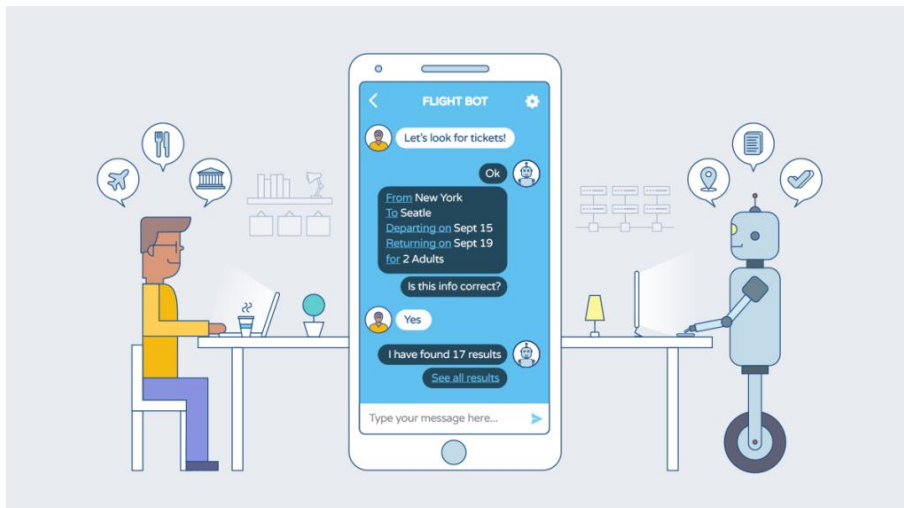


Département des mathématiques et de  
L'informatique  
Master Sciences des données et systèmes  
Intelligents (SDSI)

# Chatbot Friends basé sur GPT



Réalisé par :  
Najoua ELFETHI  
Noura YOUSFI

Encadré par :  
Mr. Anas El ansari

Année Universitaire : 2023-2024



الكلية متعددة التخصصات الناكور  
+٠٣٤٧٧٠١٦ +٠٣٤٤٣٨١٤٦ I ١١.٤٣٢  
Faculté Pluridisciplinaire de Nador

CHAPITRE 5. IMPLÉMENTATION DU CHATBOT FRIENDS	21
5.1 Description des attributs	22
5.2 Les outils et les bibliothèques utilisées	22
CHAPITRE 6. ANALYSE EXPLORATOIRE ET PRÉPARATION DES DONNÉES	26
6.1 Importation des bibliothèques nécessairese	27
6.2 Chargement du dataset	27
6.3 Analyse Exploratoire des données	28
6.3.1 Analyse de forme	28
6.3.2 Analyse de fond	29
6.4 Préparation des données	34
CHAPITRE 7. MODÉLISATION	37
7.1 Obtenir les données contextuelles du Joey Tribbiani	38
7.2 Installation de la bibliothèque transformers	38
7.3 Importation des bibliothèques	39
7.4 Chargement du modèle	39
7.5 Division du dataset en parties d'entraînement et test	39
7.6 Réduire la taille du train et test set	40
7.7 Convertir jeu de données en une format approprié	40
7.8 Gestion des checkpoints et du cache	41
7.9 Configuration du modèle	42
7.10 Entraînement du modèle	43
7.11 Evaluation du modèle	43
7.12 Discuter avec Joey Tribbiani	44
CHAPITRE 8. DÉPLOIEMENT	45
8.1 Installation de la bibliothèque du Telegram	46
8.2 Création du Chatbot Telegram	46
8.3 Conversation avec Joey Tribbiani sur Telegram	47
Conclusion générale	48

# Introduction générale

Dans le cadre de l'ère numérique actuelle, les chatbots ont émergé comme des outils puissants et polyvalents pour améliorer l'interaction utilisateur et automatiser les services de communication. Ce rapport porte sur une application chatbot innovante qui permet aux utilisateurs de communiquer avec un personnage emblématique de la série télévisée bien-aimée "Friends". Cette application offre une expérience immersive en recréant les dialogues et les interactions avec l'un des personnages principaux de la série.

L'objectif principal de Chatbot est de capturer l'esprit et l'humour unique du personnage de "Friends" et de le rendre accessible à travers des conversations dynamiques et engageantes. Destiné aux fans de la série, ce chatbot vise à fournir un divertissement interactif et à renforcer la nostalgie pour l'une des sitcoms les plus populaires de tous les temps.

L'application est conçue pour répondre à une variété de demandes des utilisateurs, allant de simples conversations à des conseils humoristiques et des anecdotes, en passant par des références culte de la série. Grâce à l'utilisation de technologies avancées de traitement du langage naturel (NLP) et de modèles d'apprentissage automatique, Chatboot parvient à reproduire fidèlement le style de conversation et la personnalité du personnage, offrant ainsi une expérience utilisateur authentique et mémorable.

Ce rapport détaillera les fonctionnalités principales de Chatboot, les technologies sous-jacentes, le processus de développement, ainsi que les défis rencontrés et les solutions mises en place. De plus, une évaluation de la performance et des cas d'utilisation illustratifs seront présentés pour démontrer l'efficacité et l'impact de cette application unique.

## Chapitre 1 : Présentation de projet

---

*Dans ce chapitre introductif, nous plongerons dans le projet de développement d'un chatbot basé sur le célèbre sitcom « Friends ». Ce projet explore l'utilisation d'un ensemble de données de la série pour créer une expérience conversationnelle unique, mettant en avant les défis et les stratégies de développement utilisées.*

## 1.1 Cadre de projet

Nous définirons le cadre du projet de développement d'un chatbot utilisant le dataset de la série Friends. Nous explorerons les objectifs visés, la méthodologie prévue, et les technologies spécifiques qui seront employées pour créer une expérience interactive captivante, inspirée par les personnages et les dialogues emblématiques de la série.

Ce chapitre établit le cadre du projet de développement d'un chatbot utilisant le dataset de la série Friends. Le projet vise à créer une interface conversationnelle interactive inspirée des personnages et des dialogues de la série télévisée. Nous explorerons les éléments suivants :

- **Objectifs du Projet :**

Créer un chatbot capable de dialoguer de manière naturelle en s'appuyant sur les dialogues de Friends. Offrir une expérience utilisateur immersive et divertissante en recréant l'ambiance et l'humour distinctifs de la série.

- **Méthodologie :**

Analyse et prétraitement des données : Nettoyage des scripts pour extraire les dialogues pertinents, suppression des indications scéniques et des annotations.

Modélisation du chatbot : Utilisation de techniques de traitement du langage naturel (NLP) pour comprendre et générer des réponses basées sur les interactions des personnages.

Évaluation et itérations : Test du chatbot pour améliorer sa compréhension et sa capacité à répondre de manière contextuelle.

- **Technologies Utilisées :**

Python : Langage principal pour le développement.

Bibliothèques : Utilisation de NLTK, NumPy, seaborn, sklearn ...etc

- Plateforme de développement : Google Colab

## 1.2 Contexte et problématique

Dans cette partie, nous explorerons le contexte entourant le projet de chatbot basé sur le dataset de la série Friends, ainsi que la problématique principale à résoudre.

### Contexte :

Le projet s'inscrit dans un contexte où les chatbots et les interfaces conversationnelles deviennent de plus en plus populaires pour l'interaction utilisateur. En utilisant les dialogues riches et variés de Friends, nous cherchons à capturer l'essence unique des personnages et à offrir aux utilisateurs une expérience nostalgique et engageante

### Problématique :

La problématique du dataset « Friends » est de prédire quelle réplique ou réponse d'un personnage de la série serait appropriée dans un contexte de conversation donné. Il s'agit d'un problème de traitement du langage naturel où nous devons générer ou sélectionner une réponse pertinente à partir d'un large éventail de dialogues et de situations rencontrées dans la série. Ce dataset permet de former un chatbot capable de comprendre et de réagir de manière contextuellement appropriée en imitant le style et le ton des personnages de Friends.

## 1.3 Objectifs

Les objectifs spécifiques que nous visons à atteindre avec le projet de chatbot basé sur le dataset de la série Friends :

- Création d'une Interface Interactive
- Compréhension du Langage Naturel
- Optimisation de la Performance
- Intégration et Déploiement

## Chapitre 2 : Choix méthodologique

---

*Dans ce chapitre, nous explorerons notre approche méthodologique pour le développement du chatbot basé sur le dataset de la série Friends. Nous commencerons par détailler le Cycle de Vie CRISP, adapté à notre projet pour guider depuis la compréhension des besoins jusqu'à l'évaluation des résultats. Ensuite, nous aborderons le processus KDD pour la sélection, la préparation et l'interprétation des données de dialogue. Enfin, nous discuterons de notre choix méthodologique spécifique, intégrant des techniques avancées de NLP et une approche itérative pour optimiser continuellement les performances du chatbot tout en répondant aux exigences d'une interaction conversationnelle immersive et fidèle à l'esprit de la série.*



## 2.1 Cycle de Vie CRISP (Cross-Industry Standard Process for Data Mining)

Le Cycle de Vie CRISP est un cadre méthodologique robuste largement utilisé dans l'exploration de données. Adapté spécifiquement à notre projet de chatbot Friends, ce cycle comprendra les étapes suivantes :

- Compréhension des Besoins Métier :

Analyse des objectifs et des besoins spécifiques pour le chatbot basé sur les dialogues de la série Friends.

Compréhension des Données :

Collecte des données à partir des scripts et des dialogues de la série Friends.

Exploration préliminaire pour comprendre la structure des données et les défis potentiels liés à l'analyse.

- Préparation des Données :

Nettoyage des données pour éliminer les indications scéniques, les annotations et autres éléments non pertinents.

Transformation des données pour structurer les dialogues en formats utilisables par les algorithmes de traitement du langage naturel (NLP).

- Modélisation des Données :

Utilisation de techniques avancées de NLP pour analyser et extraire des informations significatives des dialogues.

Développement de modèles de langage et de réponse pour permettre au chatbot de générer des réponses cohérentes et contextuellement appropriées.

- Évaluation des Modèles :

Évaluation rigoureuse des performances des modèles à l'aide de mesures telles que la précision, le rappel et la pertinence des réponses générées.

Validation par des tests utilisateurs pour ajuster et améliorer les performances du chatbot.

