Université Abdelmalek essaadi
Ecole Nationale des Sciences Appliquées d'Al Hoceima

Rapport de stage d'initiation

Sous le thème:

Génération automatique des commentaires pour des codes Java

Réalisé par :

- **KHAYYI Hanae**
- **RACHDAOUI Rachida**

Encadré par :

- **Pr. Mohamed CHERRADI**

Année universitaire : 2023/2024

➔Table des matières:

Partie théorique:

I. Introduction:

L'intelligence artificielle (IA) d'aujourd'hui a largement dépassé l'informatique quantique des données en étude qualitative et prédictive et sélective des données. Cela est dû au fait que d'énormes ressources informatiques sont facilement accessibles à l'homme du commun. Les développeurs en profitent maintenant pour créer de nouveaux modèles d'apprentissage automatique et pour réaliser les modèles existants pour de meilleures performances et de meilleurs résultats. La disponibilité facile du calcul de haute performance a entraîné une augmentation soudaine dans la demande des professionnels de l'informatique des compétences en apprentissage automatique et intelligence artificielle.

Les domaines de la science des données, de l'apprentissage automatique dite « Machine Learning » et de l'intelligence artificielle se chevauchent beaucoup, mais ils ne sont pas interchangeables et sont pratiquement liés dans plusieurs domaines d'étude des données diverses.
Les domaines se chevauchent beaucoup et il y a suffisamment d'exemples des domaines et des cas d'études dans la plupart des professionnels de ces domaines ont une compréhension intuitive de la façon ; dont un travail particulier pourrait être classé en science des données, en apprentissage automatique (ML) ou en intelligence artificielle (IA), même s'il est difficile à mettre en mots.

On distingue des définitions simplifiées pour éclairer les différents champs sujets de notre étude :

- ➢ *La **science des données** produit des idées*
- ➢ *L'**apprentissage automatique** produit des prédictions*

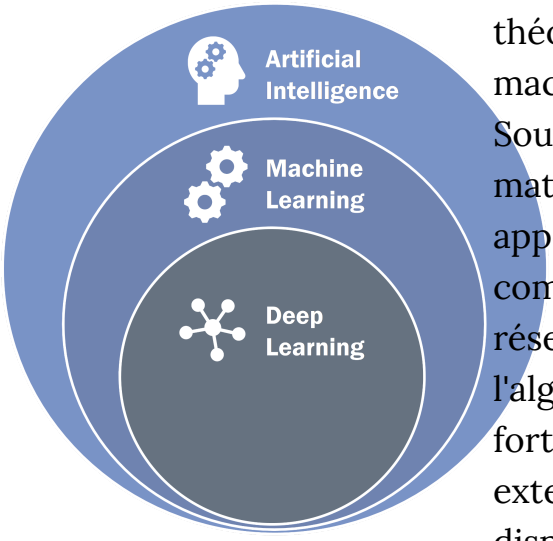
➢ *L'**intelligence artificielle** produit des actions*
➢ *Le **deep learning** produit des représentations complexes*

Lorsqu'on étudie la science des données (Data Science), on confronte souvent des questions du type :

➔ **"Quelle est la définition de l'apprentissage automatique ?"**
➔ **"Quelle est la différence entre l'Intelligence Artificielle et l'apprentissage automatique ?"**
➔ **"Que signifie l'intelligence artificielle ?"**
➔ **"Que signifie Deep Learning ?"**

II. Définitions :

A. Intelligence artificielle:



L'intelligence artificielle (IA) est un ensemble de théories et de techniques visant à réaliser des machines capables de simuler l'intelligence humaine. Souvent classée dans le domaine des mathématiques et des sciences cognitives, l'IA fait appel à des disciplines telles que la neurobiologie computationnelle (qui a notamment inspiré les réseaux neuronaux artificiels), les statistiques, ou l'algèbre linéaire. Elle vise à résoudre des problèmes à forte complexité logique ou algorithmique. Par extension, dans le langage courant, l'IA inclut les dispositifs imitant ou remplaçant l'homme dans certaines mises en œuvre de ses fonctions cognitives. Les applications de l'IA incluent notamment les moteurs de recherche, les systèmes de recommandation, la compréhension automatique des langues, les voitures autonomes, les chatbots, les outils de génération d'images, les outils de prise de décision automatisée et les programmes compétitifs dans des jeux de stratégie.

B. Machine Learning:

L'apprentissage automatique est un sous-domaine de l'informatique qui a évolué à partir de l'étude de la reconnaissance des formes et de la théorie de l'apprentissage informatique dans l'intelligence artificielle. L'apprentissage automatique explore la construction et l'étude d'algorithmes capables d'apprendre et de faire des prédictions sur les données. De tels algorithmes fonctionnent en construisant un modèle à partir d'exemples d'entrées afin de faire des prédictions ou des décisions basées sur des données, plutôt que de suivre des instructions de programme strictement statiques ou dynamiques.

L'apprentissage automatique est étroitement lié aux statistiques informatiques et les chevauchés souvent ; une discipline également spécialisée dans l'analyse, la prévision et la prédiction. Il a des liens étroits avec l'optimisation mathématique, qui fournit des méthodes, une théorie et des domaines d'application sur les modèles de la Machine Learning. L'apprentissage automatique est utilisé dans une gamme de tâches informatiques où la conception et la programmation d'algorithmes explicites sont irréalisables.

Les techniques fondamentales en machine learning pour résoudre une variété de problèmes, de la classification à la prédiction et à la découverte de motifs dans les données sont :

1. **Apprentissage supervisé :**
   C'est une approche en intelligence artificielle et en machine learning où l'algorithme est entraîné sur un ensemble de données étiquetées. Concrètement, cela signifie que pour chaque exemple de données dans l'ensemble d'entraînement, on connaît à l'avance la sortie désirée, ou l'étiquette

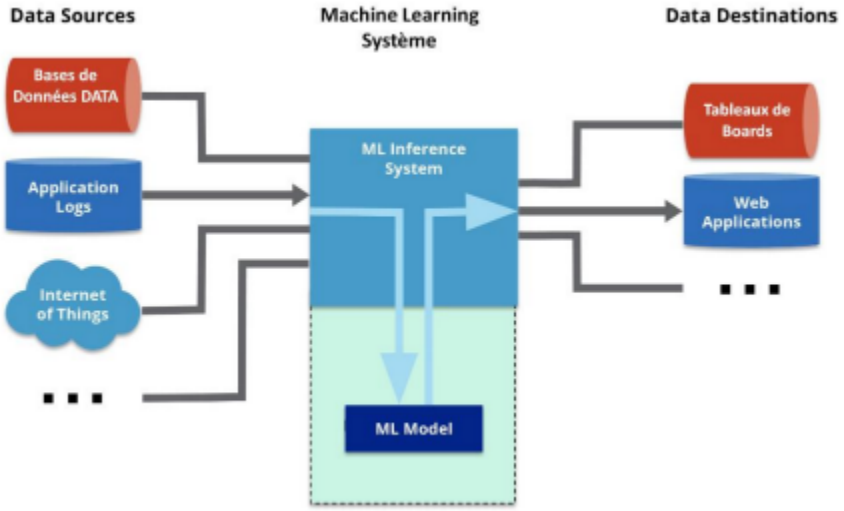
- ***Classification*** : C'est un type d'apprentissage supervisé où l'algorithme apprend à classifier les données en fonction d'exemples étiquetés, par exemple, prédire si un e-mail est un spam ou non.
- ***Régression*** : Également un type d'apprentissage supervisé, où l'algorithme apprend à prédire des valeurs numériques en trouvant une relation entre les variables d'entrée et de sortie, comme prédire le prix d'une maison.

2. **Apprentissage non supervisé :**
   C'est une méthode d'apprentissage automatique où l'algorithme est confronté à des données non étiquetées et doit trouver des structures ou des modèles significatifs par lui-même.

- **Clustering** : C'est un type d'apprentissage non supervisé où l'algorithme regroupe les données similaires en clusters sans utiliser d'étiquettes préexistantes. Par exemple, regrouper des clients en segments basés sur leurs comportements d'achat.

- **Association** : Un autre type d'apprentissage non supervisé où l'algorithme identifie des relations fréquentes entre des variables dans de grandes bases de données transactionnelles. Par exemple, identifier des combinaisons de produits souvent achetés ensemble dans un supermarché.

