Royaume du Maroc

Haut-commissariat au plan

Ecole des Sciences de l’Information

Année Universitaire : 2023-2024

Module : Text Mining & Analytics



**Rapport de projet Text Mining & Analytics :**

Application de « News Summarisation »

Groupe\_5

3ème ACI – ICSD

Réalisé par :

ELGHAZI Soufiane

MAASRI Amine

OUAHIB Yassine

DAHMOU Youssef

**Année Universitaire : 2023/2024**

Table des matières

[**Liste des figures 3**](#_Toc154872659)

[**Introduction 4**](#_Toc154872660)

[**I. Collecte et préparation des données 5**](#_Toc154872661)

[**1. Introduction : 5**](#_Toc154872662)

[**2. Collection des données : 5**](#_Toc154872663)

[**3. Préparation des données : 8**](#_Toc154872666)

[**4. Conclusion : 10**](#_Toc154872667)

[**II. Création et évaluation des modèles 10**](#_Toc154872669)

[**1. Introduction : 10**](#_Toc154872670)

[**2. Configuration des Modèles : 10**](#_Toc154872673)

[** Modèle Pegasus : 10**](#_Toc154872674)

[** Modèle Bart : 13**](#_Toc154872675)

[** Modèle LSTM : 15**](#_Toc154872676)

[**3. Evaluation des résultats : 17**](#_Toc154872677)

[** Modèle Pegasus : 17**](#_Toc154872678)

[** Modèle BART : 17**](#_Toc154872680)

[** Modèle LSTM : 18**](#_Toc154872681)

[**4. Conclusion : 18**](#_Toc154872682)

[**III. Amélioration des performances : 18**](#_Toc154872684)

[**1. Introduction : 18**](#_Toc154872685)

[**2. LSTM : Attention Layer 18**](#_Toc154872686)

[**3. BART : NER 19**](#_Toc154872687)

[**4. Conclusion : 20**](#_Toc154872688)

[**IV. Déploiement du modèle 20**](#_Toc154872689)

[**1. Introduction 20**](#_Toc154872690)

[**2. Frontend (JavaScript - home.js): 21**](#_Toc154872691)

[**3. Backend (Python – app.py) : 23**](#_Toc154872692)

[**4. Conclusion : 24**](#_Toc154872693)

[**Conclusion : 25**](#_Toc154872694)

# Liste des figures

[**Figure 1 : Fonction de Scrapping** 5](#_Toc154871270)

[**Figure 2 : Liens des articles** 6](#_Toc154871271)

[**Figure 3 : Résumé et stockage des résultats** 6](#_Toc154871272)

[**Figure 4 : Script de connexion avec l'api newsapi** 7](#_Toc154871273)

[**Figure 5 : Sauvegarde des fichiers de données** 7](#_Toc154871274)

[**Figure 6 : Ajout des résumes** 7](#_Toc154871275)

[**Figure 7 : Données d'enrichissement** 8](#_Toc154871276)

[**Figure 8 : Dataframe du corpus** 8](#_Toc154871277)

[**Figure 9 : Fonction de traitement de données** 9](#_Toc154871278)

[**Figure 10 : Traitement corpus** 9](#_Toc154871279)

[**Figure 11 : Split dataset** 10](#_Toc154871280)

[**Figure 12 : Pegasus Tokenizer** 10](#_Toc154871281)

[**Figure 13 : Pegasus tokenization** 11](#_Toc154871282)

[**Figure 14 : Dataset personalisé** 11](#_Toc154871283)

[**Figure 15 : Création de Custom Dataset** 12](#_Toc154871284)

[**Figure 16 : Création de l'instance DataLoader** 12](#_Toc154871285)

[**Figure 17 : Initialisation de Pegasus** 12](#_Toc154871286)

[**Figure 18 : Paramètres d'entrainement Pegasus** 12](#_Toc154871287)

[**Figure 19 : Lancer Pegasus Trainer** 12](#_Toc154871288)

[**Figure 20 : Sauvgard de modèle pegasus ajusté** 13](#_Toc154871289)