Déploiement et Suivi :

Suivi continu des performances et des interactions utilisateur pour assurer une expérience utilisateur optimale et une évolution continue du chatbot.

## 2.2 KDD (Knowledge Discovery in Databases)

Pour notre projet il est possible aussi d'appliquer le processus KDD, Voici comment chaque étapes sera détaillée :

### ○ **Sélection des Données :**

- Identification et collecte rigoureuse des dialogues pertinents de la série Friends à partir des scripts disponibles.
- Assurer que les données sélectionnées représentent fidèlement les interactions et les caractéristiques linguistiques des personnages.

### ○ **Prétraitement des Données :**

- Nettoyage des données pour éliminer les indications scéniques, les annotations et autres éléments non essentiels qui pourraient affecter l'analyse.
- Standardisation et normalisation des données pour garantir la cohérence et la qualité des dialogues utilisés.

### ○ **Transformation des Données :**

- Transformation des dialogues structurés en un format adapté aux algorithmes de traitement du langage naturel (NLP).
- Utilisation de techniques de réduction de dimensionnalité ou d'encodage pour améliorer la performance des modèles de NLP.

### ○ **Data Mining :**

- Application d'algorithmes avancés de NLP pour extraire des motifs linguistiques, des thèmes récurrents et des relations entre les personnages à partir des dialogues.
- Exploration des données pour découvrir des informations cachées et des insights utiles pour enrichir l'interaction du chatbot.

### ○ **Évaluation des Modèles :**

- Validation qualitative par des tests utilisateurs pour évaluer la pertinence et la satisfaction des réponses du chatbot dans différents scénarios d'utilisation.

## 2.3 Choix de la méthodologie

On a choisi de travailler avec la méthodologie CRISP parce qu'il semble être un choix robuste et approprié pour notre projet de chatbot basé sur les dialogues de la série Friends, en offrant une structure méthodologique claire et adaptable pour gérer et optimiser toutes les phases de développement et d'implémentation du chatbot.

## Chapitre 3 : Introduction aux Chatbots

---

*Dans ce chapitre nous explorons les fondements essentiels des chatbots, débutant par une définition précise de ces agents conversationnels automatisés. Ensuite, l'histoire de leur évolution. Nous examinerons également les architectures typiques des chatbots. Une exploration des différents types de chatbots suivra, illustrant leur diversité fonctionnelle et leurs applications variées. Enfin, nous citons les terminologies couramment utilisées dans le développement des chatbots, facilitant ainsi une compréhension approfondie des concepts abordés dans ce chapitre.*

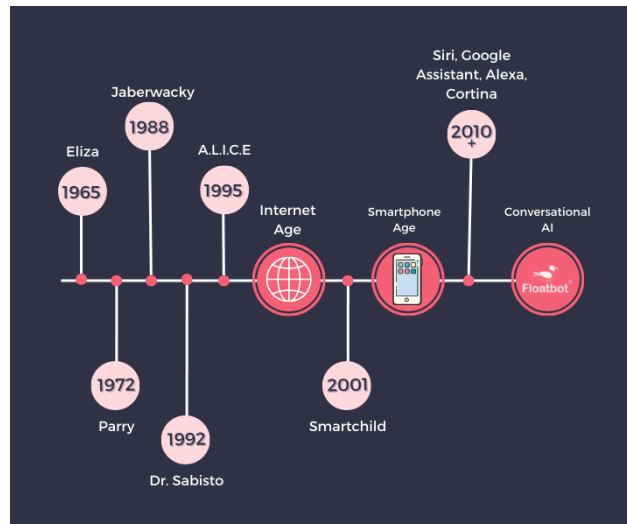
### 3.1 Définition

Les chatbots sont des programmes informatiques conçus pour interpréter le langage naturel des utilisateurs et générer des réponses pertinentes et intelligentes. Ils peuvent fonctionner sur des moteurs basés sur des règles ou utiliser des techniques d'intelligence artificielle (IA). Ces agents conversationnels interactifs sont souvent intégrés à diverses plateformes de messagerie comme Facebook Messenger, Slack, Skype, et Microsoft Teams, via des API accessibles aux développeurs.

Ces dernières années, avec les avancées en technologie vocale, des entreprises telles que Google, Apple et Amazon ont lancé des agents vocaux intelligents. Siri d'Apple, tandis que Google propose Google Home, et Amazon a introduit Alexa, tous utilisés pour des tâches variées comme la gestion des calendriers, le contrôle des lumières, ou la commande de services comme des voitures de location et la lecture de musique sur Spotify.

### 3.2 Bref historique du Chatbots

En 1966, Joseph Weizenbaum a développé ELIZA le tout premier CHATBOT introduit avant le développement du premier ordinateur. Selon un ensemble de règles définies, ELIZA traite les mots-clés reçus en entrée et déclenchant ensuite les réponses de sortie. Plusieurs CHATBOT encore utilise cette méthodologie pour générer des résultats. Après ELIZA, PARRY est arrivé relativement vite appeler ensuite « ELIZA avec attitude ». Dans 1995, ALICE ou Alicebot, l'inspiratrice d'ELIZA, évoluée par Richard Wallace. Bien qu'il négligée pour passer à travers l'évaluation de Turing, ALICE est restée l'une des plus enracinées dès son genre et récompensé à plusieurs reprises par le prix Loebner, un concours annuel d'IA. Cette évolution est illustrée dans la figure ci-dessous.



**Figure 1 : Historique du Chatbots**

### 3.3 Architecture du Chatbot

Le processus en arrière-plan du Chatbot commence par l'appel de l'utilisateur, par exemple : « Qu'est-ce que NLP? » au BOT déployé sur l'application du système de messagerie comme Facebook, Telegram, WhatsApp, site Web, etc, ou à l'appareil en utilisant le speech comme input par exemple Google Assistant, Amazon Alexa. Après réception de la demande de l'utilisateur, le Natural Language Understanding (NLU) l'analyse ou le mappe à l'intention de l'utilisateur et rassemble des informations connexes supplémentaires (intent : « traduire », entités : [word :« NLP »]). Une fois qu'un CHATBOT atteint le score d'interprétation ou de confiance de haut niveau, il doit décider comment procéder et réagir en conséquence. Il peut agir directement sur de nouvelles informations, rappeler ce qu'il a compris et attendre de voir ce qui se passe ensuite, exiger plus d'informations contextuelles, ou demander des éclaircissements. Par exemple, « Demande de l'utilisateur pour réserver un train billet de Casablanca à Nador », mais pour réserver un billet, d'autres informations supplémentaires sont également requises comme la date du voyage, l'heure du voyage. Lorsqu'il y a une compréhension claire de la demande, une action supplémentaire et la récupération des informations ont lieu. Après avoir récupéré les données, BOT destiné à effectuer les actions demandées ou à récupérer les données d'intérêt à partir de ses sources de données, une base de données de base de connaissances BOT ou un appel API qui accède à des ressources externes. Le système de gestion du dialogue conserve les informations sur toutes les conversations avec les utilisateurs.

Cette architecture est illustrée ci-dessous.

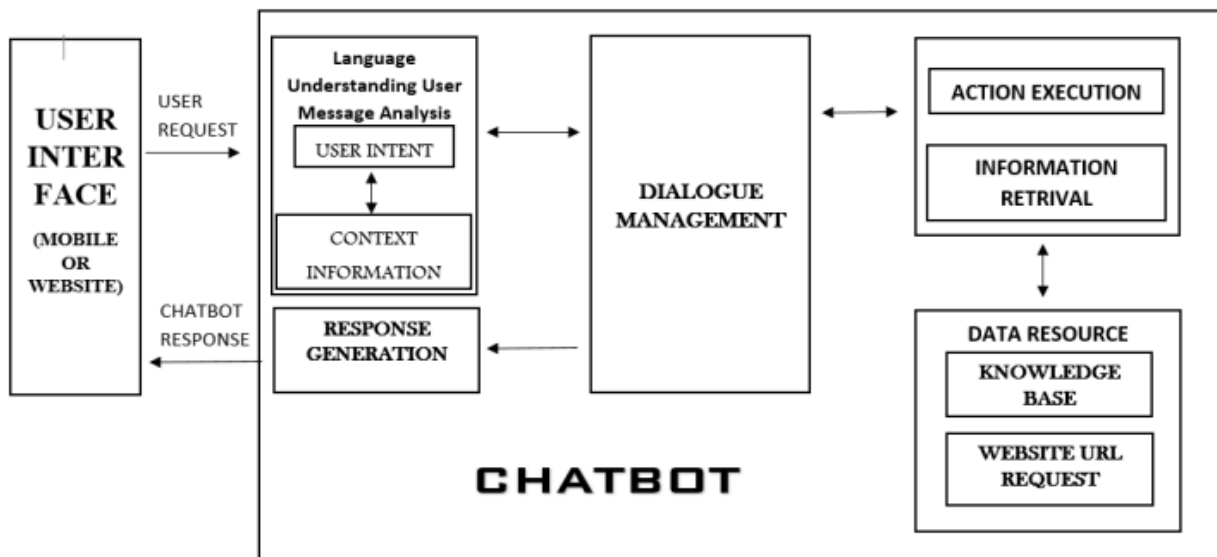


Figure 2 : Architecture du Chatbot

### 3.4 Types du Chatbot

Les chatbots peuvent être classés selon plusieurs variables, notamment le niveau d'interaction et la manière dont les réponses sont générées. Voici une classification schématique des chatbots :

- **Domaine de Connaissance :**
  - ✓ **Domaine Ouvert (Open Domain) :** Capables de traiter des sujets généraux et de répondre de manière appropriée.
  - ✓ **Domaine Fermé (Closed Domain) :** Focalisés sur une seule aire spécifique de connaissance, comme la réservation de vols, et peuvent ne pas répondre à des questions en dehors de ce domaine.
- **Service Fourni :**
  - ✓ **Interpersonnel :** Pour la communication et services comme la réservation de tables ou de trains.
  - ✓ **Intrapersonnel :** Intégrés dans des applications de messagerie personnelle comme Facebook Messenger, Telegram, et WhatsApp, gérant des tâches personnelles comme la gestion de calendrier.
  - ✓ **Inter-agent :** Nécessaires pour la communication entre différents chatbots, comme l'intégration entre Alexa et Cortana.
- **Objectif Basé sur le But :**
  - ✓ **Informatif :** Fournit des informations ou des données à partir d'une base de données fixe, comme les FAQ ou les inventaires en entrepôt.
  - ✓ **Conversationnel :** Interagit avec l'utilisateur de manière similaire à une conversation humaine, utilisant des techniques comme les questions croisées et la politesse.
  - ✓ **Basé sur les Tâches :** Effectue des tâches spécifiques comme la réservation d'une chambre d'hôtel ou la réservation d'une table au restaurant.
- **Méthode de Génération de Réponse :**
  - ✓ **Méthodes d'Intelligence :** Utilisent des systèmes experts pour générer des réponses, en utilisant la compréhension du langage naturel pour interpréter les requêtes des utilisateurs.
  - ✓ **Systèmes Basés sur des Règles :** Interagissent avec les utilisateurs selon un plan prédéfini, anticipant les requêtes et déterminant les réponses en fonction d'un arbre de conversation.
  - ✓ **Hybrides :** Combinent des règles et des techniques d'apprentissage automatique, utilisant par exemple un flux de conversation prédéfini tout en utilisant le traitement du langage naturel pour répondre.