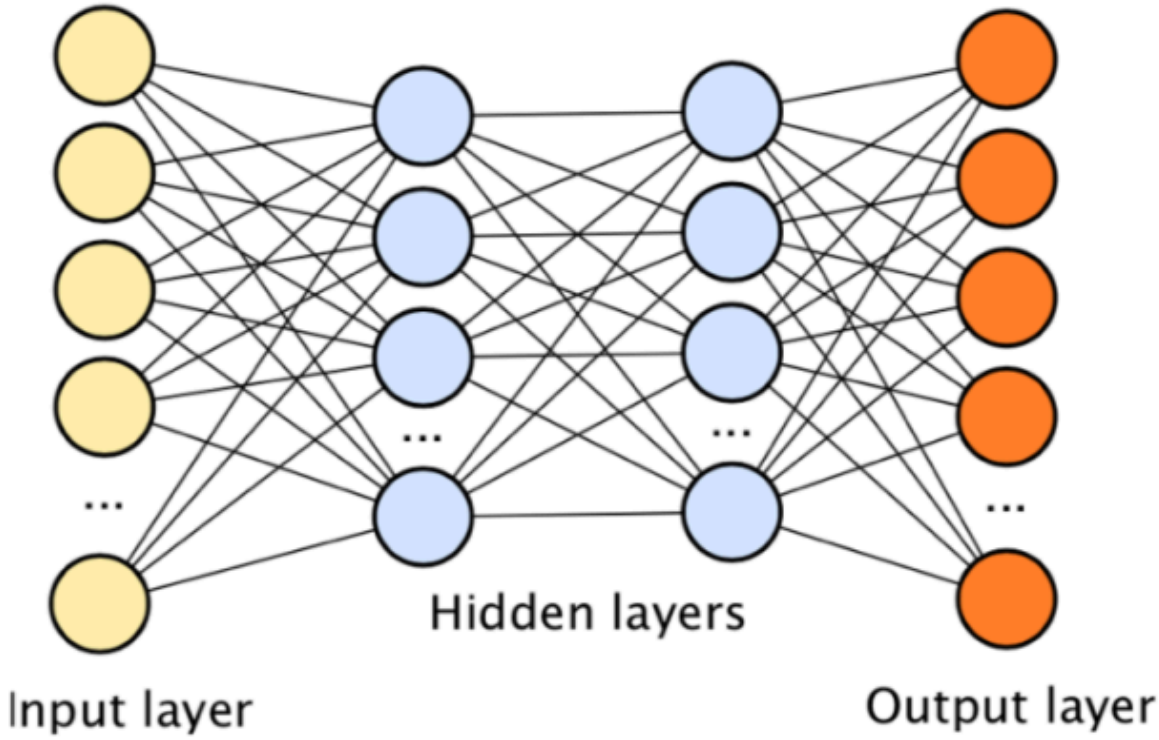
➔ **Architecture Usuelle de Machine Learning**



D. Deep Learning:

C'est une technique de machine learning reposant sur le modèle des réseaux neurones artificiels : des dizaines voire des centaines de couches de neurones sont empilées pour apporter une plus grande complexité à l'établissement des règles.

L'idée de réseaux de neurones artificiels est dérivée des réseaux de neurones du cerveau humain. Le cerveau humain est vraiment complexe. Étudiant attentivement le cerveau, les scientifiques et les ingénieurs ont proposé une architecture qui pourrait s'adapter à notre monde numérique d'ordinateurs binaires. Une telle architecture typique est illustrée dans le diagramme ci-dessous :



Il y a une couche d'entrée qui a de nombreux capteurs pour collecter des données du monde extérieur. Sur le côté droit, nous avons une couche de sortie qui nous donne le résultat prédit par le réseau. Entre ces deux, plusieurs couches sont cachées. Chaque couche supplémentaire ajoute une complexité supplémentaire à la formation du réseau, mais fournira de meilleurs résultats dans la plupart des situations. Il existe plusieurs types d'architectures conçues dont nous allons discuter par la suite.

III. Introduction aux LLMs et NLP:

A. Qu'est-ce qu'un grand modèle de langage (LLM) ?

Les grands modèles de langage, ou **LLMs** (Large Language Models), sont des modèles de réseaux de neurones conçus pour comprendre et générer du langage humain. Ils sont capables de traiter de vastes quantités de données textuelles pour effectuer diverses tâches liées au langage naturel. Ces modèles sont généralement basés sur des architectures de réseaux de neurones profondes et sont pré-entraînés sur des corpus de texte massifs avant d'être affinés pour des tâches spécifiques.

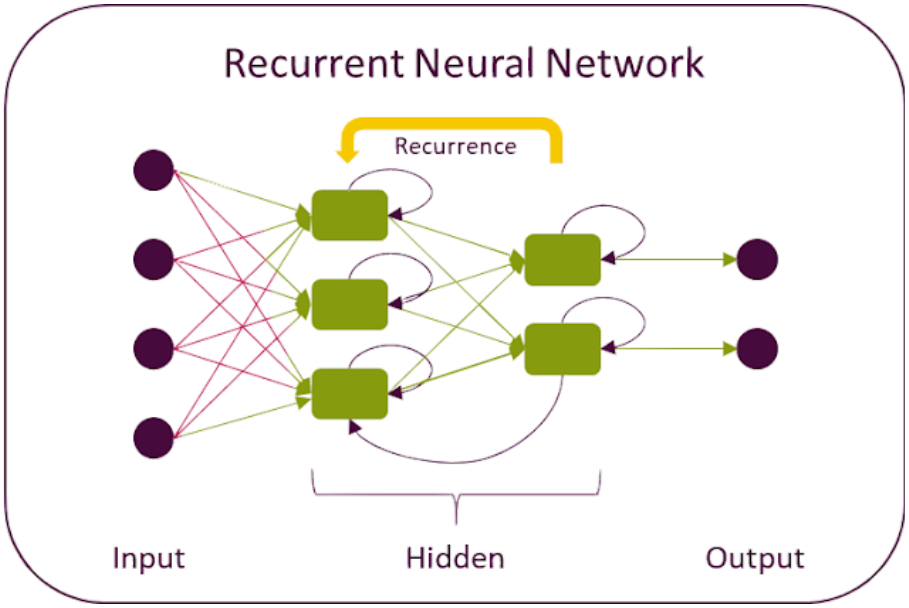
B. Définition du traitement du langage naturel (NLP):

Le traitement du langage naturel (**NLP**) est une sous-discipline de l'intelligence artificielle (**IA**) qui se concentre sur l'interaction entre les ordinateurs et le langage humain. Le NLP implique l'utilisation de techniques et d'algorithmes pour permettre aux machines de comprendre, interpréter et générer du langage humain de manière utile. Les applications du NLP incluent la traduction automatique, l'analyse de sentiment, la reconnaissance vocale, et bien plus encore.

C. Architecture des Réseaux de Neurones:

Les réseaux de neurones sont des modèles mathématiques inspirés par le fonctionnement du cerveau humain, composés de neurones artificiels organisés en couches. Voici une exploration des principales architectures utilisées dans les grands modèles de langage (LLMs) et d'autres applications de traitement du langage naturel (NLP) :

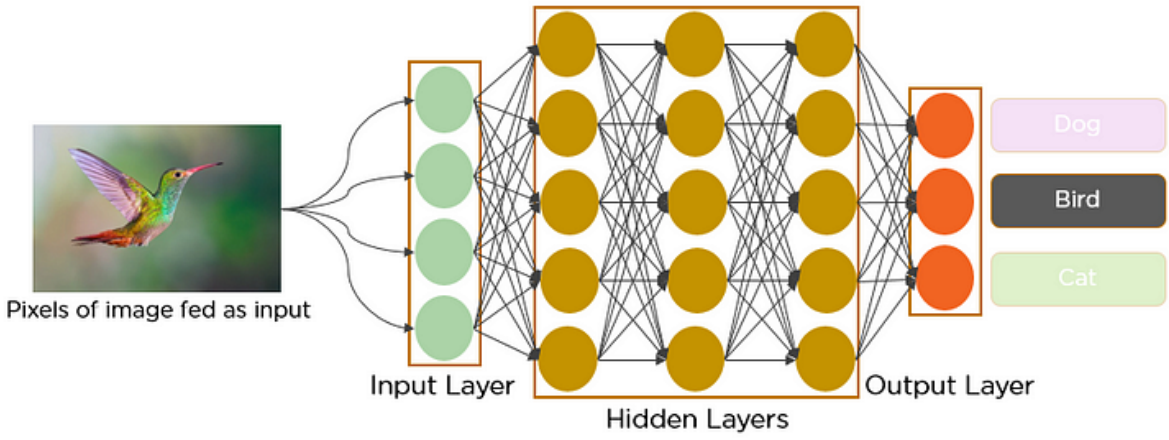
- **Réseaux Neuronaux Récurrents (RNN):**



Les RNN sont conçus pour traiter des données séquentielles en prenant en compte les relations temporelles entre les éléments d'une séquence.

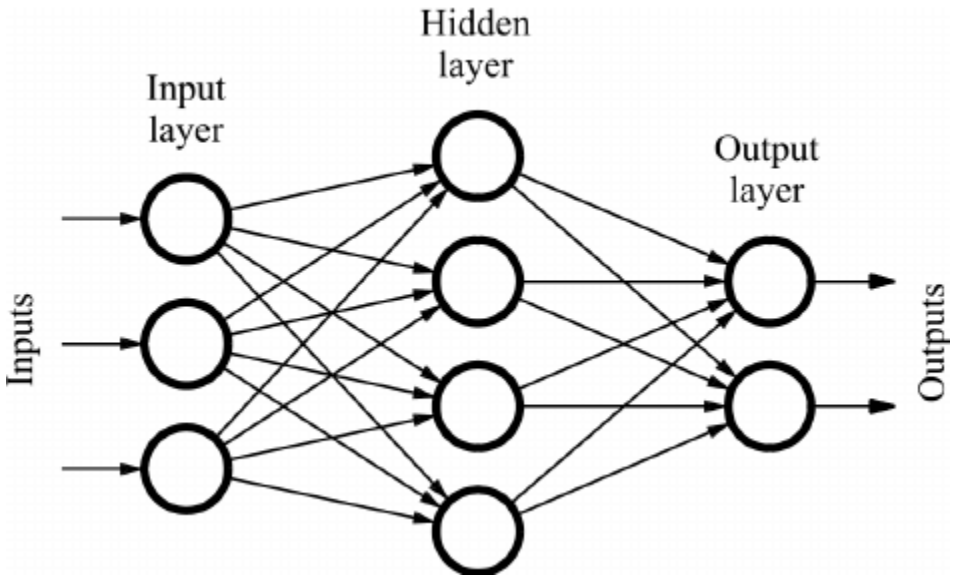
Chaque neurone dans un RNN reçoit une entrée de la couche précédente ainsi qu'une connexion récurrente provenant de sa propre sortie précédente, ce qui lui permet de conserver une mémoire à court terme

- **Réseaux de Neurones Convolutionnels (CNN):**



Les réseaux de neurones convolutionnels (CNN) sont une classe de modèles de réseaux de neurones spécialement conçus pour traiter des données disposées en grilles, comme les images.

