# Génération de Texte à partir des Tweets de Donald Trump

Abdelmouhaimen Sarhane
Abdelhakim Ourkia

Département Sciences du Numérique - Troisième année
2024-2025

# Table des matières

**Résumé**

Ce rapport présente notre projet visant à générer du texte basé sur les tweets de Donald Trump. Nous décrivons les pré-traitements effectués, l'architecture utilisée, et les résultats obtenus, ainsi que des comparaisons entre différentes approches et hyperparamètres.

# 1 Introduction

Ce projet porte sur la génération de texte à l'aide de techniques d'apprentissage profond, appliquées à un ensemble de tweets de Donald Trump. L'objectif principal est de concevoir un modèle capable de générer des phrases cohérentes, tout en imitant le style et les particularités linguistiques des tweets originaux.

# 2 Pré-traitements

## 2.1 Chargement des données

Les données ont été téléchargées à partir d'une base publique contenant les tweets de Donald Trump. Voici un aperçu des premières lignes de données :

```
"Be sure to tune in and watch Donald Trump on Late Night..."
"Donald Trump will be appearing on The View tomorrow morning..."
```

## 2.2 Filtrage et Nettoyage

Après le chargement des tweets depuis la base de données, nous avons constaté la présence de caractères qui ne correspondent ni à la langue française ni à l'anglais. Afin d'éviter que ces tweets en d'autres langues affectent négativement l'apprentissage du modèle, nous avons ajouté une étape de nettoyage des données pour ne conserver que les tweets rédigés en français ou en anglais. Toutefois, ces tweets représentant une minorité dans la base de données, leur suppression n'a pas significativement réduit la quantité de données d'entraînement.

Voici un exemple du vocabulaire extrait de la base de données après un tokenization caractère par caractère, avant et après nettoyage :

— **Avant nettoyage** :
<sot> !"#$%&'()*+,-./0123456789:;<=>?´º≫½ÉáâèéíïñòóôöøúğıÍō|‘’\"•...€™<eot>

— **Après nettoyage** :
<sot> !"',-.0123456789?ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz<eot>

## 2.3 Tokenization

Dans notre approche de tokenization, nous avons utilisé deux techniques distinctes. Le premier tokenizer est basé sur une tokenization caractère par caractère. Pour ce faire, nous avons concaténé tous les tweets nettoyés en un seul texte et extrait les caractères uniques présents dans l'ensemble des tweets pour construire un vocabulaire. Cette méthode nous a permis de capturer une large variété de caractères présents dans les tweets tout en construisant un vocabulaire spécifique à notre jeu de données.

Le deuxièm tokenizer utilise l'outil SentencePiece, un algorithme qui permet de créer un modèle de tokenization basé sur des sous-units, telles que des caractères ou des paires de caractères. Après avoir concaténé tous les tweets nettoyés dans un fichier, nous avons entraîné un modèle SentencePiece en utilisant la méthode SentencePieceTrainer.Train(). Ce modèle a été entraîné avec des paramètres spécifiques pour générer un vocabulaire de 8000 unités et une couverture de caractères de 0.9995, ce qui garantit une bonne représentation des caractères présents dans les tweets, tout en permettant d'éviter de traiter des caractères trop rares ou non

pertinents. Le modèle généré utilise la méthode Byte Pair Encoding (BPE), qui est un type de tokenization qui fusionne progressivement les paires de caractères les plus fréquentes pour créer des unités de vocabulaire plus grandes et plus significatives.

### 2.4 Préparation des données pour l'entraînement

Pour préparer les données en vue de l'entraînement du modèle, nous avons d'abord utilisé une fonction de prétraitement, `data_preprocessing`, qui prend en entrée un ensemble de tweets nettoyés et un dictionnaire de tokens (`token_to_id`). Cette fonction procède à l'encodage des tweets en utilisant les indices des tokens. Chaque tweet est encodé en ajoutant des jetons spéciaux `<sot>` (Start of Tweet) au début et `<eot>` (End of Tweet) à la fin, afin de marquer respectivement le début et la fin de chaque séquence. Les tweets sont ensuite convertis en une séquence d'indices.

Nous avons ensuite concaténé toutes les séquences d'indices pour former un grand corpus de texte encodé. Parallèlement, nous avons conservé l'indice du début de chaque tweet, correspondant à chaque séquence dans la grande liste d'indices. Ce corpus d'indices est utilisé pour constituer la variable d'entrée $X$, où chaque ligne de $X$ correspond au début d'un tweet et s'étend jusqu'à atteindre la taille du contexte spécifiée (`context_size`). Ce découpage permet de générer des fenêtres de texte adaptées à l'apprentissage, chaque entrée représentant un sous-ensemble du texte, prêt à être utilisé pour entraîner le modèle.

