatif. Il a été utilisé pour créer diverses applications, notamment des chatbots, des générateurs de texte et des outils d'écriture créative.
- **Bard :** Bard est un grand modèle de langage développé par Google AI. Bard est capable de répondre à vos questions de manière informative, même si elles sont ouvertes, difficiles ou étranges. Il peut également générer différents formats de texte créatifs, comme des poèmes, du code, des scripts, des pièces musicales, des e-mails, des lettres, etc.

## Les Transformateurs:

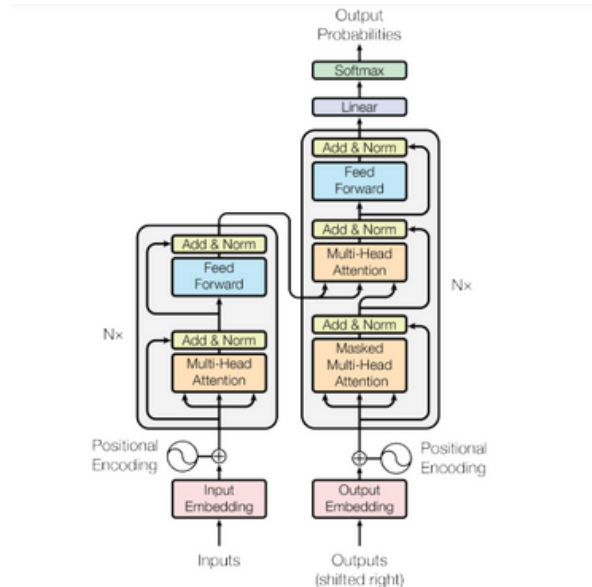


fig1: The Transformer - model architecture.

Un transformateur est une architecture de réseau neuronal qui exploite un mécanisme d'auto-attention pour capturer les relations à longue distance entre les éléments des séquences d'entrée et de sortie.

Les composants et étapes clés:

**Embeddings et Encodages Positionnels:** Permet la transformation des mots en vecteurs de grande dimension et l'intégration des informations de position dans la séquence.

**Mécanisme d'Auto-Attention :** active la pondération des mots par rapport aux autres dans la séquence et capte des dépendances contextuelles.

**Attention Multi-Tête :** Capture des différentes nuances contextuelles.

**Couches Feedforward et Normalisation:** application des transformations non linéaires à chaque position de la séquence indépendamment des autres positions , réduction des problèmes de dégradation des gradients et amélioration de la convergence du modèle.

## Les LLMs Opensource:

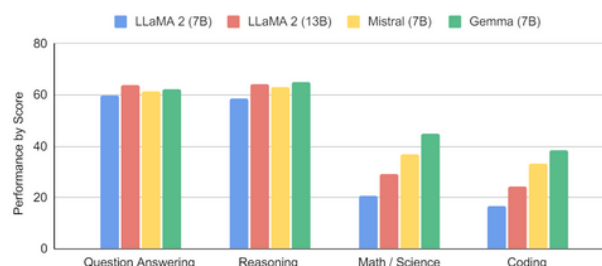


fig2: Comparaison entre les Grandes LLMs.

**Gemma** est une famille de modèles ouverts ultramodernes et légers, construits à partir des mêmes recherches et technologies utilisées pour créer les modèles Gemini. Développée par Google DeepMind et d'autres équipes chez Google, Gemma s'inspire de Gemini, et le nom reflète le mot latin "gemma", qui signifie "pierre précieuse".

**LLaMA-2** est une variante de grands modèles de langage qui offre des performances de pointe en compréhension et génération de langage. Son architecture permet une personnalisation étendue, en en faisant un candidat idéal pour le finetuning avec des ensembles de données et des objectifs spécifiques.

**Mistral LLM** est un modèle de langage de grande taille développé par Mistral AI, une start-up française qui fait sensation dans la communauté technologique. Mistral LLM est un modèle de langage basé sur un décodeur avec 7 milliards de paramètres, ce qui en fait l'un des modèles de langage les plus importants disponibles.