❖ **Réseaux de Neurones feed-forward :**



Un réseau neuronal de type feed-forward, également connu sous le nom de perceptron multicouche (MLP pour Multi-Layer Perceptron), est une architecture de réseau de neurones artificiels dans laquelle l'information circule dans une seule direction, de l'entrée vers la sortie, sans cycles ni boucles rétroactives

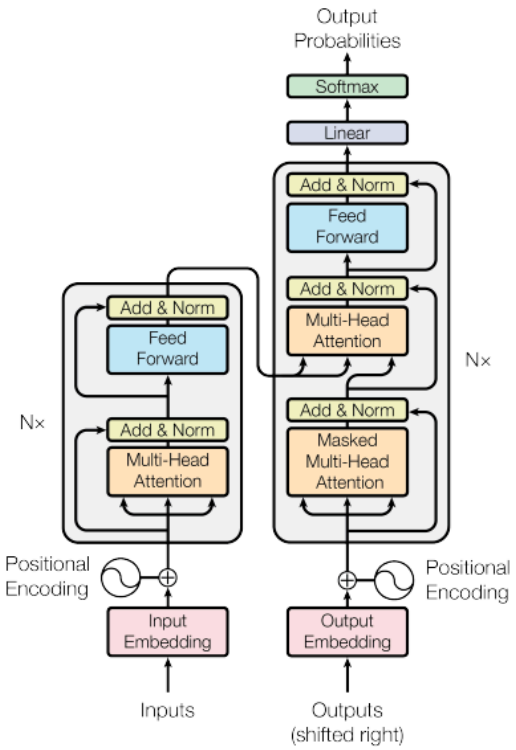
❖ **Transformateurs:**
Les transformateurs sont une architecture récente qui a révolutionné le domaine du NLP depuis leur introduction par le modèle "Attention is All You Need".

Ils se concentrent sur l'attention entre les différentes parties d'une séquence, permettant de capturer efficacement les dépendances à long terme.
Structure Principale :

- Les transformateurs sont composés de blocs d'encodeurs et de décodeurs.

- Chaque bloc d'encodeur et de décodeur utilise des mécanismes d'auto-attention pour pondérer l'importance des différents éléments d'entrée.

Attention is All You Need:



"Attention is All You Need" est un article de recherche publié en 2017 par des chercheurs de Google, qui a présenté le modèle Transformer, une architecture inédite qui a révolutionné le domaine du traitement du langage naturel (NLP) et est devenue la base des LLM que nous connaissons aujourd'hui - tels que GPT, PaLM et d'autres. L'article propose une architecture de réseau neuronal qui remplace les réseaux neuronaux récurrents (RNN) et les réseaux neuronaux convolutifs (CNN) traditionnels par un mécanisme entièrement basé sur l'attention.
Le modèle Transformer utilise l'auto-attention pour calculer les représentations des séquences d'entrée, ce qui lui permet de capturer les dépendances à long terme et de paralléliser efficacement le calcul. Les auteurs démontrent que leur modèle atteint des performances de pointe dans plusieurs tâches de traduction automatique et surpasse les modèles précédents qui s'appuient sur des RNN ou des CNN.

L'architecture du Transformer se compose d'un encodeur et d'un décodeur, chacun étant constitué de plusieurs couches. Chaque couche se compose de deux sous-couches : un mécanisme d'auto-attention à têtes multiples et un réseau neuronal de type feed-forward. Le mécanisme d'auto-attention à têtes multiples permet au modèle de s'intéresser à différentes parties de la séquence d'entrée, tandis que le réseau neuronal en aval applique une couche entièrement connectée par point à chaque position séparément et identiquement.

Le modèle Transformer utilise également des connexions résiduelles et une normalisation des couches pour faciliter l'apprentissage et éviter l'ajustement excessif. En outre, les auteurs introduisent un schéma de codage positionnel qui code la position de chaque jeton dans la séquence d'entrée, ce qui permet au modèle de saisir l'ordre de la séquence sans qu'il soit nécessaire de recourir à des opérations récurrentes ou convolutifs

Vous pouvez lire l'article sur les transformateurs ici

Partie pratique:

I. Problématique :

Avec l'augmentation continue de la complexité des projets de développement logiciel, la documentation du code devient un défi majeur pour les équipes de développement. Les commentaires de code jouent un rôle crucial en facilitant la compréhension et la maintenance du code, mais leur rédaction est souvent négligée en raison du manque de temps ou de la charge cognitive. Le manque de commentaires adéquats peut entraîner des erreurs de compréhension, des difficultés de collaboration et un allongement des cycles de développement.

Dans ce contexte, la génération automatique de commentaires pour le code Java représente une solution potentielle pour améliorer la qualité et la maintenabilité du code. Cependant, la création d'un système efficace pour générer des commentaires pertinents et informatifs pose plusieurs défis :

- **Compréhension du contexte :** Comment peut-on développer un modèle qui comprend le contexte du code Java afin de générer des commentaires qui reflètent fidèlement la logique et l'intention du code ?

- **Précision et pertinence :** Comment garantir que les commentaires générés sont non seulement précis, mais aussi pertinents et utiles pour les développeurs qui consulteront le code ?

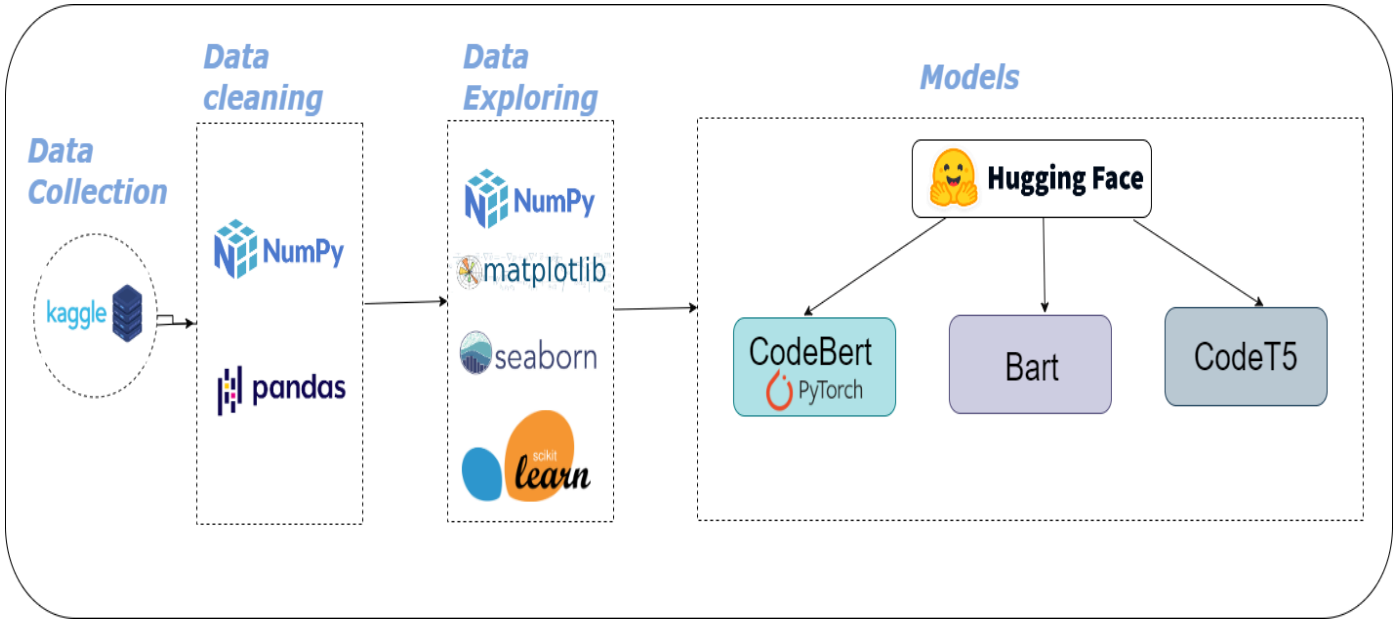
- **Adaptabilité :** Comment le modèle peut-il être adapté à différents styles de codage et à diverses pratiques de programmation pour produire des commentaires cohérents et adaptés à chaque contexte spécifique ?

- **Évaluation et amélioration :** Quels critères et méthodes d'évaluation peuvent être utilisés pour mesurer la qualité des commentaires générés et comment peut-on itérer et améliorer le modèle en fonction des retours et des besoins des utilisateurs ?