La figure ci-dessous regroupe les différents types du Chatbot.

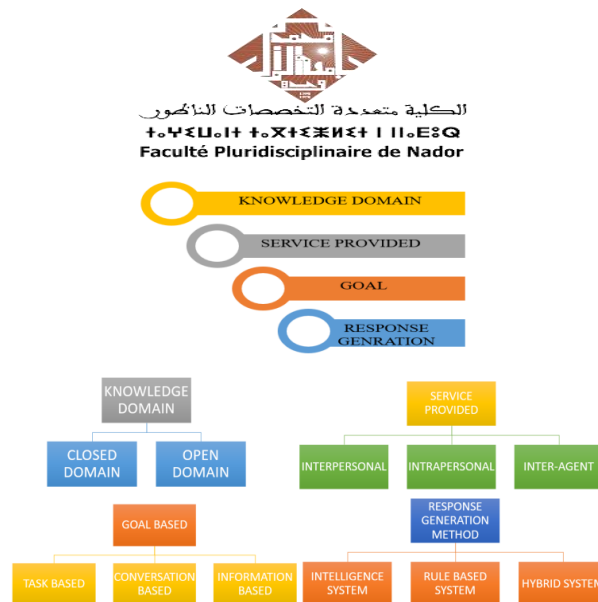


Figure 3 : Types du Chatbot

### 3.5. Terminologies courantes utilisées dans Chatbot

- **Intents :**

Les intentions sont des déclarations d'utilisateur potentielles qui déclenchent des actions ou des objectifs spécifiques lors de l'interaction avec un chatbot. Ils représentent les objectifs et les motivations des interactions des utilisateurs avec une application de robot ou un service Web. Les intentions classent les interactions des utilisateurs en deux types principaux :

- ✓ **Recherche d'informations** : les utilisateurs recherchent des informations spécifiques, telles que les horaires des trains ou les conditions météorologiques.
- ✓ **Action** : les utilisateurs ont l'intention d'effectuer une action spécifique, comme réserver une table dans un restaurant ou acheter des billets de cinéma.

- **Entities :**

Les entités sont des modificateurs associés à des intentions qui fournissent un contexte ou des détails supplémentaires à la demande de l'utilisateur. Ils aident le chatbot à comprendre et à traiter des informations spécifiques dans une intention. Par exemple, dans l'intent « book\_movie », les entités peuvent être « film » ou « vol », selon le contexte de la demande de l'utilisateur.

- **Utterance :**

Les énoncés sont des variations d'expressions ou de phrases utilisées par les utilisateurs pour exprimer leurs intentions sous différentes formes. Ils représentent les termes exacts ou les questions posées par les utilisateurs dans divers contextes. Par exemple, les agents de voyages peuvent rencontrer des énoncés tels que « Réservez un vol de Londres à Paris aujourd'hui » ou « Pourriez-vous s'il vous plaît réserver un vol de Londres à Paris aujourd'hui ».

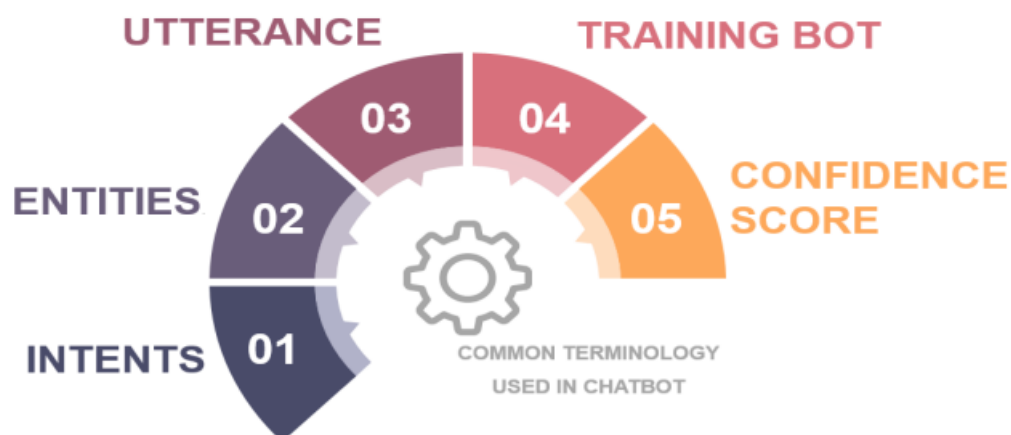
- **Entrainement du Bot :**

Après avoir alimenté les intentions et les entités, le concepteur d'énoncés doit entraîner le Bot.

Pour construire un modèle tel qu'il reconnaîtra l'ensemble existant d'intentions/entités définies lorsque de nouvelles déclarations sont fournies.

- **Score de confiance :**

Le score de confiance est le score qui indique le degré de confiance ou le montant en pourcentage du modèle reconnaît les intentions des utilisateurs avec les intentions sortant dans le système formé base de données.



**Figure 4 : Terminologies utilisées dans Chatbot**



## Chapitre 4: Explication des Transformers

---

*Dans ce chapitre, nous plongerons dans l'univers fascinant des Transformers. Nous commencerons par explorer leur architecture innovante. Nous détaillerons le fonctionnement du mécanisme d'attention, y compris ses variantes comme Self-Attention, Multi-Head Attention et Cross-Attention. Ensuite, nous aborderons le fine-tuning des Transformers, en mettant en lumière DialoGPT, un modèle spécialement conçu pour la génération de dialogues. En parallèle, nous examinerons les Large Language Models (LLMs), leur évolution, leur architecture générale, et nous comparerons leur fonctionnement avec celui des Transformers. Enfin, nous discuterons des différents types de modèles de langage à grande échelle et des perspectives futures de cette technologie révolutionnaire. Ce chapitre vise à offrir une vue d'ensemble complète et approfondie des Transformers et des LLM, mettant en lumière leur impact et leurs applications dans le domaine croissant de l'intelligence artificielle.*

## 4.1. Définition

Ces dernières années, le domaine de l'intelligence artificielle a connu un changement de paradigme avec l'introduction de modèles basés sur des transformateurs. Les transformateurs se sont avérés essentiels au développement de modèles de traitement du langage naturel (NLP) plus précis, tout en étendant également leur employabilité dans d'autres domaines de l'apprentissage automatique.

Le transformateur est une architecture qui repose sur le concept d'attention, une technique utilisée pour attribuer des poids aux différentes parties d'une séquence d'entrée afin d'obtenir une meilleure compréhension de son contexte sous-jacent. Cela permet aux transformateurs d'effectuer des traductions automatiques, des générations de texte et de nombreuses autres tâches NLP.

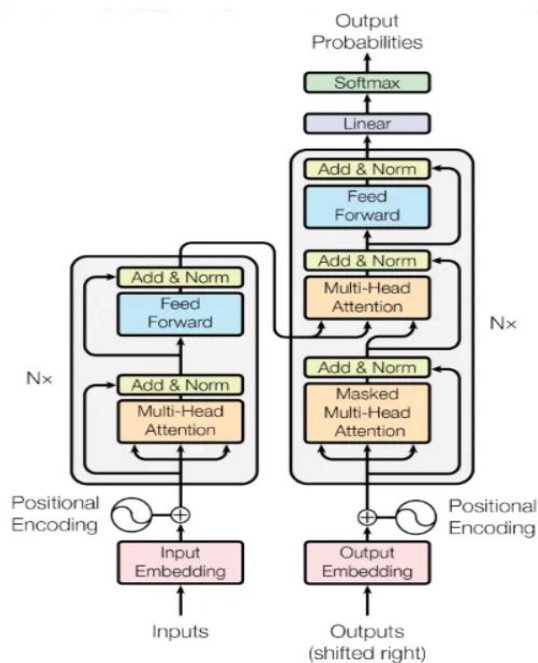
De plus, les transformateurs traitent les entrées en parallèle, ce qui les rend plus efficaces et évolutifs par rapport aux modèles séquentiels traditionnels tels que RNN et LSTM.

## 4.2. Architecture du modèle

Le transformateur suit une structure codeur-décodeur, où un vecteur d'entrée  $x$  est introduit dans un codeur qui produit une représentation cartographiée  $z$ . Avec  $z$ , le décodeur génère une sortie  $y$ , un élément à la fois, c'est-à-dire qu'il génère d'abord  $y_1$ , puis  $y_2$ , jusqu'à  $y_m$ . À chacune de ces étapes, le modèle est autorégressif, ce qui signifie que les sorties générées précédemment affecteront toujours les sorties futures.

Le transformateur suit une architecture contenant des couches d'attention empilées suivies de couches entièrement connectées dans le codeur et le décodeur.

Cette architecture est illustrée ci-dessous.



**Figure 5 : Architecture des Transformers**

- **Encodeur**



L'attention croisée sert de pont entre le codeur et le décodeur. La sortie du codeur est utilisée comme clé et valeur dans la deuxième couche d'attention multi-têtes du décodeur. En outre, la requête est dérivée d'une séquence d'entrée distincte.