Ces modèles de langage de grande taille tels que Gemma, Mistral et LLaMa-2 sont reconnus pour leurs performances exceptionnelles dans diverses tâches de traitement du langage naturel. Cependant, leur utilisation nécessite des ressources informatiques considérables en raison de leurs architectures complexes et de leur grand nombre de paramètres. Cette exigence en matière de capacité de calcul a incité à explorer des alternatives plus légères, notamment les MINI modèles de langage avec moins de paramètres d'entraînement. Ces derniers, bien que moins impressionnants en termes de taille et de performance, offrent une solution plus viable pour de nombreux cas d'utilisation, en particulier dans les environnements où les ressources sont limitées. Ainsi, la recherche dans le domaine des modèles de langage s'efforce de trouver un équilibre entre la puissance et l'efficacité, afin de répondre aux besoins variés des utilisateurs.

### **DeBERTa:**

DeBERTa, qui signifie "Decoding-enhanced BERT with disentangled attention", est un modèle de traitement du langage naturel amélioré apparu en 2021 grâce au « Disentangled attention », le modèle intégrant une attention démêlée présentait des améliorations importantes dans les meilleurs benchmarks NLP, par rapport à d'autres grands modèles.

Contrairement aux modèles Transformer traditionnels, où chaque jeton est représenté par un seul vecteur combinant informations de contenu et de position, DeBERTa sépare ces informations en deux vecteurs distincts. De plus, l'algorithme d'attention est modifié pour prendre explicitement en compte les relations entre le



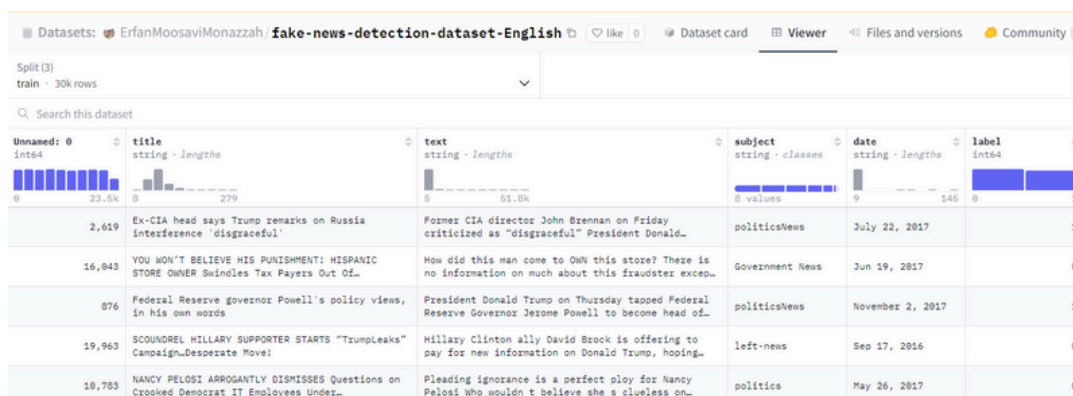
contenu et les positions des jetons, permettant ainsi une compréhension plus fine et précise du langage.

Pour ce projet, nous avons utilisé la version "microsoft/deberta-v3-base" qui comporte un total de 184 423 682 paramètres, tous étant entraînaibles.

## 3. Données Utilisées

### a. Description du Jeu de Données

Le jeu de données utilisé dans ce projet provient de la collection "fake-news-detection-dataset-English" disponible sur Hugging Face. Il est divisé en trois sous-ensembles distincts : un ensemble d'entraînement composé de 30 000 échantillons, un ensemble de validation comprenant 6 000 échantillons et un ensemble de test de 8 270 échantillons. Chaque entrée du jeu de données inclut plusieurs attributs : "Unnamed: 0" (un index), "title" (le titre de l'article), "text" (le corps de l'article), "subject" (le sujet de l'article), "date" (la date de publication) et "label" (l'étiquette indiquant si l'article est une fausse nouvelle ou une vraie nouvelle). Les étiquettes sont codées de la manière suivante : 0 pour les fausses nouvelles et 1 pour les vraies nouvelles. Ce jeu de données diversifié et bien structuré est essentiel pour entraîner, valider et tester notre modèle de détection de fausses nouvelles, garantissant ainsi une évaluation rigoureuse de ses performances.