Ce projet vise à développer et à évaluer une approche basée sur des techniques d'intelligence artificielle pour la génération automatique de commentaires de

code Java, en explorant des méthodes innovantes pour surmonter ces défis et améliorer la qualité de la documentation du code.

Architecture générale du projet:



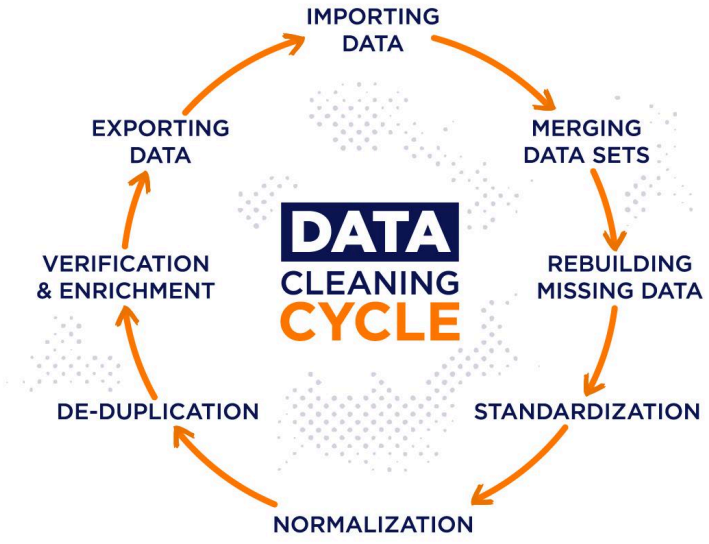
Dans ce projet, nous explorons le processus complexe de génération automatique de commentaires de code en utilisant des techniques de traitement du langage naturel (NLP). Le flux de travail commence par la collecte de données sur **Kaggle**, suivie de phases rigoureuses de nettoyage et d'exploration des données en utilisant des outils puissants tels que **NumPy**, **Pandas**, **Matplotlib**, et **Seaborn**. Ces étapes garantissent que l'ensemble de données est prêt pour l'entraînement des modèles avancés.

Pour analyser et prédire les commentaires, nous utilisons des modèles de pointe comme **CodeBERT**, **BART** et **CodeT5**, qui font partie de l'écosystème Hugging Face. Ces modèles sont spécifiquement choisis pour leur capacité à comprendre et générer du langage naturel à partir d'entrées de code, ce qui les rend idéaux pour la tâche de génération automatique de commentaires de code. Ce pipeline représente une approche robuste pour résoudre l'un des défis clés du développement logiciel : améliorer la lisibilité et la maintenabilité du code grâce à une documentation automatisée.

II. Les étapes de la réalisation du projet :

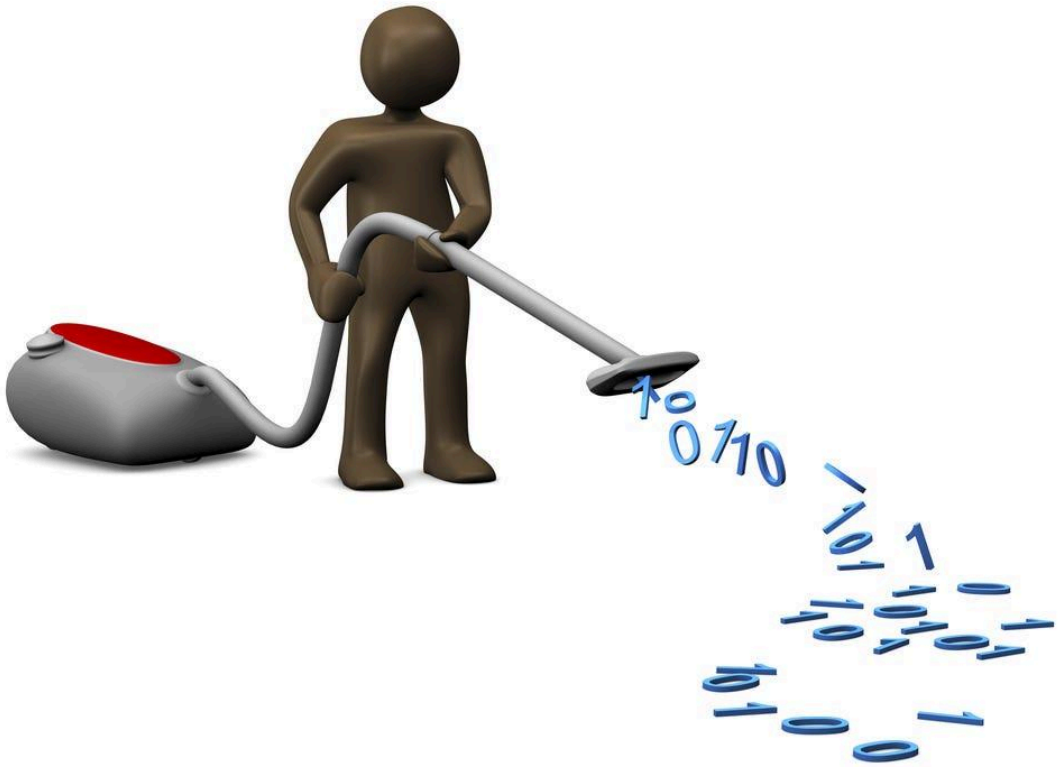
La réalisation de ce projet de génération automatique de commentaires pour le code Java, en utilisant des techniques de traitement du langage naturel (NLP), se déroule en plusieurs étapes clés. La première étape consiste à la collecte et à la préparation des données. Cela implique la constitution d'un corpus de code Java annoté avec des commentaires de qualité, ainsi que le prétraitement de ces données pour assurer leur compatibilité avec les modèles NLP. La seconde étape concerne le développement et l'entraînement du modèle de génération de commentaires. Pour ce faire, nous exploitons des architectures avancées telles que BERT ou GPT, adaptées aux besoins spécifiques de la génération de texte à partir du code. La troisième étape est l'évaluation du modèle, où nous mesurerons la qualité des commentaires générés en termes de précision, de pertinence et de clarté, en utilisant des métriques spécifiques et des évaluations qualitatives par des développeurs expérimentés. Enfin, nous procéderons à l'itération et à l'amélioration du modèle en fonction des résultats obtenus, afin d'optimiser ses performances et d'adapter les commentaires générés aux besoins des utilisateurs finaux. Cette approche méthodique nous permettra de créer un système robuste et efficace pour améliorer la documentation du code Java grâce aux capacités avancées du NLP.

A. Collection des données :



Pour le développement et l'évaluation du modèle de génération automatique de commentaires de code Java, nous utiliserons le jeu de données fourni par le projet DeepCom, disponible sur le repository GitHub DeepCom ou bien plus précisément dans la plateforme KAGGLE sous le lien suivant: https://www.kaggle.com/datasets/frabbisw/code_comments . Ce jeu de données constitue une ressource précieuse pour notre projet, car il contient des paires de code source Java et leurs commentaires correspondants, offrant ainsi un corpus riche et diversifié pour l'entraînement et la validation des modèles. La collecte des données se fait en téléchargeant le fichier compressé '*data.7z*' depuis le repository. Une fois décompressé, le jeu de données est structuré de manière à inclure des exemples variés de code source accompagnés de commentaires détaillés, ce qui permet de créer un modèle capable de générer des commentaires cohérents et informatifs. La qualité et la diversité des paires code-commentaire dans ce jeu de données sont essentielles pour entraîner un modèle robuste qui peut comprendre et expliquer efficacement le code Java. En intégrant ces données dans notre pipeline de traitement, nous nous assurons de disposer d'une base solide pour entraîner notre modèle, évaluer ses performances et l'améliorer en fonction des retours obtenus.

B. Nettoyage des données



La phase de nettoyage des données a été menée avec une attention méticuleuse pour garantir la qualité et la pertinence du jeu de données destiné à l'entraînement de notre modèle de génération automatique de commentaires de code. Nous avons commencé par enlever les caractères non **ASCII** des dataframes, ce qui a permis de s'assurer que seules les données textuelles standard étaient conservées, éliminant ainsi tout caractère spécial pouvant poser des problèmes lors du traitement du texte. Ensuite, nous avons effectué une vérification approfondie pour détecter la présence de **valeurs manquantes** dans le dataframe d'entraînement. Cette étape a été suivie par une visualisation des valeurs manquantes, nous permettant de mieux comprendre leur distribution et leur impact potentiel sur l'intégrité des données.

Pour nettoyer les commentaires, nous avons supprimé toutes les balises **HTML**, ce qui a permis d'éliminer les éléments de mise en forme inutiles et de garantir que les commentaires soient présentés dans un format brut et lisible. Cette opération a été cruciale pour éviter que le modèle ne soit influencé par des balises HTML non pertinentes. Par la suite, nous avons procédé à la suppression des **valeurs manquantes** en les remplaçant par des valeurs par défaut ou en

éliminant les entrées concernées. Nous avons également éliminé les **espaces doubles** et les **doublons** afin de nettoyer le texte et de garantir la cohérence des données. Ces actions ont contribué à réduire le bruit et à améliorer la qualité globale des données.

Enfin, nous avons retiré les **JavaDocs** des commentaires dans le dataframe, en conservant uniquement la description pertinente. Cette démarche a permis de simplifier le contenu des commentaires et de se concentrer sur les informations utiles pour la génération automatique, en supprimant les sections formelles de documentation qui pourraient être superflues pour notre modèle. Donc, ces étapes de nettoyage ont assuré que les données sont non seulement propres mais également adaptées pour entraîner un modèle performant et fiable, en optimisant la qualité des informations fournies.