[**Figure 21 : split data pour Bart** 13](#_Toc154871290)

[**Figure 22 : Transformer Dataset en Dataframe** 13](#_Toc154871291)

[**Figure 23 : Configuration de modèle Bart** 13](#_Toc154871292)

[**Figure 24 : Instance de la classe BartConditionalGeneration** 14](#_Toc154871293)

[**Figure 25 : Tokenisation de bart** 14](#_Toc154871294)

[**Figure 26 : Appliquer la fonction preprocess** 14](#_Toc154871295)

[**Figure 27 : Data Collator Bart** 14](#_Toc154871296)

[**Figure 28 : Parametres d'entrainement de Bart** 15](#_Toc154871297)

[**Figure 29 : Lancement de Bart Trainer** 15](#_Toc154871298)

[**Figure 30 : LSTM split** 15](#_Toc154871299)

[**Figure 31 : LSTM tokenization** 15](#_Toc154871300)

[**Figure 32 : texts to sequence** 16](#_Toc154871301)

[**Figure 33 : Architecture de LSTM** 16](#_Toc154871302)

[**Figure 34 : Plot de Lstm Architecture** 16](#_Toc154871303)

[**Figure 35 : Entrainement de LSTM** 17](#_Toc154871304)

[**Figure 36: Evaluation Pegasus** 17](#_Toc154871305)

[**Figure 37 : Evaluation Bart** 17](#_Toc154871306)

[**Figure 38 : Evaluation LSTM** 18](#_Toc154871307)

[**Figure 39 : LSTM avec Attention Layer** 18](#_Toc154871308)

[**Figure 40 : Fonction d'extraction des entités** 19](#_Toc154871309)

[**Figure 41 : Fonction d'ajout de NER au Bart** 19](#_Toc154871310)

[**Figure 42 : Bart avec NER trainer** 19](#_Toc154871311)

[**Figure 43 : BLEU score de modèle Bart amelioré** 20](#_Toc154871312)

[**Figure 44 : Architecture de l'application** 21](#_Toc154871313)

[**Figure 45 : Interface de l'application** 21](#_Toc154871314)

[**Figure 46 : Button generate summary** 22](#_Toc154871315)

[**Figure 47 : Affichage de Résume** 22](#_Toc154871316)

[**Figure 48 : Generation du resume** 23](#_Toc154871317)

[**Figure 49 : Summary request** 23](#_Toc154871318)

[**Figure 50 : Model** 24](#_Toc154871319)

# Introduction

Ce rapport explore le développement et l'implémentation d'une application de summarization de texte, conçue pour extraire les informations essentielles à partir d'un texte donné. L'application repose sur un modèle de deep learning entraîné à l'aide d'un vaste ensemble de données provenant de sources journalistiques diverses.

Notre démarche se déploie à travers quatre axes majeurs, débutant par la collecte et la préparation minutieuse de données issues d'articles de presse, soulignant ainsi l'importance cruciale de la qualité des informations alimentant notre modèle. Ensuite, nous explorons la création et l'évaluation de trois modèles distincts pour la génération automatique de résumés, chacun sélectionné pour ses caractéristiques spécifiques. Une étape cruciale suivante concerne l'amélioration des performances de nos modèles par l'intégration de fonctionnalités novatrices. Enfin, après ce parcours rigoureux, nous abordons avec enthousiasme le déploiement concret de notre modèle, visant à offrir une expérience utilisateur fluide et intuitive, où la complexité algorithmique se dissimule derrière une interface accessible à tous.

Notre rapport détaille ainsi chaque étape de ce voyage, mettant en lumière les défis, les choix stratégiques et les résultats obtenus.

# Collecte et préparation des données

Introduction :

La qualité des données est fondamentale dans le développement d'un modèle performant pour la génération de résumés automatiques. Dans cette phase cruciale de notre projet, nous plongeons dans le monde dynamique des actualités en ligne pour récolter les données qui alimenteront notre modèle choisi après diverses analyses.