## 4.4 Fine-tuning DialoGPT

### 4.4.1 Définition du DialoGPT

DialoGPT, développé par OpenAI, est une variante orientée vers le dialogue du modèle populaire GPT-2. Il a été affiné sur un ensemble de données de Reddit pour fournir une interaction semblable à celle d'un être humain. Il est particulièrement doué pour générer des dialogues cohérents et adaptés au contexte. Contrairement à ses prédécesseurs, il est conçu pour gérer les nuances de la conversation humaine et les interactions engageantes. Contrairement à ses frères, il n'est pas formé à partir de textes Internet, mais uniquement à partir de nos propres conversations sur Reddit.

### 4.4.2 Importance du Fine-tuning

Le réglage fin consiste en fait à entraîner un modèle pré-entraîné sur votre propre ensemble de données, quel qu'en soit le sujet. C'est en fait comme l'entraînement d'un modèle sur vos données. Vous trouvez un modèle vraiment bon et vous voulez y ajouter des connaissances, c'est ce que l'on appelle le réglage fin.

Pour la génération de dialogue, le réglage fin est crucial car il permet au modèle d'apprendre à partir de modèles et de contextes conversationnels spécifiques. Ce processus améliore la capacité du modèle à générer des réponses pertinentes et de haute qualité, adaptées aux besoins de votre application.

### 4.4.3 Choix du modèle

Nous avons choisi de travailler avec Dialo-GPT pour un certain nombre de raisons, dont nous citerons que quelques-unes :

- **Spécialisation dans les dialogues :**

DialoGPT est spécialement conçu pour les conversations et les interactions en langage naturel. En tant que version fine-tunée de GPT-2, il est optimisé pour générer des réponses contextuellement pertinentes et naturelles dans une conversation, ce qui est crucial pour un chatbot basé sur des dialogues comme celui du sitcom Friends.

- **Qualité des réponses :**

Comparé à d'autres modèles, DialoGPT est capable de produire des réponses plus cohérentes et engageantes. Cela est dû à sa formation sur de vastes ensembles de données de conversation, permettant une compréhension plus profonde des nuances conversationnelles et des contextes variés.

- **Flexibilité et adaptabilité :**

DialoGPT peut être fine-tuned avec des ensembles de données spécifiques, comme les dialogues de la série Friends, pour capturer le style unique et l'humour des personnages. Cela permet de créer un chatbot qui répond de manière authentique et fidèle aux personnages de la série.

- **Facilité de déploiement sur Telegram :**

DialoGPT peut être facilement intégré et déployé sur des plateformes de messagerie comme Telegram. Grâce à des bibliothèques Python comme **python-telegram-bot**, la création et le déploiement d'un chatbot sur Telegram deviennent simples et efficaces. Cette intégration permet d'atteindre un large public avec une interface utilisateur conviviale.

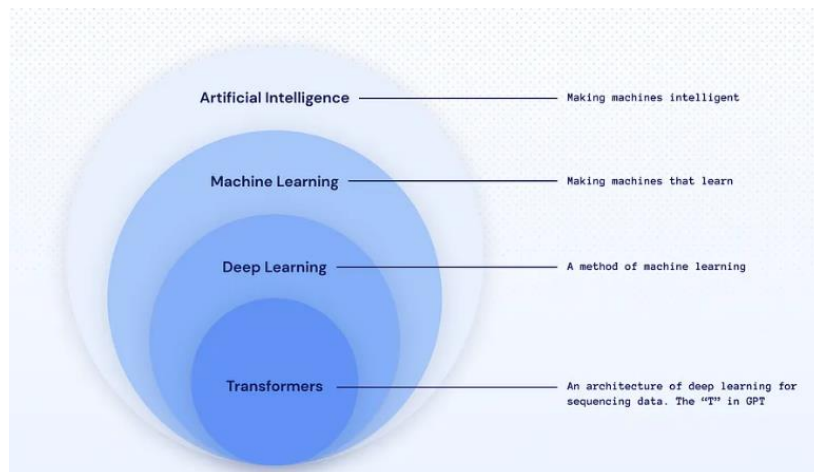
## 4.5 Large Language Models (LLM)

### 4.5.1 Introduction aux LLM et leur évolution

Les modèles de langage à grande échelle (LLM) représentent l'une des innovations les plus fascinantes dans le domaine de l'intelligence artificielle. Ces modèles, capables de comprendre et de générer des textes avec un niveau de sophistication sans précédent, redéfinissent ce qui est possible en matière de traitement du langage naturel. Depuis leurs débuts jusqu'aux versions les plus récentes, les LLM ont connu une évolution remarquable, ouvrant de nouvelles frontières dans l'interaction homme-machine.

### 4.5.2 Fonctionnement des LLM

L'image suivante fournit une représentation claire de la relation entre l'intelligence artificielle, l'apprentissage automatique, l'apprentissage profond et l'architecture Transformer, qui est au cœur du fonctionnement des LLM.



**Figure 6: Relation entre AI, Machine Learning, Deep Learning, et Transformers**

Cette hiérarchie met en évidence le fait que l'apprentissage profond, et en particulier les transformateurs, sont des spécialisations de l'apprentissage automatique, qui est lui-même un sous-ensemble de l'intelligence artificielle.

Les LLM sont basés sur des architectures de transformateurs qui leur permettent de traiter efficacement des séquences de mots. Ils utilisent la formation supervisée et non supervisée pour apprendre à partir de grandes quantités de données textuelles.

### 4.5.3 Types de modèles de langage à grande échelle

Les LLM peuvent être classés de différentes manières :

- **Modèles basés sur des transformateurs** : Tels que GPT et BERT, célèbres pour leur capacité à gérer de vastes contextes linguistiques.
- **Modèles spécifiques à une tâche** : Optimisés pour des tâches spécifiques telles que la traduction, le résumé ou la génération de texte.
- **Modèles multilingues** : Capables de comprendre et de générer des textes dans plusieurs langues.
- **Modèles d'apprentissage zéro et faible** : Conçus pour effectuer des tâches avec peu ou pas d'informations de formation spécifiques.

### 4.5.4 Avenir des LLM

Les LLMs sont appelés à jouer un rôle crucial sur la voie de l'intelligence artificielle générale (AGI). L'AGI fait référence à une intelligence artificielle capable d'effectuer toute tâche intellectuelle qu'un être humain peut accomplir, et les LLM nous rapprochent de cet objectif grâce à leurs capacités avancées de traitement du langage.

## Chapitre 5 : Implémentation du Chatbot Friends

---

*Ce chapitre se concentre sur la réalisation concrète de notre chatbot basé sur les dialogues de la série Friends. Nous commencerons par explorer en détail les données extraites du fichier "Friends.csv", en mettant en avant les aspects clés qui serviront de fondement à notre modèle de chatbot. Ensuite, nous examinerons les choix technologiques et les outils utilisés pour le développement, en mettant en lumière les décisions stratégiques qui ont guidé notre approche. Enfin, nous étudierons les bibliothèques essentielles de traitement du langage naturel (NLP) et d'intégration système intégrées dans notre solution pour assurer la performance et la fiabilité du chatbot dans divers contextes d'utilisation.*

### 5.1. Description des attributs

La dataset « Friends », tiré de la série télévisée Friends, contient des informations sur les dialogues et les scènes de la série.

Elle contient 6 attributs :

- 1- **text** : Contient le dialogue ou la réplique spécifique dite par un personnage ou les indications de scène.
- 2- **speaker** : Indique le personnage qui prononce la réplique. Les personnages principaux incluent Monica Geller, Joey Tribbiani, Chandler Bing, Phoebe Buffay, Ross Geller, et Rachel Green. Les indications de scène sont marquées par "Scene Directions".
- 3- **season** : Spécifie la saison de la série télévisée Friends dans laquelle la réplique apparaît. Les saisons sont numérotées de 1 à 10.
- 4- **episode** : Numéro de l'épisode au sein de la saison correspondante.
- 5- **scene** : Numéro de la scène au sein de l'épisode. Cela aide à contextualiser où la réplique se situe dans le déroulement de l'épisode.
- 6- **utterance** : Numéro de l'énoncé dans la scène, offrant une séquence chronologique des répliques.

## 5.2 Les outils et les bibliothèques utilisées



### 1. Python

Python est le langage de programmation le plus utilisé dans le domaine du Machine Learning, du Big Data et de la Data Science. On l'a utilisé dans ce projet pour le développement de notre code.



### 2- Pandas :

Pandas est une bibliothèque Python puissante dédiée à la manipulation et à l'analyse de données structurées, comme les tableaux et les séries chronologiques. Elle propose des structures de données flexibles et expressives qui facilitent la manipulation, le nettoyage et l'analyse efficace des données.



### 3- Matplotlib :





## Pickle

### 8- Pickle:

pickle est un module de python qui permet de sauvegarder dans un fichier, au format binaire, n'importe quel objet Python.



### 9- Google Colab:

Google Colab est un service cloud de Google qui permet d'écrire, d'exécuter et de partager des notebooks Jupyter directement depuis le navigateur.



## Hugging Face

### 10-Hugging Face :

Hugging Face est une entreprise qui développe et propose des outils open source pour le traitement du langage naturel (NLP) et l'apprentissage automatique, en se spécialisant dans les modèles basés sur des transformateurs comme BERT et GPT.



### 11-Dialog-GPT :

DialogGPT est un modèle de traitement du langage naturel développé par Microsoft, conçu pour générer des conversations interactives et cohérentes en se basant sur le modèle GPT-2 de transformateurs.



### 12- Telegram bot API : The Botfather

L'API BotFather est une interface fournie par Telegram pour créer et gérer des bots, permettant aux utilisateurs de configurer et d'obtenir des tokens d'accès pour leurs bots Telegram.



PyTorch est une bibliothèque open-source de machine learning et de calcul numérique développée par Facebook. Elle offre une flexibilité et une efficacité considérables pour la création de modèles d'apprentissage automatique, en particulier dans les domaines du deep learning et du traitement du langage naturel.