C. Exploration des données :

La phase d'exploration des données a été soigneusement orchestrée pour obtenir des insights précieux sur la structure et la distribution des données, afin d'orienter efficacement l'entraînement du modèle. Nous avons d'abord développé une fonction pour obtenir le nombre de chaque token dans une colonne de DataFrame, ce qui nous a permis d'analyser la fréquence des différents tokens et d'identifier les termes les plus courants dans les commentaires. En complément, nous avons créé une fonction pour tracer un graphique à barres des tokens les

plus fréquents, en affichant les `top_k` tokens les plus courants, ce qui a facilité la visualisation des termes dominants et leur distribution dans les données.

Pour obtenir une vue d'ensemble des caractéristiques statistiques des données, nous avons affiché des statistiques descriptives, incluant la moyenne, la médiane, l'écart type, ainsi que les valeurs minimales et maximales, ainsi que les percentiles 25 et 75. Ces statistiques ont fourni un aperçu détaillé de la répartition des longueurs des tokens et des commentaires, nous aidant à comprendre les variations et les tendances dans les données.

Nous avons également créé un histogramme pour comparer la distribution des longueurs des tokens entre les méthodes et les commentaires. Cette comparaison visuelle a permis de mettre en évidence les différences potentielles entre ces deux types de données et d'identifier les éventuelles anomalies ou biais.

Enfin, pour évaluer la similarité entre les commentaires, nous avons calculé la similarité cosinus en utilisant la méthode **TF-IDF** pour transformer le texte en vecteurs. Cette mesure a permis de quantifier la similarité entre les commentaires, fournissant des informations sur leur cohérence et leur diversité, et aidant à mieux comprendre les relations entre les différentes descriptions présentes dans le jeu de données.

D.Entrainement :

Ce projet se concentre sur l'utilisation de modèles de NLP avancés pour la génération automatique de commentaires de code source en Java. trois modèles principaux ont été employés dans cette étude :

1. ***ROBERTa :*** Une variante du modèle BERT, optimisée pour des performances accrues, est utilisée ici dans un cadre Seq2Seq (Sequence-to-Sequence) pour capturer le contexte du code source et l'encoder .
2. ***BART :*** Un autre modèle Seq2Seq, qui combine les capacités de bidirectionnalité des encodeurs avec un décodeur autorégressif, est également testé pour évaluer sa performance dans la génération de commentaires de code source.

3. ***CodeT5 :*** un modèle basé sur les transformateurs conçu pour la compréhension et la génération de code source. C'est une extension du modèle T5 (Text-to-Text Transfer Transformer), adapté spécifiquement pour les langages de programmation.

Premier Model :

Qu'est-ce que BERT ?

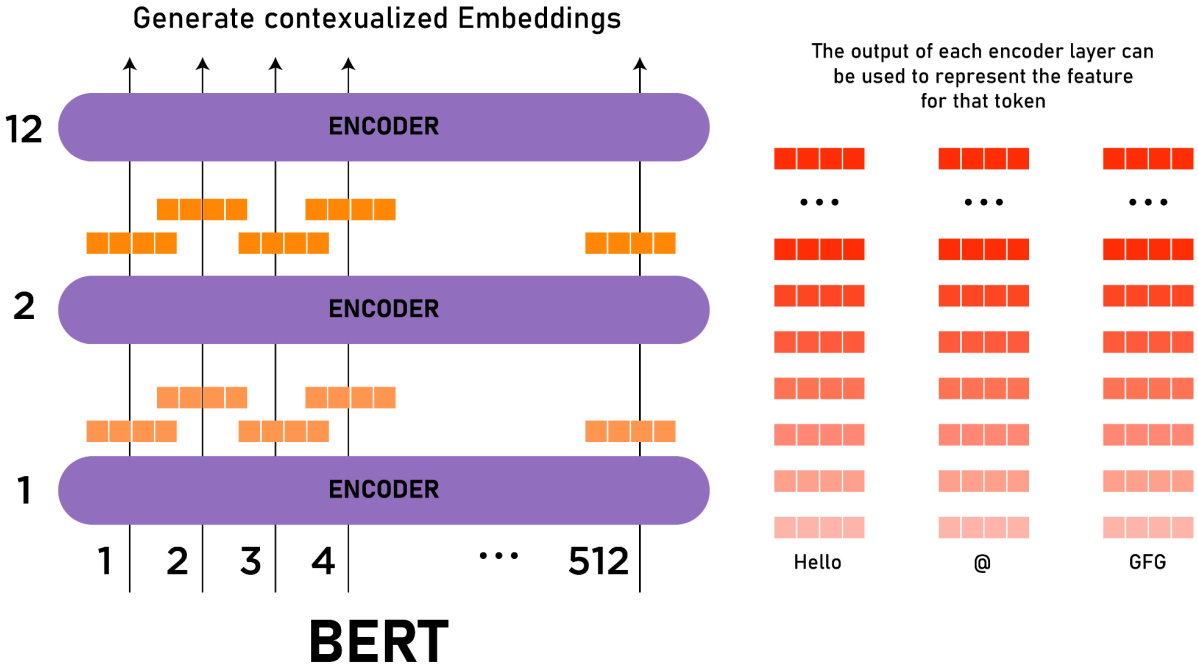
BERT est un modèle de traitement du langage naturel (NLP) développé par Google AI et introduit en 2018. Il est conçu pour comprendre le contexte des mots dans une phrase en utilisant une approche bidirectionnelle, ce qui signifie qu'il prend en compte les mots à la fois à gauche et à droite d'un mot cible pour mieux saisir son sens.

- Principales Caractéristiques :

Bidirectionnalité : Contrairement aux modèles de langage traditionnels qui lisent les textes de gauche à droite (ou de droite à gauche), BERT lit les textes dans les deux directions simultanément. Cela lui permet de capturer un contexte plus riche et plus précis pour chaque mot.
Pré-entraînement et Ajustement (Fine-Tuning) : BERT est d'abord pré-entraîné sur un grand corpus de texte avec deux tâches principales :

Masquage des mots (Masked Language Model, MLM) : Certains mots de la phrase sont masqués, et le modèle apprend à prédire ces mots en utilisant le contexte des autres mots.

Prédiction de la relation entre les phrases (Next Sentence Prediction, NSP) : Le modèle apprend à prédire si une phrase suit logiquement une autre dans un texte.

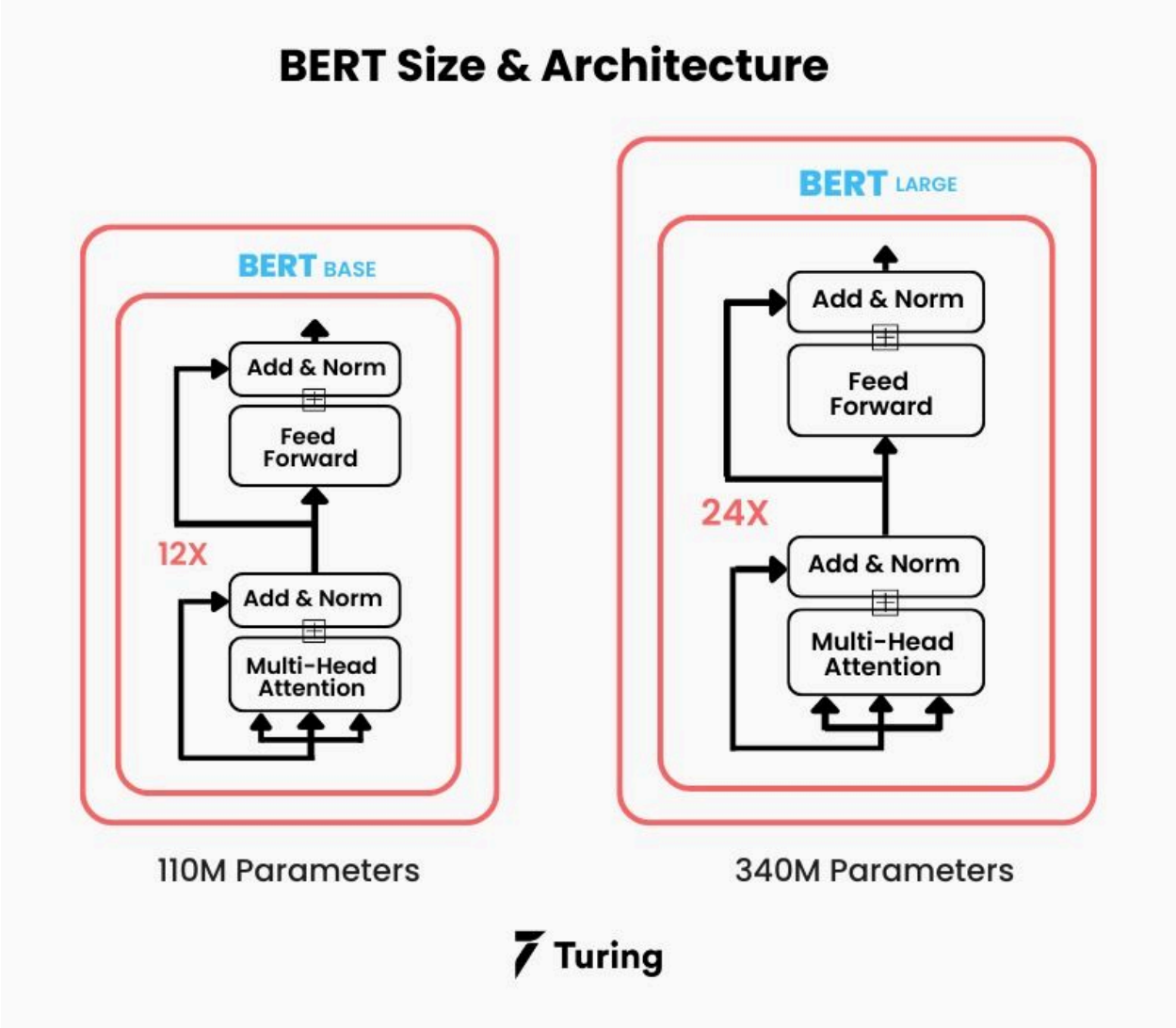


Transformers : BERT est basé sur l'architecture des Transformers, qui utilise des mécanismes d'attention pour gérer les dépendances contextuelles entre les mots d'une phrase. Les Transformers sont efficaces pour traiter des séquences de texte et permettent au modèle de se concentrer sur des parties spécifiques du texte.