Collection des données :

La collecte des données est l’étape qui constitue le point de départ de notre projet, Nous devons dans cette partie constituer un corpus diversifié en rassemblant des textes provenant d’une source donnée qui peuvent nous servir pour entraîner notre modèle, pour ce faire le choix a tombé sur les sites et les articles de presse vu que ce genre de contenu très répandue sur internet et on peut le trouver facilement, mais la difficulté qui reste est de pouvoir l’extraire et obtenir son résumé. C’est pour cette raison qu’on a utilisé deux méthodes.

* **Scrapping des données avec la bibliothèque newspaper :**

Cette méthode consiste à automatiser un script qui prend en argument l’URL de l’article souhaité et récupère les informations à partir des liens hypertextes présents dans le contenu HTML de la page. Il utilise les bibliothèques requests et BeautifulSoup pour obtenir le contenu de la page web et analyser le HTML. La fonction parcourt les liens, télécharge les articles associés à ces liens en utilisant une classe Article (non définie dans ce code), et stocke le contenu et le résumé de chaque article dans une liste. La récupération s'arrête lorsque le nombre maximum d'articles spécifié est atteint.

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Figure 1 : Fonction de Scrapping

Ensuite on va faire parcourir sur un ensemble d’articles dont on a cherché les liens manuellement et les mettre dans une liste comme dans la figure ci-dessous



Figure 2 : Liens des articles

Et finalement on va se servir de ce petit bout de code permettant de donner le résumé de chaque article et puis on va transformer les résultas en une dataframe se composant de deux colonnes, la colonne de texte « Content » ainsi que celle de résumé « Summary » .

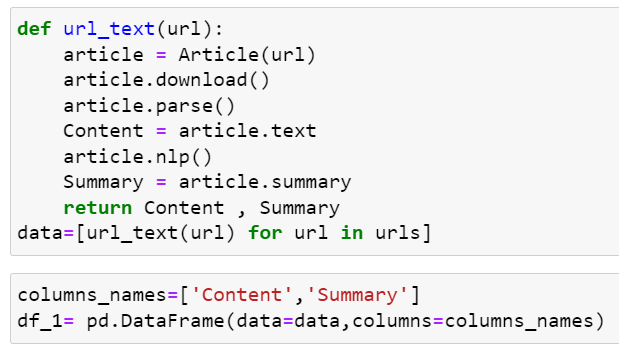


Figure 3 : Résumé et stockage des résultats

* **Scrapping avec un API donnant d’articles :**

Cette méthode qui repose sur l’usage d’un API dédié à la fourniture d'articles nous offre une alternative plus structurée et simplifiée pour obtenir des données. Plutôt que de parcourir manuellement les pages web et d'extraire les informations avec le code précédent, cette méthode fournit donc des articles prêts à l'emploi, il suffit juste d’avoir un script avec le lien et la clé de l’api Le script interagit avec l'API en envoyant des requêtes spécifiques, dans notre cas on a utilisé un l’api appelé newsapi.ai .

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Figure 4 : Script de connexion avec l'api newsapi

On a choisi deux sujets distincts dont on va extraire les articles qui y sont liés, autrement dit on doit préciser les mots clés à mettre dans les requêtes de l’api pour obtenir l’ensemble d’articles relatives à ces sujets, qui sont

« L’intelligence artificielle » et « Palestine Israel », et puis ont exécuté le code suivant pour enregistrer les résultats sous format de fichier csv.

Une image contenant texte, Police, capture d’écran, ligne

Description générée automatiquement

Figure 5 : Sauvegarde des fichiers de données

Ensuite après avoir obtenu les fichiers de données on va essayer d’ajouter une colonne contenant les résumés des articles parce que cette api par défaut ne donne pas les résumés ce qui fait, on doit essayer de l’ajouter nous-même, mais vu que les articles sont nombreux on a utilisé le modèl T5 pour nous donner les résumés et les ajouter par la suite au fichier des articles avec le code ce-dessous

Une image contenant texte, Police, capture d’écran

Description générée automatiquement

Figure 6 : Ajout des résumes

* **Enrichissement des données d’entrainement :**

Dans cette étape après avoir collecté des données à l’aide de scrapping et l’api, on s’est aperçue qu’on peut enrichir nos données de plus en utilisant des données qu’on a trouvé sur kaggle non seulement pour diversifier nos sources de données mais aussi parce que l’api nous donne un nombre limitée de requête à ne pas dépasser et que la méthode de la recherche de liens d’articles manuellement pour le scrapping prend du temps. C’est ainsi qu’on a ajouté ces données là qui étaient prêtes à l’ensemble de nos données d’entraînement.