## Chapitre 6 : Analyse exploratoire et Préparation des données

---

*Dans ce chapitre, nous allons explorer en détail la phase exploratoire et la phase de préparation des données. Pendant la phase exploratoire, notre objectif sera d'analyser visuellement notre jeu de données afin d'identifier les relations entre les variables. Dans la phase de préparation, nous nous concentrerons sur le nettoyage et la transformation des données pour les rendre prêtes à être utilisées dans notre application chatbot.*

### 6.1 Importation des bibliothèques nécessaires

- Pour notre projet, nous avons inclus les bibliothèques nécessaires qui seront utilisées à toutes les étapes de développement :

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from plotly.express import bar
import string
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from textblob import TextBlob
from nltk.stem import PorterStemmer, WordNetLemmatizer
from wordcloud import WordCloud
from sklearn.feature_extraction.text import CountVectorizer
from gensim.models import Word2Vec
from sklearn.decomposition import PCA
import random
import pickle
import re
```

Figure 7: Importation des bibliothèques

## 6.2 Chargement du dataset

```
dataset = pd.read_csv("/content/drive/MyDrive/chatbot_friends/dataset.csv")
df = dataset.copy()
df.head()
```

	text	speaker	season	episode	scene	utterance
0	There's nothing to tell! He's just some guy I ...	Monica Geller	1	1	1	1
1	C'mon, you're going out with the guy! There's ...	Joey Tribbiani	1	1	1	2
2	All right Joey, be nice. So does he have a hum...	Chandler Bing	1	1	1	3
3	Wait, does he eat chalk?	Phoebe Buffay	1	1	1	4
4	(They all stare, bemused.)	Scene Directions	1	1	1	5

Figure 8: Chargement du dataset

## 6.3 Analyse Exploratoire des données

### 6.3.1 Analyse de forme

- Afficher des informations sur la taille de notre DataFrame en termes de lignes et de colonnes, puis le types des données



**الكلية متعددة التخصصات الناظور**  
**+0٤٢٧.١٦ +0٨٩٣٨٠٤٩٦ | II.E:Q**  
**Faculté Pluridisciplinaire de Nador**

```
#Nombre de lignes et de colonnes
num_rows = df.shape[0]
num_cols = df.shape[1]
print("Nombre de lignes:", num_rows)
print("Nombre de colonnes:", num_cols)
```

```
Nombre de lignes: 67373
Nombre de colonnes: 6
```

```
#Type de données
df.dtypes
```

```
text      object
speaker   object
season    int64
episode   int64
scene     int64
utterance int64
dtype: object
```

**Figure 9: informations à propos dataset**

- Un résumé statistique des colonnes numériques du DataFrame df (cout, ecart-type, mean...etc)

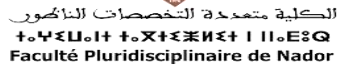
```
#statistics
df.describe()
```

	season	episode	scene	utterance
count	67373.000000	67373.000000	67373.000000	67373.000000
mean	5.438069	12.608241	6.858831	18.079557
std	2.788974	6.962668	4.285814	21.145387
min	1.000000	1.000000	1.000000	0.000000
25%	3.000000	7.000000	3.000000	6.000000
50%	5.000000	13.000000	6.000000	12.000000
75%	8.000000	19.000000	10.000000	22.000000
max	10.000000	25.000000	29.000000	255.000000

### Figure 10: Statistiques

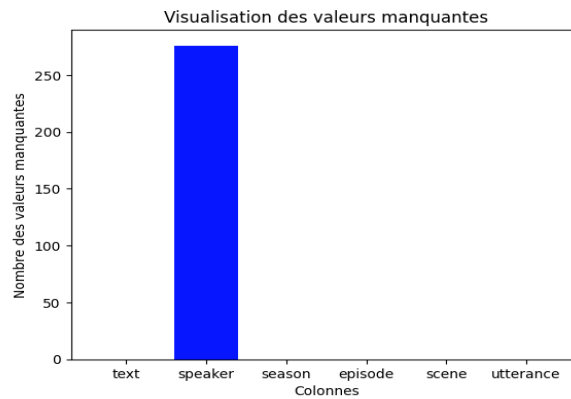
- Identifier et visualiser les colonnes du DataFrame `df` où des valeurs manquantes sont présentes

### 6.3.2 Analyse de fond



```
#Calculution du pourcentage des valeurs manquantes
missing_values = df.isnull().sum()
plt.bar(missing_values.index,missing_values.values, color='Blue')

plt.xlabel("Colonnes")
plt.ylabel("Nombre des valeurs manquantes")
plt.title("Visualisation des valeurs manquantes")
plt.show()
```

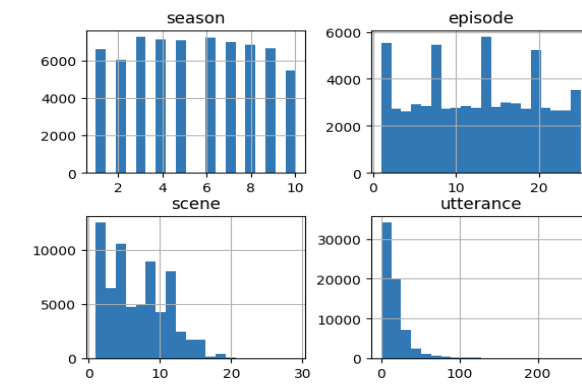


### Figure 11: Visualisation des valeurs manquantes

- Analyse univariée :

- Explorer visuellement la distribution des données dans les colonnes numériques spécifiées du DataFrame `df`, en utilisant des histogrammes pour mieux comprendre la répartition et la variabilité des valeurs dans ces colonnes.

```
#Visualisation des variables
#Pour les donnees numeriques
numeric_cols = ["season","episode","scene","utterance"]
df[numeric_cols].hist(bins=20)
plt.show()
```



**Figure 12: Visualisation de la distribution des données**

- Visualiser la répartition des longueurs de texte dans la colonne "text" du DataFrame df à l'aide d'un diagramme de distribution. Cette visualisation permet d'analyser et de mieux comprendre la variation des longueurs des textes présents dans l'ensemble de données.



الكلية متعددة التخصصات الناجور  
ⵜⴰⵎⴷⵓⵔⵜ ⵜⴰⵎⴷⵓⵔⵜ ⵜⴰⵏⵔⴰⵢⵜ ⵜⴰⵎⴷⵓⵔⵜ  
Faculté Pluridisciplinaire de Nador

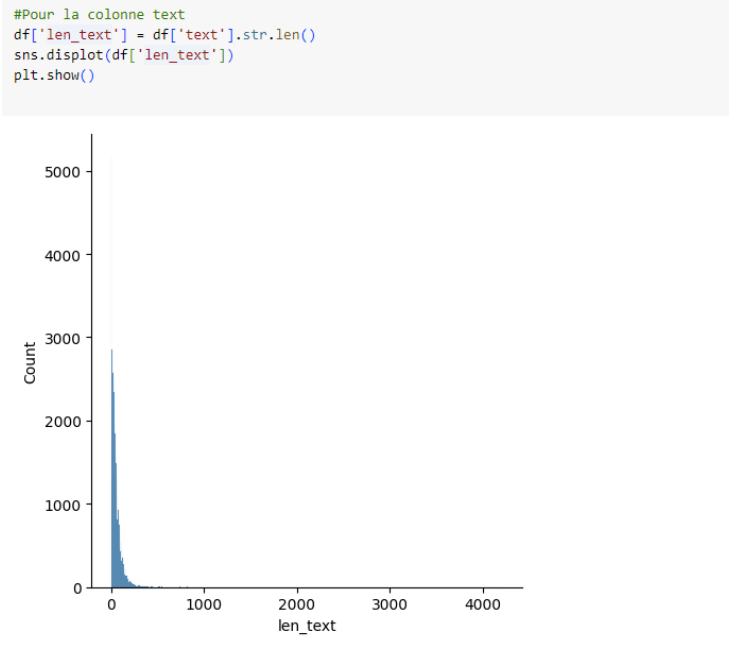


Figure 13: Visualisation de la répartition des longueurs de texte dans la colonne "text"

- Utiliser un diagramme de distribution pour visualiser la variabilité des longueurs des noms des locuteurs dans la colonne "speaker" du DataFrame df.

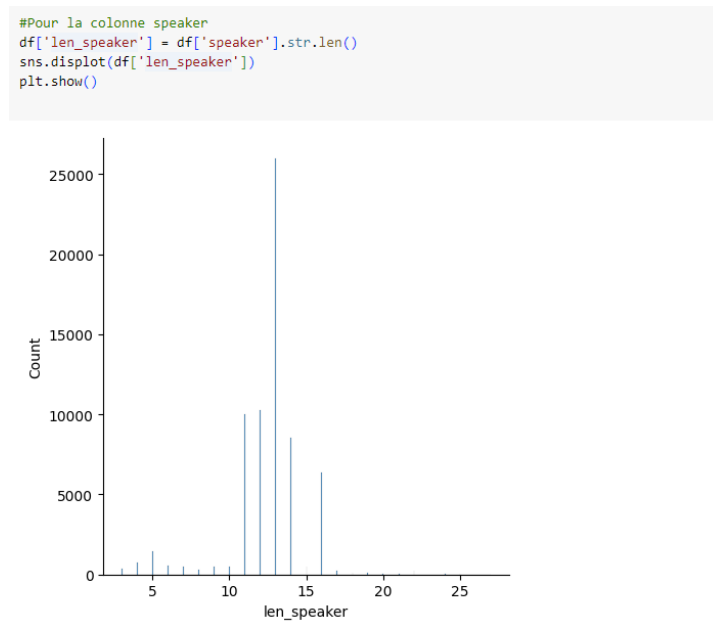


Figure 14: visualisation de la variabilité des longueurs des noms des locuteurs dans la colonne "speaker"

## ■ Analyse Bivarie :







```
#Top 7 speakers dans chaque saison
speakers = df['speaker'].value_counts().to_frame().reset_index().head(6)['speaker'].values
top_speakers = df[df['speaker'].isin(speakers)]
bar(data_frame=top_speakers[['season','speaker']].groupby(by=['season','speaker']).size().reset_index().rename({0:'count'}, axis=1), x='speaker',y='count',color='season')
```

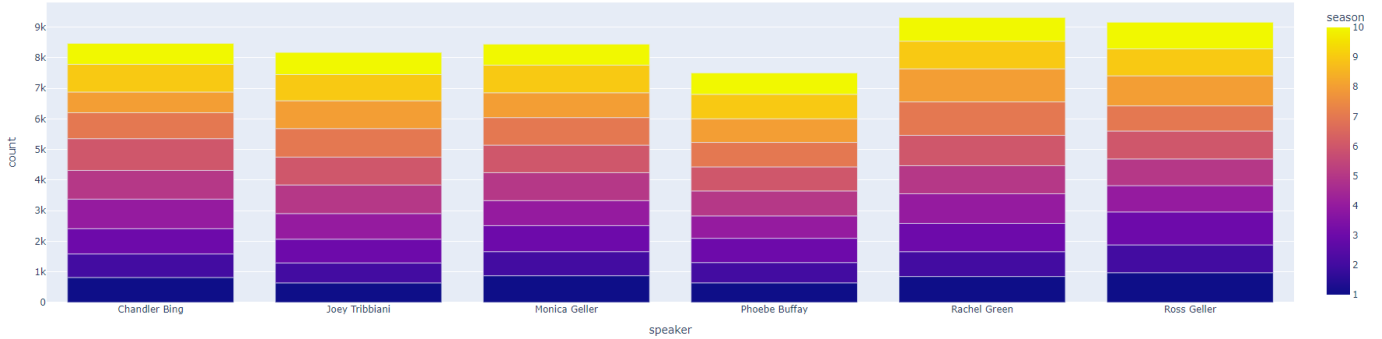


Figure 17: TOP 7 speakers plus fréquents

### ■ Analyse Mutivarie :

- Calculer la matrice de corrélation des colonnes numériques du DataFrame et la visualiser avec des annotations. La visualisation est affichée avec le titre 'Matrice de Correlation'.