Représentation Contextuelle : BERT génère des représentations contextuelles pour chaque mot dans une phrase, ce qui signifie que la représentation d'un mot peut changer en fonction du contexte dans lequel il apparaît.

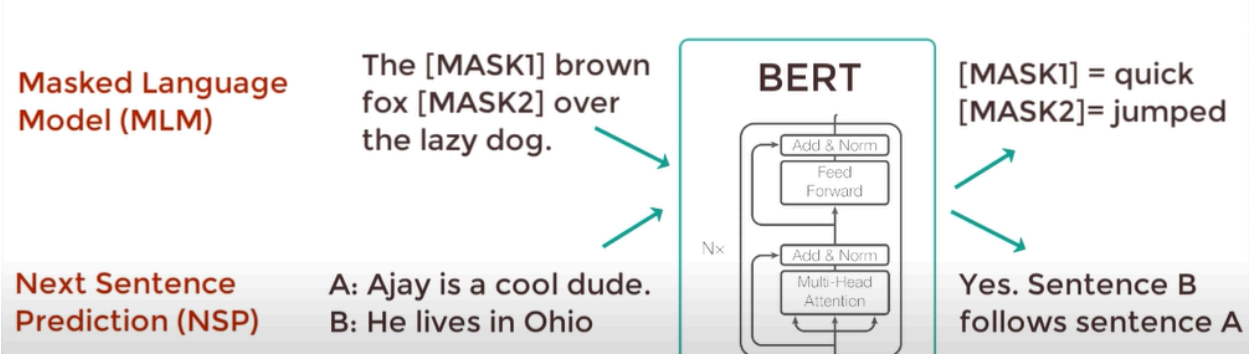
- ## Modèles BERT Notables :

1. **BERT-Base** : Composé de 12 couches de Transformer, 768 unités cachées, et 110 millions de paramètres.
2. **BERT-Large** : Composé de 24 couches de Transformer, 1024 unités cachées, et 345 millions de paramètres.



MLM & NSP

En BERT, deux tâches principales sont utilisées pendant l'entraînement : **MLM (Masked Language Modeling)** et **NSP (Next Sentence Prediction)**.



1. MLM (Masked Language Modeling) :

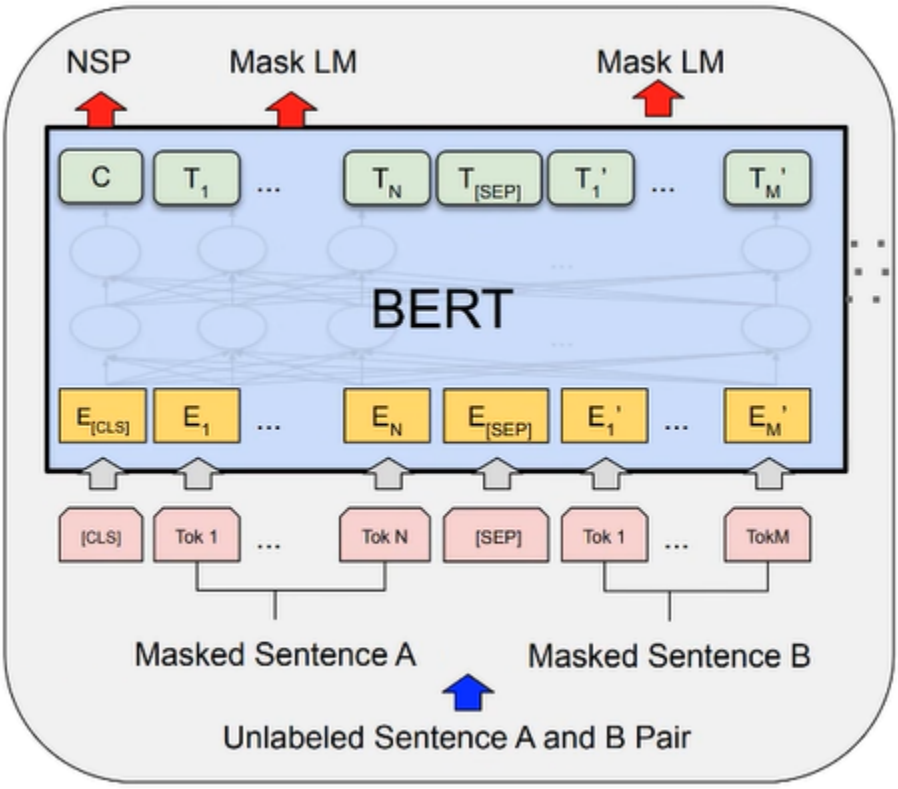
- **Concept** : Le modèle masque aléatoirement certains mots dans une phrase et essaie de prédire ces mots masqués en se basant sur le contexte des mots non masqués.
- **But** : Apprendre à comprendre le contexte bidirectionnel (avant et après) d'un mot dans une phrase, ce qui permet à BERT de mieux saisir les relations entre les mots.
- **Comment ça marche** : Dans une phrase donnée, environ 15% des mots sont masqués (remplacés par le token `[MASK]`). Le modèle doit prédire les mots originaux à partir des indices fournis par le reste de la phrase.

2. NSP (Next Sentence Prediction) :

- **Concept** : Le modèle reçoit deux phrases et doit prédire si la seconde phrase suit directement la première dans le texte d'origine.
- **But** : Apprendre la relation entre deux phrases, ce qui aide dans des tâches comme la compréhension de texte, les questions-réponses, et les tâches nécessitant la cohérence entre plusieurs phrases.
- **Comment ça marche** : Pendant l'entraînement, le modèle est présenté avec paires de phrases :
    - **50% des cas** : La deuxième phrase est la phrase qui suit réellement la première.
    - **50% des cas** : La deuxième phrase est une phrase aléatoire du corpus. Le modèle apprend à distinguer les paires cohérentes des incohérentes.

- ## L'architecture de BERT :

L'architecture de BERT est basée sur le modèle Transformer, et elle se compose de plusieurs composants clés qui travaillent ensemble pour créer des représentations contextuelles des mots dans une phrase.



Embedding Layer (Couche d'Incorporation) :

- **Token Embeddings** : Chaque mot dans une phrase est converti en un vecteur d'embedding fixe à l'aide d'une table d'embeddings. Cette table est apprise pendant l'entraînement.
- **Segment Embeddings** : Pour les tâches nécessitant la compréhension de relations entre deux phrases (comme la prédiction de la phrase suivante), les embeddings de segment sont utilisés pour indiquer à quel segment (phrase) appartient chaque mot.
- **Positional Embeddings** : Comme le modèle Transformer ne possède pas de notion intrinsèque d'ordre des mots, des embeddings positionnels sont ajoutés pour représenter la position de chaque mot dans la séquence.

Transformer Layers (Couches Transformer) :

- **Self-Attention Mechanism (Mécanisme d'Attention Autonome) :**

  Multi-Head Attention : Permet au modèle de se concentrer sur différentes parties du texte simultanément, en créant plusieurs "têtes" d'attention pour capturer diverses relations entre les mots.

  Attention Scores : Calculés pour chaque paire de mots dans la phrase, indiquant l'importance relative d'un mot par rapport à un autre dans le contexte donné.

- **Feed-Forward Neural Network (Réseau de Neurones à Propagation Avant) :**

  Applique une transformation non linéaire sur les sorties de la couche d'attention. Chaque couche Transformer contient deux sous-couches : une pour l'attention et une pour le réseau de neurones feed-forward.

- **Layer Normalization (Normalisation de Couche) :**

  Appliquée après chaque sous-couche pour stabiliser l'apprentissage et accélérer la convergence.

- **Residual Connections (Connexions Résiduelles) :**

  Permettent aux informations de passer directement d'une sous-couche à l'autre, facilitant ainsi l'apprentissage et la propagation des gradients.

Output Layer (Couche de Sortie) :

- **Pooling (Regroupement) :**

  BERT utilise une opération de pooling pour extraire une représentation globale de la séquence. Pour des tâches comme la classification de texte, la sortie de la couche [CLS] (une balise spéciale ajoutée au début de chaque séquence) est souvent utilisée comme représentation de l'ensemble de la séquence.