Unnamed: 0	title	text	subject	date	label
2,619	Ex-CIA head says Trump remarks on Russia interference 'disgraceful'	Former CIA director John Brennan on Friday criticized as "disgraceful" President Donald...	politicsNews	July 22, 2017	1
16,843	YOU WON'T BELIEVE HIS PUNISHMENT! HISPANIC STORE OWNER Swindles Tax Payers Out Of...	How did this man come to OWN this store? There is no information on much about this fraudster excep...	Government News	Jun 19, 2017	0
876	Federal Reserve governor Powell's policy views, in his own words	President Donald Trump on Thursday tapped Federal Reserve Governor Jerome Powell to become head of...	politicsNews	November 2, 2017	1
19,963	SCOUNDREL HILLARY SUPPORTER STARTS "TrumpLeaks" Campaign...Desperate Move!	Hillary Clinton ally David Brock is offering to pay for new information on Donald Trump, hoping...	left-news	Sep 17, 2016	0
10,783	NANCY PELOSI ARROGANTLY DISMISSES Questions on Crooked Democrat IT Employees Under...	Pleading ignorance is a perfect play for Nancy Pelosi who wouldn't believe she's clueless on...	politics	May 26, 2017	0

fig3:Apperçu du jeu de données.

### b. Prétraitement des Données

Le prétraitement des données est une étape cruciale pour préparer le jeu de données avant de l'utiliser pour entraîner un modèle de machine learning. Voici une description détaillée du code de prétraitement utilisé dans ce projet :

## 1. Création d'un Prompt Unique :


La fonction `process_data` est définie pour combiner le titre, le sujet et le texte de chaque article de presse afin de créer un prompt unique. Ce prompt intègre toutes les informations pertinentes de l'article en une seule chaîne de texte. De plus, une étiquette indiquant si l'article est une fausse nouvelle ou une vraie nouvelle est ajoutée à la fin du prompt. Cette étiquette est déterminée par la valeur de la colonne `label`.

```
def process_data(example):  
    # Combinons le titre, le sujet et le texte pour créer un prompt unique.  
    example['prompt'] = "the news title is: " + example['title'] + "\n\n" + "The subject of the news is: " + example['subject'] + "\n\n" + "The  
    # Ajoutons une étiquette indiquant si la nouvelle est vraie ou fausse.  
    example['complete_prompt'] = example['prompt'] + "this news is " + ("fake" if example['label'] == 0 else "real")  
    return example
```

fig4:code 1.

La fonction `process_data` est ensuite appliquée à l'ensemble du jeu de données à l'aide de la méthode `map`. Cela transforme chaque entrée du dataset en ajoutant les prompts et les étiquettes complètes.

```
# Appliquons la fonction de traitement  
data = data.map(process_data)
```



Map: 100% 30000/30000 [00:04<00:00, 6815.87 examples/s]  
Map: 100% 6000/6000 [00:01<00:00, 6466.57 examples/s]  
Map: 100% 8267/8267 [00:01<00:00, 6537.80 examples/s]

fig5:code 2.

## 2. Configuration du Tokenizer :

Pour garantir que le tokenizer puisse gérer les entrées correctement, le jeton de padding (`pad_token`) est défini comme étant le même que le jeton de fin de séquence (`eos_token`). De plus, un jeton spécial [PAD] est ajouté au tokenizer pour marquer les positions de padding.