```
matrice = df[numeric_cols].corr()
plt.figure(figsize=(25, 15))
sns.heatmap(matrice, annot=True,cmap='YlGnBu')
plt.title('Matrice de Correlation')
plt.show()
```

Figure 18: code Matrice de corrélation

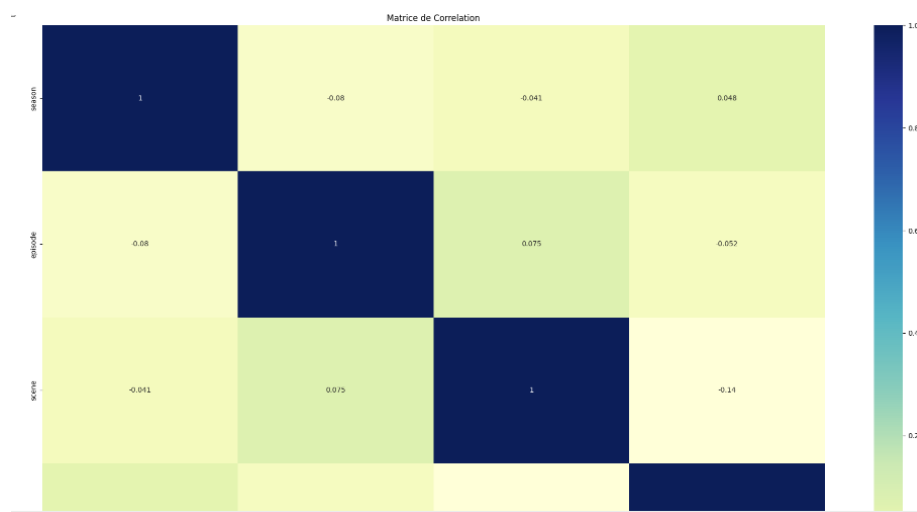


Figure 19: Matrice de corrélation

- Créer un diagramme en boîte (box plot) pour visualiser les valeurs aberrantes des colonnes numériques du DataFrame.

```
#Visualisation des valeurs aberrantes
df.boxplot(numeric_cols)
plt.ylabel("Valeurs")
plt.title("Box plot avec les valeurs aberrantes")
plt.show()
```

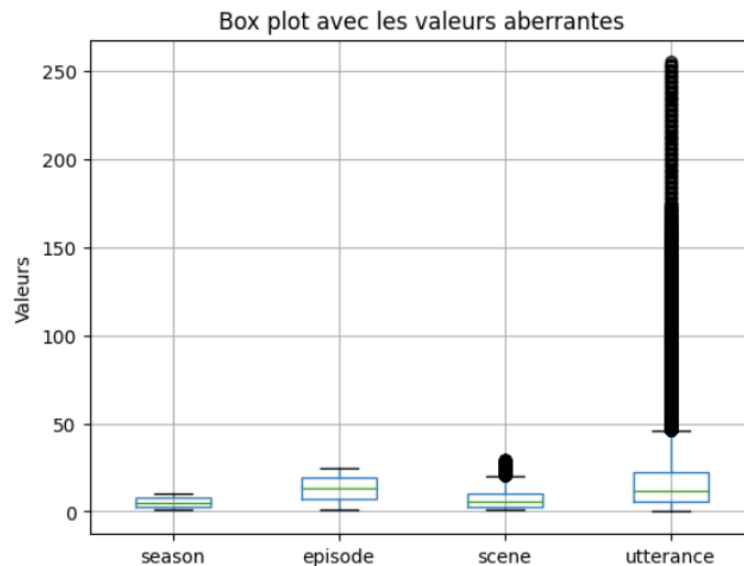


Figure 20: Box plot avec les valeurs aberrantes

## 6.4 Préparation des données

- Nettoyage des données :

- Vérifier combien de valeurs manquantes il y a dans chaque colonne du DataFrame, et les remplacer par zéro.

## Missing values

```
df.isna().sum()
```

```
text          0
speaker      276
season        0
episode       0
scene         0
utterance     0
len_text      0
len_speaker   276
dtype: int64
```

## Filling in missing values

```
df.fillna(0, inplace=True)
```

**Figure 21: Nombre de valeurs manquantes et les remplir par zero**

- Supprimer les indications scéniques et les annotations entre crochets et parenthèses du texte dans la colonne 'text' du DataFrame 'df'. Ensuite, éliminer les étiquettes des haut-parleurs suivies d'un deux-points, et stocker le texte nettoyé dans une nouvelle colonne 'processed\_text'.

### Fonction Prétraitement

```
def pretraitement(text):
    #Supprimer les indications sceniques et les annotations
    text = re.sub(r'\[.*?\]', '', text)
    text = re.sub(r'\(.*?\)', '', text)

    #Supprimer les etiquettes des haut-parleurs
    text = re.sub(r'\b[A-Z][a-z]+\b:', '', text)
    return text
```

```
df['processed_text'] = df['text'].apply(pretraitement)
```

```
df.head(20)
#df.drop(columns=['processed_text'],inplace=True)
```

**Figure 22: Fonctions de prétraitement**

- Combiner tous les textes nettoyés de la colonne 'processed\_text' du DataFrame en une seule chaîne, puis générer et afficher un nuage de mots basé sur cette chaîne à l'aide de WordCloud, sans axes visibles.

[illegible]

35



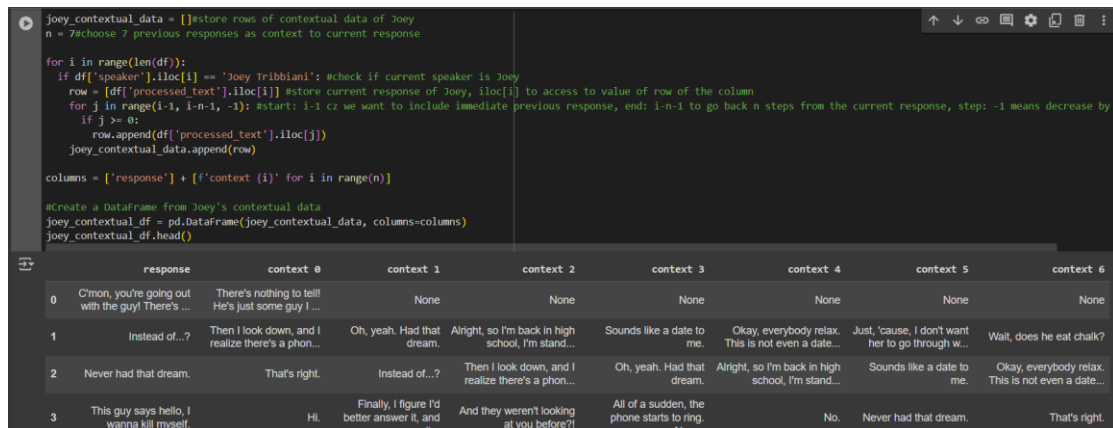


الكلية متعددة التخصصات الناحور  
ⵜⴰⵎⵓⵔⵜ ⵏ ⵜⴰⵎⵓⵔⵜ ⵏ ⵜⴰⵎⵓⵔⵜ ⵏ ⵜⴰⵎⵓⵔⵜ  
Faculté Pluridisciplinaire de Nador

*processus de modélisation, depuis l'obtention des données contextuelles jusqu'à l'entraînement et l'interaction avec le modèle.*

## 7.1 Obtenir les données contextuelles du Joey Tribbiani

- Nous convertirons cet ensemble de données de manière à ce que chaque ligne de réponse contienne n réponses précédentes en tant que contexte. Pour nos besoins, 7 réponses suffiront.

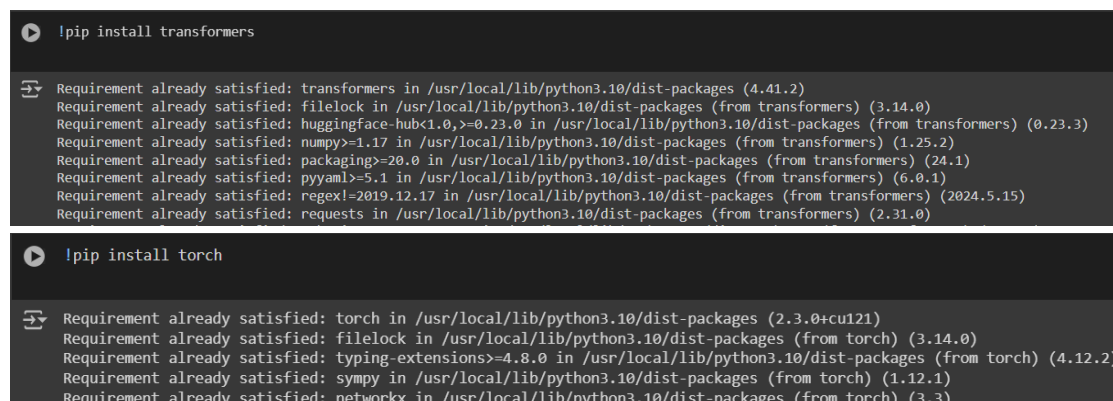


	response	context 0	context 1	context 2	context 3	context 4	context 5	context 6
0	C'mon, you're going out with the guy! There's...	There's nothing to tell! He's just some guy! I...	None	None	None	None	None	None
1	Instead of...?	Then I look down, and I realize there's a phon...	Oh, yeah. Had that dream.	Alright, so I'm back in high school, I'm stand...	Sounds like a date to me.	Okay, everybody relax. This is not even a date...	Just, 'cause, I don't want her to go through w...	Wait, does he eat chalk?
2	Never had that dream.	That's right.	Instead of...?	Then I look down, and I realize there's a phon...	Oh, yeah. Had that dream.	Alright, so I'm back in high school, I'm stand...	Sounds like a date to me.	Okay, everybody relax. This is not even a date...
3	This guy says hello, I wanna kill myself.	Hi.	Finally, I figure I'd better answer it, and it	And they weren't looking at you before?!	All of a sudden, the phone starts to ring. Now	No.	Never had that dream.	That's right.

Figure 25 : Context dataframe

## 7.2 Installation de la bibliothèque transformers

- Nous installons les deux bibliothèques « transformers » de Hugging Face et « torch » (PyTorch) de deep learning