Pour construire les labels $Y$, il suffit de décaler $X$ d'un token.

## 3 Architecture du Modèle

### 3.1 Description

L'architecture du modèle que nous avons développé pour ce projet repose sur une variante de **GPT (Generative Pre-trained Transformer)**. Ce modèle est constitué de plusieurs modules clés : l'embedding des tokens et des positions, des blocs Transformer, ainsi que des couches de normalisation et de sortie pour la génération de texte. Le processus commence avec la classe `TokenAndPositionEmbedding`, qui génère les embeddings des tokens et des positions. Chaque token est mappé à un vecteur dense de dimension `embedding_dim` à l'aide de la couche `nn.Embedding`, et la position de chaque token dans la séquence est également encodée via une autre couche `nn.Embedding`, ce qui permet de tenir compte de l'ordre des tokens dans la séquence. Ces embeddings sont ensuite additionnés pour obtenir la représentation finale des tokens enrichie de leur position.

Les séquences de tokens sont ensuite traitées par une série de blocs Transformer via la classe `TransformerBlock`. Chaque bloc contient une couche d'attention multi-têtes qui permet au modèle d'examiner la séquence de manière parallèle et de se concentrer sur les relations entre les différents tokens. Un masque est appliqué pour empêcher l'attention sur les tokens futurs pendant l'entraînement, garantissant une attention causale. Les résultats de l'attention sont ensuite passés à un réseau de neurones feedforward, composé de deux couches linéaires avec une activation ReLU, et suivis d'une normalisation de couche pour stabiliser l'apprentissage.

Enfin, après avoir traversé tous les blocs Transformer, les sorties sont normalisées à l'aide d'une couche de normalisation finale `nn.LayerNorm` et sont passées à une couche linéaire, qui génère les logits correspondant à chaque token du vocabulaire, servant à la prédiction du prochain token dans la séquence.

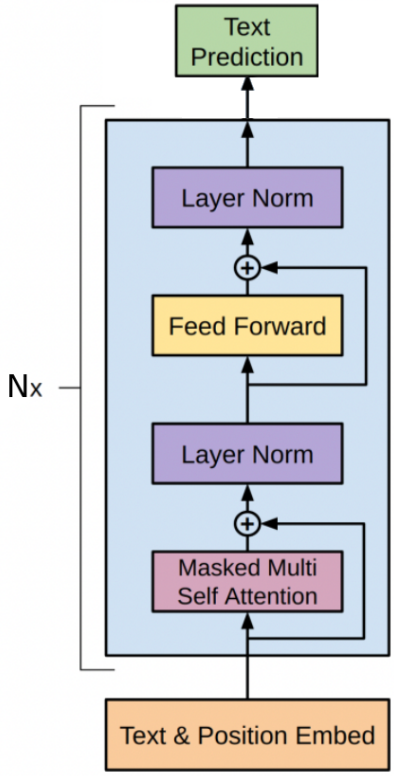


FIGURE 1 – Schéma de l'architecture finale utilisée.

### 3.2 Les Hyperparamètres

Les différents hyperparamètres que nous avons définis sont :
$batch\_size = 30$
$context\_size = 512$
$epochs = 150$
$learning\_rate = 1e-4$
$n\_embd = 150$
$n\_head = 3$
$n\_layer = 2$
Notez que `n_layer` correspond au nombre de blocs Transformer dans notre modèle.

## 4 Entraînement et optimisation

### 4.1 Fonction de perte et algorithme d'optimisation

Pour l'entraînement du modèle, nous avons utilisé la fonction `CrossEntropy` pour calculer la perte, ce qui est adapté pour les tâches de classification multi-classes, comme la prédiction du prochain token dans une séquence. En ce qui concerne la méthode d'optimisation, nous avons choisi l'optimiseur `Adam` qui est un optimiseur adaptatif largement utilisé pour ses performances efficaces.