- **Task-Specific Heads (Têtes Spécifiques à la Tâche) :**

Pour chaque tâche spécifique (comme la classification ou l'extraction d'informations), des couches supplémentaires peuvent être ajoutées au-dessus du modèle de base pour effectuer des prédictions sur la sortie de BERT.

- Résumé Visuel :

- **Entrée :** Phrase → [Token Embeddings + Segment Embeddings + Positional Embeddings]
- **Couches Transformer :** [Self-Attention + Feed-Forward Neural Network + Layer Normalization + Residual Connections] x N (où N est le nombre de couches)
- **Sortie :** Représentation contextuelle → [Pooling + Tâche Spécifique]

RoBERTa est une variante optimisée de BERT (Bidirectional Encoder Representations from Transformers). Alors que BERT utilise une architecture transformer bidirectionnelle pour comprendre le contexte des mots, RoBERTa apporte des améliorations spécifiques dans l'entraînement ou la configuration du modèle pour améliorer ses performances sur certaines tâches.

Comparaison entre BERT et Roberta :

- Données et Entraînement :
- ***BERT :*** Entraîné sur 16 Go de texte avec une tâche de prédiction de la prochaine phrase (NSP).
- ***ROBERTa :*** Entraîné sur 160 Go de texte, sans la tâche NSP, avec plus de temps d'entraînement.
- Masquage :
- ***BERT :*** Utilise un masquage statique (les mêmes mots sont masqués à chaque fois).
- ***RoBERTa :*** Utilise un masquage dynamique (les mots masqués changent à chaque itération).
- Performances :
- ***BERT :*** Bonnes performances, mais limité par la quantité de données et la méthode d'entraînement.

- ***ROBERTa :*** Meilleures performances grâce à plus de données, à un masquage dynamique, et à l'abandon de la tâche NSP.

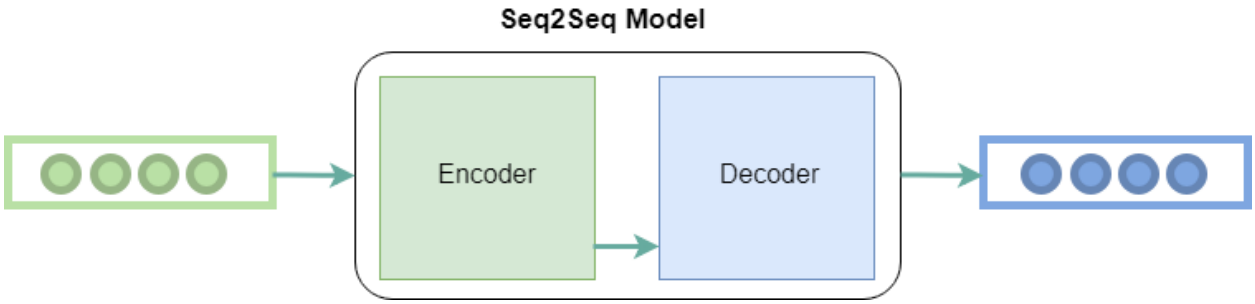
- CodeBERT :

CodeBERT est une variante de BERT spécialement conçue pour les langages de programmation. Il est utilisé pour diverses tâches liées au code, telles que la summarisation de code, la complétion de code et la recherche de code. Il a été pré-entraîné sur un grand corpus de code provenant de divers langages de programmation, ce qui lui permet de comprendre et de générer des extraits de code.

Quelques caractéristiques clés de CodeBERT :

1. **Pré-entraînement** : CodeBERT est pré-entraîné sur une grande quantité de données de code et de texte naturel. Cela lui permet de comprendre à la fois le code et le langage naturel, ce qui est utile pour des tâches impliquant les deux, telles que la summarisation de code.
2. **Encodeurs Doubles** : Il utilise une approche à double encodeur où un encodeur traite le code et l'autre traite le texte naturel. Cela permet de faire le lien entre le code et le texte.
3. **Ajustement Fin** : Après le pré-entraînement, CodeBERT peut être ajusté pour des tâches spécifiques telles que la complétion de code, la détection de bogues ou la summarisation de code en l'entraînant sur des ensembles de données étiquetées pour ces tâches.
4. **Multilingue** : Il prend en charge plusieurs langages de programmation, ce qui le rend polyvalent pour des projets impliquant du code dans différents langages.

- Qu'est-ce que Seq2Seq ? :



Le modèle Sequence-to-Sequence (Seq2Seq) est une architecture populaire pour les tâches de génération de séquences dans le traitement du langage naturel (NLP). Il se compose généralement de deux parties : un encodeur qui encode la séquence d'entrée en un vecteur de contexte et un décodeur qui génère la séquence de sortie à partir de ce vecteur. Dans cette étude, nous explorons l'utilisation de BERT (Bidirectional Encoder Representations from Transformers) comme encodeur et d'un décodeur Transformer pour la génération de séquences.

- ## Architecture du modèle

Le modèle Seq2Seq étudié dans le premier modèle se compose des deux principaux composants suivants :

RoBERTa / CodeBERT

➢ **Rôle :** Encodage
➢ **Description :** RoBERTa et CodeBERT sont des modèles de type Transformer pré-entraînés principalement pour la compréhension du langage. Dans notre configuration, RoBERTa ou CodeBERT est utilisé comme **encodeur** dans le modèle Seq2Seq. Leur rôle est de transformer les entrées (par exemple, le texte source) en une représentation vectorielle de haute dimension qui capture les informations sémantiques et contextuelles du texte.

Seq2Seq (Encoder-Decoder)

➢ **Rôle :** Génération de texte

➢ **Description :** Voici les composants spécifiques :
  - **Encodeur (RoBERTa/CodeBERT) :** Encode la séquence d'entrée en une représentation contextuelle.
  - **Décodeur (TransformerDecoder) :** Prend la représentation contextuelle générée par l'encodeur et génère une séquence de sortie. Il utilise des couches de Transformer pour produire des prédictions de séquence, en utilisant des mécanismes d'attention pour se concentrer sur différentes parties de la séquence d'entrée pendant la génération.

- **RoBERTa vs. CodeBERT**

➢ **CodeBERT :** Une variante de RoBERTa spécialement pré-entraînée sur des données de code source pour les tâches de traitement du code, telle que la génération de commentaires de code. Dans notre code, nous avons choisi `microsoft/codebert-base` comme modèle de base.

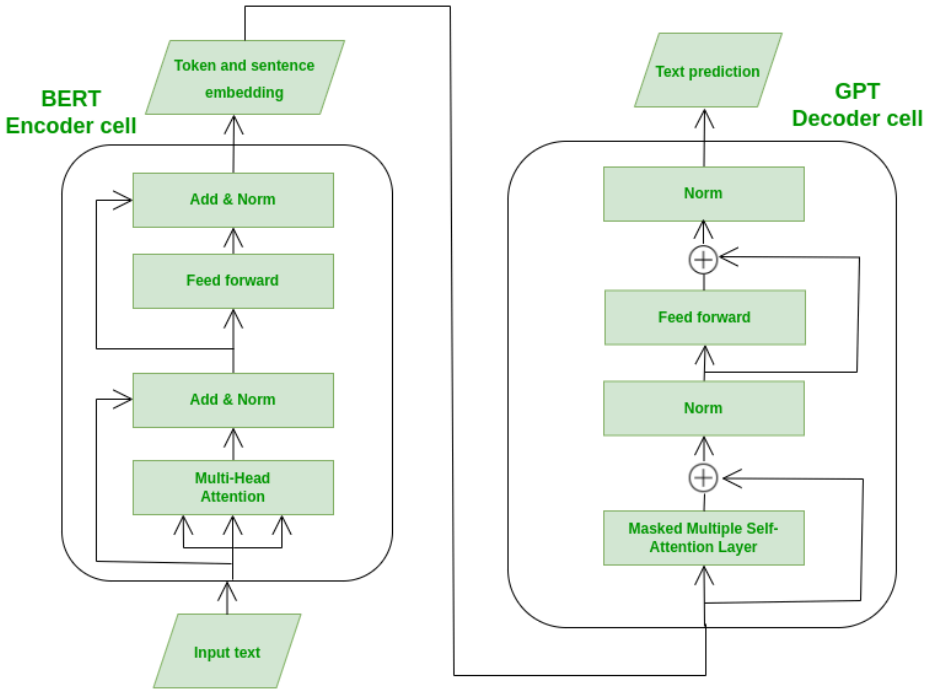
➢ **RoBERTa :** Un modèle de langage général pré-entraîné sur un large corpus de texte pour des tâches NLP générales.

Le modèle final combine ces éléments pour accomplir des tâches comme la génération automatique de commentaires en code, en prenant en entrée des séquences de code, les encodant en représentations contextuelles, puis les décodant en commentaires ou d'autres séquences pertinentes.

Deuxieme Model :

Qu'est-ce que BART ?

BART (Bidirectional and Auto-Regressive Transformers) est un modèle de traitement du langage naturel développé par Facebook AI et introduit en 2019. Il combine les idées des modèles bidirectionnels (comme BERT) et autoregressifs (comme GPT) pour atteindre des performances élevées sur diverses tâches NLP.



- ## Architecture de BART :

1. ### Embedding Layer (Couche d'Incorporation) :

   Token Embeddings : Convertit chaque mot en un vecteur dense à l'aide d'une table d'embeddings.