```

!pip install transformers

Requirement already satisfied: transformers in /usr/local/lib/python3.10/dist-packages (4.41.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from transformers) (3.14.0)
Requirement already satisfied: huggingface-hub<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from transformers) (0.23.3)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (1.25.2)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from transformers) (24.1)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from transformers) (6.0.1)
Requirement already satisfied: regex<=2019.12.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (2024.5.15)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from transformers) (2.31.0)

!pip install torch

Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (2.3.0+cu121)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch) (3.14.0)
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/python3.10/dist-packages (from torch) (4.12.2)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch) (1.12.1)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch) (3.3)
    
```

Figure 26 : Installation des bibliothèques transformers et torch



## 7.3 Importation des bibliothèques

```
[ ] from transformers import AutoModelWithLMHead, AutoTokenizer #transformers is a Hugging face's library
import torch
```

Figure 27 : Importation des bibliothèques transformers et torch


## 7.4 Chargement du modèle


- La figure ci-dessous charge le tokenizer et le modèle pré-entraîné DialoGPT-small de Microsoft, nécessaires pour le traitement et la génération de texte.


```
#Loading tokenizer
tokenizer = AutoTokenizer.from_pretrained("microsoft/DialoGPT-small")
#Loading model
model = AutoModelWithLMHead.from_pretrained("microsoft/DialoGPT-small")
```

/usr/local/lib/python3.10/dist-packages/huggingface\_hub/utils/\_token.py:89: UserWarning:

The secret `HF\_TOKEN` does not exist in your Colab secrets.  
To authenticate with the Hugging Face Hub, create a token in your settings tab (<https://huggingface.co/settings/tokens>),  
You will be able to reuse this secret in all of your notebooks.  
Please note that authentication is recommended but still optional to access public models or datasets.


tokenizer\_config.json: 100%  614/614 [00:00<00:00, 12.9kB/s]


vocab.json: 100%  1.04M/1.04M [00:01<00:00, 583kB/s]

merges.txt: 100%  456k/456k [00:00<00:00, 581kB/s]

/usr/local/lib/python3.10/dist-packages/transformers/models/auto/modeling\_auto.py:1712: FutureWarning:

The class `AutoModelWithLMHead` is deprecated and will be removed in a future version. Please use `AutoModelForCausalLM`

config.json: 100%  641/641 [00:00<00:00, 44.0kB/s]

model.safetensors: 100%  351M/351M [00:02<00:00, 148MB/s]


generation\_config.json: 100%  124/124 [00:00<00:00, 9.32kB/s]

Figure 28 : Chargement du modèle

## 7.5 Division du dataset en parties d'entraînement et test

- Dans cette étape on fait diviser le jeu de données contextuel de Joey en ensembles d'entraînement 80% et de validation 20%, puis nous affichons leur taille respective.

```
[ ] from sklearn.model_selection import train_test_split
train_set, validation_set = train_test_split(joey_contextual_df, test_size=0.2)
print("Length of trainig set:", len(train_set))
print("Length of validation set:",len(validation_set))
```

Length of trainig set: 6536  
Length of validation set: 1635

Figure 29 : Division du dataset en Train/Test sets



## 7.8 Gestion des checkpoints et du cache

- Le code ci-dessous gère le caching et la gestion des checkpoints dans le cadre de l'entraînement de modèles.

```

1  import logging
2  import os
3  import pickle #save and load preprocessed data
4  from transformers import PreTrainedTokenizer, MODEL_WITH_LM_HEAD_MAPPING, WEIGHTS_NAME, AutoConfig
5  from torch.nn.utils.rnn import pad_sequence
6  from torch.utils.data import DataLoader, Dataset, RandomSampler, SequentialSampler
7  from torch.utils.data.distributed import DistributedSampler
8  from pathlib import Path
9  from typing import Dict, List, Tuple
10
11  MODEL_CONFIG_CLASSES = list(MODEL_WITH_LM_HEAD_MAPPING.keys())
12  MODEL_TYPES = tuple(conf.model_type for conf in MODEL_CONFIG_CLASSES)
13
14  logger = logging.getLogger(__name__) #logging used for tracking what is happening in training and evaluation process
15  #convert rows of dataframe into tokens and embedding it
16  def construct_conv(row, tokenizer, eos = True):
17      flatten = lambda l: [item for sublist in l for item in sublist] #long array of tokens
18      conv = list(reversed([tokenizer.encode(x) + [tokenizer.eos_token_id] for x in row]))
19      conv = flatten(conv)
20      return conv
21
22  #Convert dataframe into Pytorch dataset
23  class ConversationDataset(Dataset):
24      def __init__(self, tokenizer, args, df, block_size=512):
25
26          block_size = min(block_size, tokenizer.model_max_length)
27          #block_size = block_size - (tokenizer.model_max_length - tokenizer.max_len_single_sentence)
28
29          directory = args.cache_dir
30          cached_features_file = os.path.join(
31              directory, args.model_type + "_cached_lm_" + str(block_size)
32          )
33
34          if os.path.exists(cached_features_file) and not args.overwrite_cache:
35              logger.info("Loading features from cached file %s", cached_features_file)
36              with open(cached_features_file, "rb") as handle:
37                  self.examples = pickle.load(handle)
38          else:
39              logger.info("Creating features from dataset file at %s", directory)
40
41              self.examples = []
42              for _, row in df.iterrows():
43                  conv = construct_conv(row, tokenizer)
44                  self.examples.append(conv)
45
46              logger.info("Saving features into cached file %s", cached_features_file)
47              with open(cached_features_file, "wb") as handle:
48                  pickle.dump(self.examples, handle, protocol=pickle.HIGHEST_PROTOCOL)
49
50      def __len__(self):
51          return len(self.examples)
52
53      def __getitem__(self, item):
54          return torch.tensor(self.examples[item], dtype=torch.long)
55
    
```

Figure 32 : Gestion des checkpoints

## 7.9 Configuration du modèle

```
1 # Args to allow for easy conversion of python script to notebook
2 class Args():
3     def __init__(self):
4         self.output_dir = 'FriendsGPT-small'
5         self.model_type = 'gpt2'
6         self.model_name_or_path = 'microsoft/DialoGPT-small'
7         self.config_name = 'microsoft/DialoGPT-small'
8         self.tokenizer_name = 'microsoft/DialoGPT-small'
9         self.cache_dir = 'cached'
10        self.block_size = 512
11        self.do_train = True
12        self.do_eval = True
13        self.evaluate_during_training = False
14        self.per_gpu_train_batch_size = 4
15        self.per_gpu_eval_batch_size = 4
16        self.gradient_accumulation_steps = 1
17        self.learning_rate = 5e-5
18        self.weight_decay = 0.0
19        self.adam_epsilon = 1e-8
20        self.max_grad_norm = 1.0
21        self.num_train_epochs = 3
22        self.max_steps = -1
23        self.warmup_steps = 0
24        self.logging_steps = 1000
25        self.save_steps = 3500
26        self.save_total_limit = None
27        self.eval_all_checkpoints = False
28        self.no_cuda = False
29        self.overwrite_output_dir = True
30        self.overwrite_cache = True
31        self.should_continue = False
32        self.seed = 42
33        self.local_rank = -1
34        self.fp16 = False
35        self.fp16_opt_level = '01'
36
37    args = Args()
```

Figure 33 : Configuration du modèle

## 7.10 Entraînement du modèle

- Nous représentons ici un aperçu du code d'entraînement du model ainsi que la fonction main, nous avons implémenté les fonctions nécessaires pour l'entraînement et l'évaluation d'un modèle de langage basé sur Transformers

```
[ ] from transformers import PreTrainedModel, AdamW, get_linear_schedule_with_warmup
from tqdm.notebook import tqdm, trange
try:
    from torch.utils.tensorboard import SummaryWriter
except ImportError:
    from tensorboardX import SummaryWriter

def train(args, train_dataset, model: PreTrainedModel, tokenizer: PreTrainedTokenizer) -> Tuple[int, float]:
    """ Train the model """
    if args.local_rank in [-1, 0]:
        tb_writer = SummaryWriter()

    args.train_batch_size = args.per_gpu_train_batch_size * max(1, args.n_gpu)

    def collate(examples: List[torch.Tensor]):
        if tokenizer.pad_token is None:
            return pad_sequence(examples, batch_first=True)
        return pad_sequence(examples, batch_first=True, padding_value=tokenizer.pad_token_id)

    train_sampler = RandomSampler(train_dataset) if args.local_rank == -1 else DistributedSampler(train_dataset)
    train_dataloader = DataLoader(
        train_dataset, sampler=train_sampler, batch_size=args.train_batch_size, collate_fn=collate, drop_last = True
    )

# Main runner
def main(df_trn, df_val):
    args = Args()

    if args.should_continue:
        sorted_checkpoints = _sorted_checkpoints(args)
        if len(sorted_checkpoints) == 0:
            raise ValueError("Used --should_continue but no checkpoint was found in --output_dir.")
        else:
            args.model_name_or_path = sorted_checkpoints[-1]

    if (
        os.path.exists(args.output_dir)
        and os.listdir(args.output_dir)
        and args.do_train
        and not args.overwrite_output_dir
        and not args.should_continue
    ):
        raise ValueError(
            "Output directory ({} already exists and is not empty. Use --overwrite_output_dir to overcome.".format(
                args.output_dir
            )
        )