### 4.2 Courbes de perte

La Figure 2 présente la courbe de la perte durant l'entraînement avec le Tokenizer par caractère. On observe une diminution progressive de la perte, indiquant une bonne adaptation du modèle avec une valeur environ 1.33

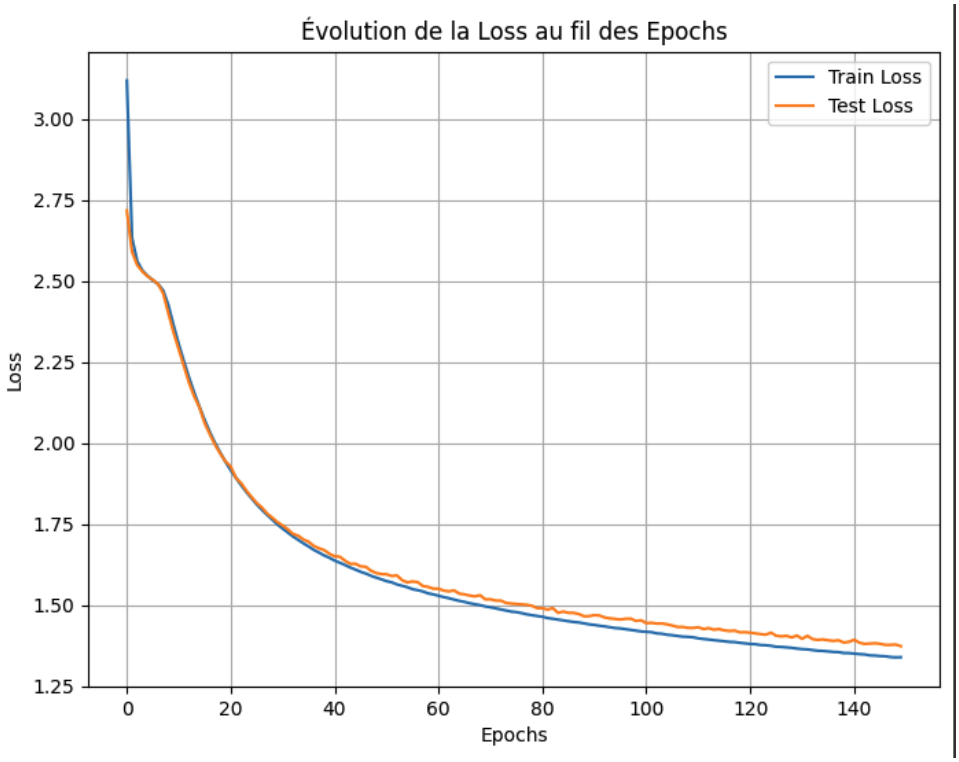


FIGURE 2 – Évolution de la fonction de perte durant l'entraînement avec le premier tokenizer par caractère

Cette deuxième figure 3 présente la courbe de la perte durant l'entraînement avec le Tokenizer SentencePiece, on observe une convergence rapide lors de l'entraînement. L'utilisation du SentencePiece Tokenizer a conduit à une meilleure perplexité.

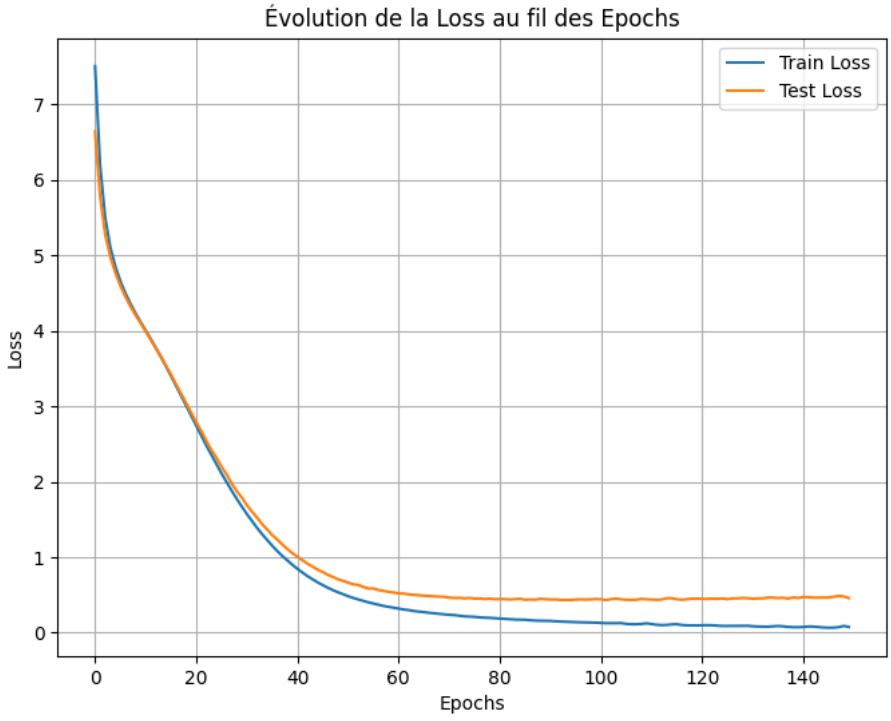


FIGURE 3 – Évolution de la fonction de perte durant l'entraînement avec SentencePieceTokenizer