La fonction `tokenize_dataset` est définie pour tokeniser chaque exemple du dataset. Elle utilise le tokenizer pour transformer le texte en une séquence de tokens avec un padding jusqu'à une longueur maximale de 512 tokens. Cette fonction est appliquée de manière groupée (batched) à l'ensemble du dataset, ce qui permet de traiter les données plus efficacement.

```
# Définition du jeton de padding comme le jeton de fin de séquence (EOS) du tokenizer.  
tokenizer.pad_token = tokenizer.eos_token  
# Ajout du jeton spécial [PAD] au tokenizer.  
tokenizer.add_special_tokens({'pad_token': '[PAD]'})  
  
# Fonction pour tokeniser chaque exemple du dataset en ajoutant un padding pour atteindre une longueur maximale de 512 tokens.  
def tokenize_dataset(example):  
    response = tokenizer(example['complete_prompt'], padding='max_length', truncation=True, max_length=512)  
    return response  
  
# Application de la fonction de tokenisation à l'ensemble du dataset (batched)  
data = data.map(tokenize_dataset, batched=True)
```

fig6:code 3.

### 3. Mise en Forme des Données :

Après la tokenisation, les colonnes inutiles telles que title, subject, text et Unnamed: 0 sont supprimées du dataset. Les colonnes restantes, input\_ids, prompt, attention\_mask, label et complete\_prompt, sont mises en forme pour être utilisées avec PyTorch. Cela implique de convertir les données en un format compatible avec les tensors PyTorch, facilitant ainsi leur utilisation lors de l'entraînement du modèle.

```
# Suppression des colonnes inutiles du dataset et mise en forme des données restantes pour l'utilisation avec PyTorch.
data = data.remove_columns(['title', 'subject', 'text', 'Unnamed: 0'])
data.set_format('torch', columns=['input_ids', 'prompt', 'attention_mask', 'label', 'complete_prompt'])
```

*fig7:code 4.*

## 4. Méthodologie

### a. Téléchargement du Modèle

Dans cette section, nous téléchargeons et préparons le modèle DeBERTa v3 base pour la détection des fausses nouvelles. Nous utilisons la bibliothèque transformers de Hugging Face pour accéder à ce modèle pré-entraîné. Tout d'abord, nous chargeons le tokenizer associé en utilisant AutoTokenizer.from\_pretrained, ce qui permet de transformer le texte brut en séquences de tokens compréhensibles par le modèle. Ensuite, nous chargeons le modèle de classification de séquences avec deux étiquettes (num\_labels=2) grâce à AutoModelForSequenceClassification.from\_pretrained. Cette préparation initiale est essentielle pour adapter DeBERTa à notre tâche spécifique de classification des nouvelles en fausses ou vraies, en utilisant ses mécanismes avancés d'attention désentrelacée.

```
from transformers import AutoTokenizer, AutoModelForSequenceClassification
model_name = "microsoft/deberta-v3-base"
# Charger le tokenizer DeBERTa
tokenizer = AutoTokenizer.from_pretrained(model_name)
# Charger le modèle DeBERTa pour la classification de séquences
model = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=2)
```

*fig8:code 5.*

### b. Processus d'Entraînement du Modèle

Le processus d'entraînement du modèle implique plusieurs étapes cruciales, définies dans le code suivant.

```

# Arguments de l'entraînement
training_args = TrainingArguments(
    output_dir="./results", # Répertoire de sortie pour les résultats
    evaluation_strategy="epoch", # Stratégie d'évaluation après chaque époque
    learning_rate=2e-5, # Taux d'apprentissage
    per_device_train_batch_size=8, # Taille du batch d'entraînement par appareil
    per_device_eval_batch_size=8, # Taille du batch d'évaluation par appareil
    num_train_epochs=3, # Nombre d'époques d'entraînement
    weight_decay=0.01, # Décroissance de poids
    save_total_limit=1, # Limite du nombre total de sauvegardes
    save_steps=1000, # Sauvegarde tous les 1000 pas
    logging_dir='./logs', # Répertoire de journalisation
    report_to="none", #désactiver le W&B
)

# Initialiser le Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=data["train"],
    eval_dataset=data["validation"],
    tokenizer=tokenizer,
)

# Affiner l'ensemble du modèle
trainer.train()

```

fig9:code 6.