    # Setup CUDA, GPU & distributed training
    device = torch.device("cuda")
    args.n_gpu = torch.cuda.device_count()
    args.device = device
```

Figure 43 : Entraînement du modèle

## 7.11 Evaluation du modèle

- La perplexité est une mesure utilisée pour évaluer la qualité des modèles de langage. Elle quantifie à quel point un modèle est perplexe face à un ensemble de données. Plus précisément, elle mesure à quel point un modèle est incertain lorsqu'il prédit la probabilité d'une séquence de mots.

Après l'entraînement du modèle nous avons pu avoir un résultat de perplexité a environ 9.5458 cela signifie que, en moyenne, le modèle a une performance raisonnablement bonne sur ce jeu de données spécifique, avec une prédiction relativement précise des séquences de mots.



الكلية متعددة التخصصات النador  
+0.480.11 +0.74361111 | II.E:Q  
Faculté Pluridisciplinaire de Nador

```
main(train_set, validation_set)

WARNING: main :Process rank: -1, device: cuda, n_gpu: 1, distributed training: False, 16-bits training: False
/usr/local/lib/python3.10/dist-packages/huggingface_hub/file_download.py:1132: FutureWarning:

`resume_download` is deprecated and will be removed in version 1.0.0. Downloads always resume when possible. If you want
to ensure that this behavior works, set `resume_from_checkpoint=True` in the constructor. You also have to set the
variable `HUGGINGFACE_HUB_CACHE` to point to where you want to store temporary files during downloading. Please check
the documentation in the git repository at https://github.com/huggingface/huggingface_hub/blob/main/docs/source/en/cheat-sheet.md
for more details.
/usr/local/lib/python3.10/dist-packages/transformers/models/auto/modeling_auto.py:1712: FutureWarning:

The class `AutoModelWithLMHead` is deprecated and will be removed in a future version. Please use `AutoModelForCausalLM`.
/usr/local/lib/python3.10/dist-packages/transformers/optimization.py:588: FutureWarning:

This implementation of AdamW is deprecated and will be removed in a future version. Use the PyTorch implementation torch.optim.AdamW instead, which implements the same algorithm without bugs.

Epoch: 100% ██████████ 3/3 [04:20<00:00, 87.49s/it]
Iteration: 100% ██████████ 375/375 [01:23<00:00, 4.61it/s]
Iteration: 100% ██████████ 375/375 [01:28<00:00, 3.43it/s]
Iteration: 100% ██████████ 375/375 [01:28<00:00, 4.71it/s]
/usr/local/lib/python3.10/dist-packages/torch/optim/lr_scheduler.py:271: UserWarning:

To get the last learning rate computed by the scheduler, please use `get_last_lr()`.

/usr/local/lib/python3.10/dist-packages/transformers/models/auto/modeling_auto.py:1712: FutureWarning:

The class `AutoModelWithLMHead` is deprecated and will be removed in a future version. Please use `AutoModelForCausalLM`.
```

### Figure 34 : Evaluation du modèle

## 7.12 Discuter avec Joey Tribbiani

Le modèle est prêt, il est donc temps de discuter avec notre caractère Joey Tribbiani en lui pose quelques questions basiques.

```
tokenizer = AutoTokenizer.from_pretrained('microsoft/DialoGPT-small')
model = AutoModelWithLMHead.from_pretrained('FriendsGPT-small')

# Let's chat for 5 lines
for step in range(20):
    # encode the new user input, add the eos_token and return a tensor in Pytorch
    new_user_input_ids = tokenizer.encode(input(">> User:") + tokenizer.eos_token, return_tensors='pt')
    # print(new_user_input_ids)

    # append the new user input tokens to the chat history
    bot_input_ids = torch.cat([chat_history_ids, new_user_input_ids], dim=-1) if step > 0 else new_user_input_ids

    # generated a response while limiting the total chat history to 1000 tokens,
    chat_history_ids = model.generate(bot_input_ids, max_length=1000, pad_token_id=tokenizer.eos_token_id)

    # pretty print last output tokens from bot
    print("Joey: {}".format(tokenizer.decode(chat_history_ids[:, bot_input_ids.shape[-1]:][0], skip_special_tokens=True)))

>> User:hi joey
Joey: Hey.
>> User:what is your name?
Joey: Joey.
>> User:what is your favorite food?
Joey: I don't know.
>> User:do you love pizza?
Joey: No.
>> User:do you love pizza?
Joey: No.
```

### Figure 35 : Discussion avec Joey Tribbiani

## Chapitre 8 : Déploiement

---

*Ce chapitre se concentre sur le déploiement du chatbot Friends sur Telegram, explorant l'installation des bibliothèques nécessaires et la mise en œuvre du chatbot pour une intégration efficace sur la plateforme Telegram.*

## 8.1 Installation de la bibliothèque du Telegram

- Dans cette étape nous installons la bibliothèque Python **python-telegram-bot** pour interagir avec l'API Telegram.

```
[ ] !pip install python-telegram-bot==13.7

Collecting python-telegram-bot==13.7
  Downloading python_telegram_bot-13.7-py3-none-any.whl (490 kB)
    490.1/490.1 kB 7.6 MB/s eta 0:00:00
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-packages (from python-telegram-bot==13.7) (2024.6.2)
Requirement already satisfied: tornado>=6.1 in /usr/local/lib/python3.10/dist-packages (from python-telegram-bot==13.7) (6.3.3)
Collecting APScheduler==3.6.3 (from python-telegram-bot==13.7)
  Downloading APScheduler-3.6.3-py2.py3-none-any.whl (58 kB)
    58.9/58.9 kB 9.9 MB/s eta 0:00:00
Requirement already satisfied: pytz>=2018.6 in /usr/local/lib/python3.10/dist-packages (from python-telegram-bot==13.7) (2023.4)
Collecting cachetools==4.2.2 (from python-telegram-bot==13.7)
  Downloading cachetools-4.2.2-py3-none-any.whl (11 kB)
Requirement already satisfied: setuptools>=0.7 in /usr/local/lib/python3.10/dist-packages (from APScheduler==3.6.3->python-telegram-bot==13.7) (67.7.2)
Requirement already satisfied: six>=1.4.0 in /usr/local/lib/python3.10/dist-packages (from APScheduler==3.6.3->python-telegram-bot==13.7) (1.16.0)
Requirement already satisfied: tzlocal>=1.2 in /usr/local/lib/python3.10/dist-packages (from APScheduler==3.6.3->python-telegram-bot==13.7) (5.2)
Installing collected packages: cachetools, APScheduler, python-telegram-bot
  Attempting uninstall: cachetools
    Found existing installation: cachetools 5.3.3
    Uninstalling cachetools-5.3.3:
      Successfully uninstalled cachetools-5.3.3
  Successfully installed APScheduler-3.6.3 cachetools-4.2.2 python-telegram-bot-13.7
```

Figure 36 : Installation de la bibliothèque du déploiement

## 8.2 Création du Chatbot Telegram

- Nous créons ici le chatbot Telegram en utilisant le modèle de langage **FriendsGPT-small** Fine-tuned.

```
from telegram import Update, ForceReply
from telegram.ext import Updater, CommandHandler, MessageHandler, Filters, CallbackContext
from transformers import AutoModelForCausalLM, AutoTokenizer
import torch

# Load the fine-tuned model and tokenizer
tokenizer = AutoTokenizer.from_pretrained('/content/drive/MyDrive/chatbot_friends/FriendsGPT-small')
model = AutoModelForCausalLM.from_pretrained('/content/drive/MyDrive/chatbot_friends/FriendsGPT-small')

# Define the start command handler
def start(update: Update, _: CallbackContext) -> None:
    user = update.effective_user
    update.message.reply_markdown_v2(
        fr'Hi {user.mention_markdown_v2()}! I am your chatbot.',
        reply_markup=ForceReply(selective=True),
    )

# Define the chat handler
def chat(update: Update, _: CallbackContext) -> None:
    user_input = update.message.text
    new_user_input_ids = tokenizer.encode(user_input + tokenizer.eos_token, return_tensors='pt')
    bot_input_ids = torch.cat([chat_history_ids, new_user_input_ids], dim=-1) if 'chat_history_ids' in globals() else new_user_input_ids
    chat_history_ids = model.generate(bot_input_ids, max_length=1024, pad_token_id=tokenizer.eos_token_id)
    bot_response = tokenizer.decode(chat_history_ids[:, bot_input_ids.shape[-1]:][0], skip_special_tokens=True)
    update.message.reply_text(bot_response)

def main() -> None:
    # Initialize the bot
    updater = Updater("7097593647:AA6Sd23QutwYUatMQ5zpw1cVuzhSgwM1Q")
```

Figure 37 : Création du chatbot Telegram





## Conclusion et Perspectives

Le projet de chatbot basé sur les dialogues de la série "Friends" a illustré avec succès comment les technologies de traitement du langage naturel et d'apprentissage automatique peuvent recréer des interactions authentiques et divertissantes. En imitant le style et l'humour des personnages emblématiques de la série, ce chatbot offre une expérience immersive et nostalgique aux fans. Le développement a permis de surmonter divers défis techniques, aboutissant à un modèle capable de fournir des réponses contextuellement appropriées.

Pour l'avenir, plusieurs axes d'amélioration et d'extension peuvent être explorés :

1. **Amélioration de la Précision** : Optimiser les modèles NLP pour améliorer la pertinence et la cohérence des réponses.
2. **Enrichissement du Dataset** : Ajouter davantage de dialogues et de contextes de la série pour couvrir une variété plus large de situations.
3. **Multilinguisme** : Adapter le chatbot pour qu'il puisse converser en plusieurs langues, rendant l'application accessible à un public mondial.
4. **Personnalisation** : Permettre aux utilisateurs de choisir ou de personnaliser leur expérience en interagissant avec différents personnages ou en ajustant le ton et le style des réponses.
5. **Analyse Sentimentale** : Intégrer des analyses de sentiment pour rendre les interactions plus émotionnelles et empathiques, améliorant ainsi l'expérience utilisateur.

Ces perspectives ouvrent des avenues prometteuses pour améliorer et étendre les capacités de ce chatbot innovant, augmentant ainsi son impact et son attrait.