D'après les deux courbes de la perte des deux tokenizers, on peut constater l'importance du tokenizer dans la convergence du modèle. En effet, comme le montrent les deux figures, le tokenizer `SentencePiece` favorise une convergence plus rapide du modèle par rapport au tokenizer caractère par caractère. Cette différence peut être expliquée par la difficulté pour le modèle d'apprendre le sens des phrases uniquement à partir des caractères, contrairement à `SentencePiece`, qui utilise une unité plus significative facilitant ainsi l'apprentissage du modèle.

# 5 Résultats et Analyse

## 5.1 Résultats Qualitatifs

Quelques exemples de phrases générées par le **tokenizer caractère par caractère** :

— "**Donald** Trump National Committee Committee, just created them oper operations a great military and will allow this a second chance chance ! chance !

— "**Trump** International Golf Links was just rated one of the greatest courses in the world world. Virtually all reviews are saying the same thing thing. thing. "

— "**I am happy to announce that the original Apprentice is coming back** to Old work at a great to largent by SOP is far National Govn Law Lincolment, farmies time in White Hillary great for 9 pm."

Les résultats montrent que la tokenization caractère par caractère entraîne une génération de texte incohérente, avec des phrases souvent grammaticalement incorrectes et des mots inexistants. Le modèle peine à maintenir la structure syntaxique et produit des enchaînements aléatoires sans réelle compréhension contextuelle. Bien que certaines phrases commencent correctement, la suite devient erratique, confirmant un manque de capture des relations à long terme.

Quelques exemples de phrases générées par le **tokenizer SentencePiece** :

— "**Donald** just line apaid annored a just be nices agree "

— "**Trump** is autis being to always never will big people us before now nothing capaign high U.S. J. By Team, and other some chiday are FAX"

L'analyse de ces phrases montre que, malgré l'utilisation d'un tokenizer basé sur des sous-mots (SentencePiece), le modèle génère toujours des séquences sans signification claire. Les erreurs restent similaires à celles observées avec la tokenization caractère par caractère : juxtaposition de mots sans lien sémantique, création de mots inexistants et structure grammaticale incohérente.

## 5.2 Résultats Quantitatifs

Nous avons utilisé le score BLEU pour évaluer la performance de notre modèle sur un ensemble de tweets servant de références. Nous avons retiré la moitié de chaque tweet et l'avons fourni à notre modèle afin qu'il génère la partie manquante.

| **Méthode** | **BLEU score** | **Test Accuracy** |
|---|---|---|
| Tokenizer caractère par caractère | 59.4% | 59.4% |
| Tokenizer SentencePiece | 42% | 93.7% |

TABLE 1 – Comparaison des performances entre différentes tokenizations.

L'analyse des résultats quantitatifs montre que le score BLEU est plus élevé avec la tokenization caractère par caractère (59.4%) qu'avec SentencePiece (42%), bien que SentencePiece

affiche une très haute précision sur l'ensemble de test (93.7%). Cela indique que, bien que le modèle entraîne mieux les données avec SentencePiece, sa capacité de généralisation et la fluidité des phrases générées restent limitées.

# 6 Conclusion et Perspectives

Ce projet a permis de développer un modèle de génération de texte basé sur des tweets de Donald Trump en utilisant une architecture GPT avec Transformer. L'évaluation a montré que la tokenization caractère par caractère limite la capacité du modèle à produire des phrases cohérentes, entraînant des erreurs syntaxiques et sémantiques. Malgré une certaine fidélité au style des tweets, le manque de continuité et l'apparition de mots inexistants confirment les limites de cette approche.
L'expérimentation avec SentencePiece n'a pas permis d'améliorer significativement la qualité des prédictions, bien que la précision du modèle sur l'ensemble de test ait été bien plus élevée. Cela suggère que la métrique de précision seule ne suffit pas à évaluer la pertinence des phrases générées.

***Améliorations possibles :***
— Explorer des modèles plus avancés comme GPT-3 ou GPT-4.
— Affinement des hyperparamètres pour optimiser la convergence et la stabilité du modèle.
— Essayer d'autres Tokenizers
— Tester des jeux de données plus variés.
— Affiner les métriques d'évaluation

# Références

— Vaswani et al., "Attention is All You Need", 2017.