   Positional Embeddings : Ajoute des informations sur la position des mots dans la séquence, car BART, comme les Transformers, n'a pas d'ordre intrinsèque des mots.

2. ### Encoder (Encodeur) :

   Self-Attention Mechanism (Mécanisme d'Attention Autonome) :

   Multi-Head Attention : Permet au modèle de se concentrer sur différentes parties du texte simultanément.

   Attention Scores : Mesurent l'importance relative des mots dans le contexte d'une phrase.

Feed-Forward Neural Network (Réseau de Neurones à Propagation Avant) :

Applique une transformation non linéaire sur les sorties de l'attention.

Layer Normalization (Normalisation de Couche) :

Stabilise et accélère l'apprentissage en normalisant les sorties des sous-couches.

Residual Connections (Connexions Résiduelles) :

Facilitent la propagation des gradients et aident à l'apprentissage en ajoutant directement les entrées aux sorties des sous-couches.

3. **Decoder (Décodeur) :**

Self-Attention Mechanism (Mécanisme d'Attention Autonome) :

Masked Multi-Head Attention : Empêche le modèle de voir les futurs tokens lors de la génération de texte (important pour la génération autoregressive).

Encoder-Decoder Attention (Attention Encodeur-Décodeur) :

Permet au décodeur de se concentrer sur les représentations du texte encodé pour générer le texte cible.

Feed-Forward Neural Network (Réseau de Neurones à Propagation Avant) :

Comme dans l'encodeur, il applique une transformation non linéaire sur les sorties d'attention.

Layer Normalization (Normalisation de Couche) :

Appliquée après chaque sous-couche pour stabiliser l'apprentissage.

Residual Connections (Connexions Résiduelles) :

Assurent une meilleure propagation des gradients et facilitent l'apprentissage.

4. **Output Layer (Couche de Sortie) :**

   Logits de Prédiction : Produits par le décodeur pour prédire la probabilité de chaque mot dans le vocabulaire, utilisés pour la génération de texte ou la prédiction de séquences.

- **Résumé Visuel :**

- **Entrée :** Phrase → [Token Embeddings + Positional Embeddings]
- **Encodeur :** [Self-Attention + Feed-Forward Neural Network + Layer Normalization + Residual Connections] x N (où N est le nombre de couches)
- **Décodeur :** [Masked Self-Attention + Encoder-Decoder Attention + Feed-Forward Neural Network + Layer Normalization + Residual Connections] x N
- **Sortie :** Logits de Prédiction

Troisième Model :

Qu'est-ce que CodeT5 ? :

CodeT5 est un modèle de traitement du langage naturel (NLP) spécialement conçu pour les tâches de traitement de code source. Développé par Microsoft Research, CodeT5 est basé sur l'architecture des transformateurs, similaire à T5 (Text-to-Text Transfer Transformer), mais adapté pour travailler avec des données de code.

- Caractéristiques de CodeT5 :

1. **Prétraitement du Code** : CodeT5 est formé sur des corpus de code source dans plusieurs langages de programmation. Il utilise des techniques de prétraitement spécifiques aux langages de programmation pour encoder efficacement le code.
2. **Modèle Multi-Task** : CodeT5 est capable d'effectuer plusieurs tâches, telles que la génération de code, la complétion de code, et la génération de commentaires. Cela le rend utile pour divers cas d'utilisation dans le développement de logiciels.

3. **Architecture Transformer** : Il utilise l'architecture des transformateurs, qui est connue pour sa capacité à capturer des relations complexes dans les données séquentielles.
4. **Génération de Commentaires** : CodeT5 peut générer des commentaires pour le code source, ce qui peut être utile pour améliorer la documentation et la compréhension du code.

- L'architecture de CodeT5 :

Encodage et Décodage avec Transformateurs

- **Encoder (BERT-like)** : CodeT5 utilise un encodeur basé sur des transformateurs, similaire à BERT. Il convertit le code source en une représentation interne. Cela permet de capturer les dépendances contextuelles dans le code.
- **Decoder (T5-like)** : Le décodeur est utilisé pour générer des sorties, telles que des commentaires ou des compléments de code. Il est basé sur l'architecture des transformateurs, comme dans T5.

- Variantes et Paramètres de CodeT5 :

- **Small** : Une version réduite avec 60 millions de paramètres, adaptée aux applications nécessitant moins de puissance de calcul.
- **Base** : Une variante standard avec 220 millions de paramètres, généralement utilisée comme modèle de référence pour diverses tâches.
- **Large** : Une version plus grande avec 770 millions de paramètres, conçue pour des tâches nécessitant une plus grande capacité de traitement.
- **XL (3B)** : Une version encore plus massive avec 3 milliards de paramètres, destinée aux applications complexes et aux grands ensembles de données.
- **XXL (11B)** : La version la plus grande avec 11 milliards de paramètres, conçue pour les applications les plus exigeantes en termes de calcul et de traitement de données.

Evaluation de LLM :

- Évaluation Automatique :

- **BLEU (Bilingual Evaluation Understudy)** : Bien que principalement utilisé pour les traductions, le score BLEU peut être adapté pour évaluer la similarité entre les commentaires générés et les commentaires de référence. Il mesure la précision des n-grammes.
- **ROUGE (Recall-Oriented Understudy for Gisting Evaluation)** : ROUGE peut évaluer la qualité des commentaires en comparant les n-grammes ou les phrases des commentaires générés avec ceux des commentaires de référence. Il est particulièrement utile pour les tâches de résumé.
- **METEOR (Metric for Evaluation of Translation with Explicit ORdering)** : Évalue la qualité des commentaires en prenant en compte les synonymes, les variations grammaticales, et les correspondances exactes.
- **F1-Score** : Pour des tâches spécifiques comme l'extraction ou la classification de commentaires, l'F1-score peut évaluer la précision et le rappel des informations extraites.

- Évaluation Humaine :

- **Qualité du Commentaire** : Les annotateurs humains évaluent les commentaires générés en termes de clarté, précision, et utilité. Ils vérifient si les commentaires sont compréhensibles, pertinents, et correctement associés aux segments de code.
- **Comparaison Pair-à-Pair** : Les annotateurs comparent les commentaires générés par différents modèles ou versions du modèle pour déterminer lequel produit les meilleurs commentaires.
- **Vérification de la Conformité** : Vérifier si les commentaires générés respectent les conventions de style et les bonnes pratiques de documentation pour le code Java.

Problèmes de Performance des Modèles NLP pour la Génération Automatique de Commentaires de Code Source Java

Lors de l'utilisation des modèles RoBERTa, BART, et CodeT5 pour générer automatiquement des commentaires de code source en Java, nous avons observé des performances globalement faibles. Cela peut être attribué principalement à deux facteurs :

1. **Capacité de Compréhension du Code par les Modèles :**
    - **CodeT5** se limite à reconnaître les noms de fonctions et de variables, sans analyser en profondeur la logique ou la structure du code. Cela indique que le modèle manque de la capacité nécessaire pour comprendre les contextes complexes et les relations entre les différentes parties du code, ce qui est crucial pour générer des commentaires pertinents.
2. **Quantité Limitée de Données d'Entraînement :**
    - La qualité et la quantité des données d'entraînement jouent un rôle essentiel dans la performance des modèles de NLP. Nos ressources limitées nous ont empêchés d'entraîner les modèles sur un ensemble de données suffisamment large et diversifié. Un entraînement sur un volume restreint de données ne permet pas aux modèles de capturer pleinement les nuances et les complexités du code source, ce qui affecte leur capacité à générer des commentaires précis et pertinents.

Ces limitations combinées ont entraîné des performances insuffisantes des modèles, soulignant l'importance d'un entraînement sur des ensembles de données plus vastes pour améliorer la qualité des générateurs automatiques de commentaires de code.

Conclusion :

Ce projet a visé à explorer et à comparer la performance de trois modèles de traitement du langage naturel dans la génération automatique de commentaires pour des codes Java : CodeBERT, CodeT5, et BART. Chacun de ces modèles offre des capacités distinctes en matière de compréhension et de génération de texte, ce qui a permis de mener une analyse approfondie de leurs forces et de leurs limites dans le contexte de la génération de commentaires.

Ainsi que ce projet souligne l'importance de choisir le modèle approprié en fonction des exigences spécifiques de la tâche et ouvre la voie à des améliorations futures, telles que l'intégration de techniques d'optimisation supplémentaires et l'exploration de modèles plus récents pour une meilleure performance en génération de texte.

Remerciement:

Nous tenons à exprimer notre profonde gratitude à toutes les personnes qui ont contribué à la réalisation de ce projet .

Tout d'abord, nous adressons nos remerciements les plus chaleureux à Monsieur CHERRADI Mohamed, notre encadrant, pour sa précieuse guidance, ses conseils avisés, et son soutien constant tout au long de ce projet. Son expertise et sa disponibilité ont été essentielles pour mener à bien ce travail. Monsieur CHERRADI a su nous orienter et nous inspirer, en nous encourageant à surmonter les défis et à atteindre nos objectifs. Sa rigueur intellectuelle et son dévouement nous ont permis d'approfondir nos connaissances et de développer des compétences cruciales dans le domaine de DATA Science.

Nous souhaitons également remercier l'ensemble des professeurs du département MI pour leur enseignement et leur encadrement tout au long de cette année. Leur passion et leur dévouement ont grandement enrichi nos connaissances et compétences.