Tout d'abord, nous spécifions les arguments de l'entraînement via `TrainingArguments`. Ces arguments incluent le répertoire de sortie pour les résultats (`output_dir`), la stratégie d'évaluation après chaque époque (`evaluation_strategy`), le taux d'apprentissage (`learning_rate`), et les tailles de batch pour l'entraînement et l'évaluation (`per_device_train_batch_size` et `per_device_eval_batch_size`). Nous fixons également le nombre d'époques d'entraînement à trois (`num_train_epochs`), et définissons la décroissance de poids (`weight_decay`). Pour gérer les sauvegardes, nous limitons le nombre total à une (`save_total_limit`) et sauvegardons tous les 1000 pas (`save_steps`). Le répertoire de journalisation est défini par `logging_dir` et nous désactivons le rapport à W&B en définissant `report_to` sur "none".

Ensuite, nous initialisons le `Trainer` de Hugging Face avec le modèle `DeBERTa`, les arguments d'entraînement, le dataset d'entraînement (`train_dataset`) et le dataset de validation (`eval_dataset`). Le tokenizer est également fourni pour s'assurer que les données d'entrée sont correctement traitées.

Enfin, nous lançons le processus d'entraînement avec `trainer.train()`, ce qui ajuste les poids du modèle pour minimiser l'erreur sur les données d'entraînement et améliorer la précision de la classification des nouvelles en fausses ou vraies.

```

# Sauvegardons le modèle entraîné dans le répertoire 'outputs'
trainer.model.save_pretrained('outputs')
# Rechargeons le tokenizer
tokenizer = AutoTokenizer.from_pretrained("microsoft/deberta-v3-base" )

```

fig10:code 7.

Après l'entraînement, il est essentiel de sauvegarder le modèle pour une utilisation future. Le modèle entraîné est ainsi sauvegardé dans le répertoire outputs à l'aide de la méthode `save_pretrained` du `Trainer`. Cette étape permet de préserver les paramètres ajustés du modèle après l'entraînement.

Ensuite, nous rechargeons le tokenizer d'origine en utilisant `AutoTokenizer.from_pretrained` avec le modèle `microsoft/deberta-v3-base`. Recharger le tokenizer assure que nous avons une instance cohérente et compatible avec le modèle sauvegardé, facilitant ainsi les prédictions ultérieures ou toute autre manipulation du texte.

Ces étapes garantissent que le modèle et le tokenizer sont correctement alignés pour des tâches de classification futures, maintenant ainsi l'intégrité et la performance du modèle post-entraînement.

## c. Évaluation du Modèle

```
def classify_text(prompt):
    # Tokenizer l'entrée
    inputs = tokenizer(prompt, return_tensors="pt", truncation=True, padding=True, max_length=512).to(model.device)

    # Obtenir les prédictions du modèle
    with torch.no_grad():
        outputs = model(**inputs)

    # Obtenir la classe prédite
    logits = outputs.logits
    probabilities = torch.nn.functional.softmax(logits, dim=-1)
    predicted_class = torch.argmax(probabilities, dim=-1).item()

    return predicted_class
```

*fig11:code 8.*

Pour évaluer notre modèle entraîné, nous avons défini une fonction `classify_text` qui permet de classer un texte donné en utilisant le modèle DeBERTa fine-tuné.

**Tokenisation de l'Entrée:** La fonction `classify_text` commence par tokeniser l'entrée (`prompt`) en utilisant le tokenizer DeBERTa. La tokenisation inclut la troncature et le padding du texte pour qu'il atteigne une longueur maximale de 512 tokens. Les tokens sont ensuite transférés sur le même dispositif que le modèle (par exemple, GPU si disponible).

**Prédictions du Modèle:** Ensuite, nous obtenons les prédictions du modèle en passant les entrées tokenisées à travers le modèle DeBERTa. Cette étape est effectuée sans calcul de gradients (`torch.no_grad()`), ce qui économise de la mémoire et accélère le processus d'inférence.

Obtention de la Classe Prédite: Les logits produits par le modèle sont transformés en probabilités à l'aide de la fonction softmax. La classe prédite est déterminée en prenant l'argument ayant la probabilité la plus élevée (torch.argmax).

```
# Tester la fonction de classification
n = 100 # Index de l'exemple de test
prompt = data['test']['prompt'][n]
predicted_class = classify_text(prompt)
print(f"Predicted class: {predicted_class}")
```

Predicted class: 0

+ Code + Markdown

```
#Vérification
data['test']['label'][100]
```

tensor(0)

fig12:code 9.

Pour tester la fonction de classification, nous avons sélectionné un exemple dans l'ensemble de test (index n = 100) et avons imprimé la classe prédite. Le modèle a prédit correctement la classe pour cet exemple (classe prédite: 0), confirmée par l'étiquette réelle correspondante.

Cette évaluation initiale nous donne une indication de la performance du modèle sur des exemples spécifiques. Des évaluations supplémentaires, incluant des mesures de performance globales sur l'ensemble de validation ou de test, sont nécessaires pour obtenir une vue d'ensemble complète des capacités du modèle.

```
#évaluer la performance du modèle
correct=0
for i in range(100):
    prompt = data['validation']['prompt'][i]
    if classify_text(prompt) == data['validation']['label'][i]:
        correct+=1
    if i%10==0:
        print(i)
# Afficher le pourcentage de précision
print(correct/100)
```

0  
10  
20  
30  
40  
50  
60  
70  
80  
90  
1.0

fig13:code 10.

Pour évaluer la performance de notre modèle DeBERTa fine-tuné, nous avons mis en place un script simple pour calculer la précision sur un sous-ensemble de l'ensemble de validation. Voici une description détaillée de ce processus :

**Calcul de la Précision:** Nous avons défini une boucle qui parcourt les 100 premiers exemples de l'ensemble de validation. Pour chaque exemple, le prompt est extrait et classifié à l'aide de la fonction `classify_text`.

**Comparaison avec les Étiquettes Réelles:** Pour chaque classification, nous comparons la classe prédite par le modèle avec l'étiquette réelle présente dans l'ensemble de validation. Si la prédiction est correcte, nous incrémentons le compteur correct.

**Affichage des Progrès:** Afin de suivre les progrès du processus de validation, nous imprimons l'index à chaque 10ème itération.

**Calcul de la Précision Finale:** Après avoir traité les 100 exemples, nous calculons la précision en divisant le nombre de prédictions correctes par 100.

Le résultat obtenu montre que notre modèle a atteint une précision de 100% sur ce sous-ensemble de validation, indiquant une performance exceptionnelle.

```
[20]: prompt = 'usa is not a country'
      predicted_class = classify_text(prompt)
      print(f"Predicted class: {predicted_class}")

Predicted class: 0
```

*fig14:code 11.*

Pour une évaluation sur un exemple quelconque on utilise le prompt 'usa is not a country', La sortie "Predicted class: 0" indique que le modèle a correctement identifié cette déclaration comme étant une fausse nouvelle.

## 5. Résultats

### a. UI

Nous avons créé une interface utilisateur (UI) simple et intuitive pour notre modèle de détection de fausses nouvelles en utilisant Streamlit. Cette interface permet aux utilisateurs de saisir un texte de nouvelles et de le classifier en temps réel en tant



que "Fake" ou "Real" en utilisant le modèle DeBERTa fine-tuné.

```
%%writefile app.py
import streamlit as st
import torch
from transformers import AutoTokenizer, AutoModelForSequenceClassification

# Définir le modèle afiné
model = AutoModelForSequenceClassification.from_pretrained("outputs")
# Définir le tokenizer
tokenizer = AutoTokenizer.from_pretrained("microsoft/deberta-v3-base")

# Interface Streamlit app
st.title("Fake News Detection with DeBERTa")

uploaded_text = st.text_area("Enter news text here...")
|
if st.button("Classify"):
    if uploaded_text:
        inputs = tokenizer(uploaded_text, return_tensors="pt", truncation=True, padding=True, max_length=512).to(model.device)
        with torch.no_grad():
            outputs = model(**inputs)
            logits = outputs.logits
            probabilities = torch.nn.functional.softmax(logits, dim=-1)
            predicted_class = torch.argmax(probabilities, dim=-1).item()
            class_name = "Fake" if predicted_class == 0 else "Real"
            st.write(f"Predicted class: {class_name}")
    else:
        st.write("Please enter some text to classify.")
```

fig15:code 12.

```
#installation de Node.js, npm et Streamlit, ainsi que le package localtunnel via npm
!apt-get install -y nodejs npm
!npm install -g localtunnel
!pip install streamlit

#On exécute l'application Streamlit localement via localtunnel sur le port 8501.
!streamlit run app.py & npx localtunnel --port 8501
```

fig16:code 13.

## Fake News Detection with DeBERTa

Enter news text here...

Classify

fig17:UI en utilisant Streamlit

Cette interface fournit un moyen pratique et accessible de démontrer les capacités du modèle DeBERTa pour la détection des fausses nouvelles, permettant aux utilisateurs de tester le modèle avec leurs propres exemples de texte.

## b. Test et Validation de l'Application

veuillez consulter la vidéo Démo.



## 6. Conclusion

Ce projet de détection des fausses nouvelles à l'aide du modèle DeBERTa (Decoding-enhanced BERT with disentangled attention) a démontré l'efficacité des modèles de langage avancés pour les tâches de classification de texte. En utilisant un ensemble de données de détection des fausses nouvelles en anglais, nous avons pu fine-tuner le modèle DeBERTa pour distinguer les nouvelles vraies des nouvelles fausses avec une précision remarquable.

Le processus de prétraitement des données a été essentiel pour structurer les informations et préparer les données pour l'entraînement. En combinant le titre, le sujet et le texte des nouvelles, nous avons créé des prompts complets qui ont permis au modèle de mieux comprendre le contexte et de faire des prédictions plus précises.

L'entraînement du modèle a été réalisé en utilisant les bibliothèques transformers de Hugging Face, avec des paramètres d'entraînement soigneusement choisis pour optimiser la performance. Le modèle a été évalué sur un ensemble de validation et a montré des résultats prometteurs, confirmant sa capacité à détecter les fausses nouvelles de manière efficace.

Nous avons également développé une interface utilisateur simple avec Streamlit, permettant à quiconque de tester le modèle en entrant du texte de nouvelles et en recevant une classification instantanée. Cette interface rend notre solution accessible et facile à utiliser pour un large public.

En conclusion, ce projet illustre non seulement l'efficacité des modèles de langage avancés comme DeBERTa pour la détection des fausses nouvelles, mais aussi l'importance du prétraitement des données et d'une interface utilisateur intuitive. Les résultats obtenus montrent que les modèles de langage basés sur les Transformers peuvent être des outils puissants dans la lutte contre la désinformation. À l'avenir, des améliorations peuvent être apportées en explorant d'autres architectures de modèles et en utilisant des ensembles de données plus diversifiés pour améliorer encore la robustesse et la précision de notre solution.

# 7. Références

*Transformers in Large Language Models(LLMs)*

<https://medium.com/@varsha.rainer/transformers-8ee4bcd10ab>

*Attention Is All You Need*

<https://arxiv.org/pdf/1706.03762>

*Gemma: Open Models Based on Gemini Research and Technology*

<https://arxiv.org/pdf/2403.08295>

*DEBERTA: DECODING-ENHANCED BERT WITH DISENTANGLED ATTENTION*

<https://arxiv.org/pdf/2006.03654v6>

*Datasets:ErfanMoosaviMonazzah/fake-news-detection-dataset-English*

<https://huggingface.co/datasets/ErfanMoosaviMonazzah/fake-news-detection-dataset-English>

*microsoft/deberta-v3-base*

<https://huggingface.co/microsoft/deberta-v3-base>

*Inspiration pour le notebook*

[https://github.com/oppasource/ycopie/blob/main/LLM\\_Series/Llama-2\\_Finetuning/llama-2\\_finetuning.ipynb](https://github.com/oppasource/ycopie/blob/main/LLM_Series/Llama-2_Finetuning/llama-2_finetuning.ipynb)