Figure 7 : Données d'enrichissement

Et maintenant que tous nos données sont prêtes, il reste juste de les concaténer et les mettre dans un seul fichier de corpus ayant les Textes et les Résumés, comme le montre cette figure.

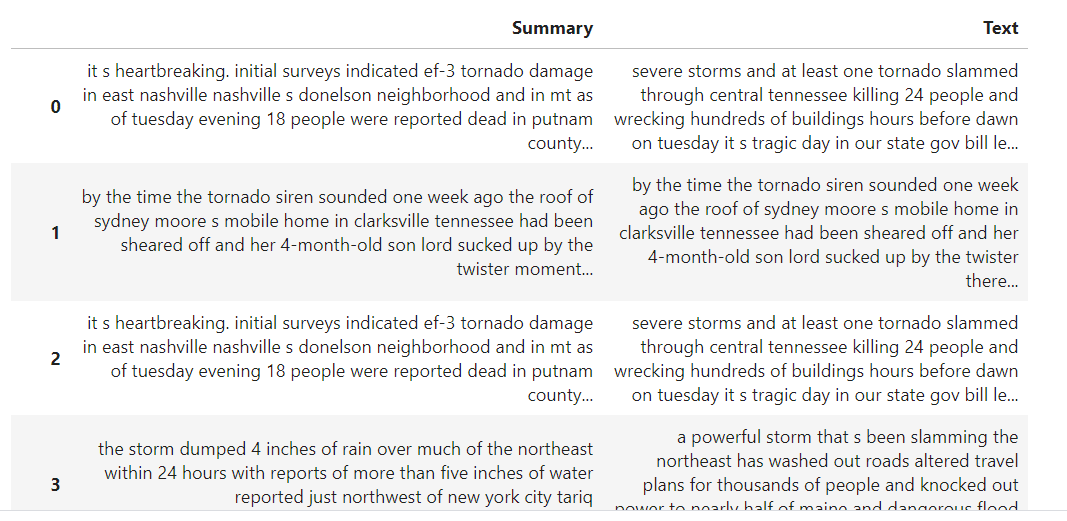


Figure 8 : Dataframe du corpus

Préparation des données :

* **Nettoyage des données :**

La fonction « **preprocess\_text »** dans l’image ci-dessous effectue une série de transformations visant à nettoyer et normaliser le texte d'entrée. Elle convertit le texte en minuscules, élimine les caractères indésirables tels que les tabulations et les retours à la ligne, supprime les guillemets doubles, et étend les contractions à leur forme complète. Elle gère également la ponctuation, les occurrences spécifiques de nombres, les espaces multiples, et remplace les URLs par leur nom de domaine, tous ces traitement-là donc, ont été incorporé dans une seule fonction qui se charge à faire tout cela afin d’adapter les données à l’entraînement.

Une image contenant texte, capture d’écran, document, Police

Description générée automatiquement

Figure 9 : Fonction de traitement de données

* **Traitement de corpus :**

Dans cette étape après avoir exécuté cette fonction de traitement sur les données on va avoir deux nouvelles colonnes qui contiennent les données traitées, c’est pour cette raison qu’on va se débarrasser des anciennes colonnes ainsi que les lignes vides avec le code suivant :

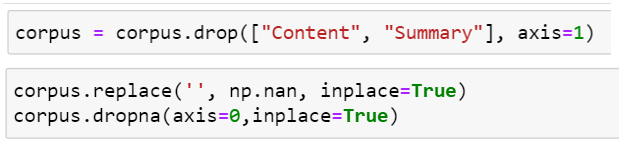


Figure 10 : Traitement corpus

Conclusion :

En concluant cette première étape, nous avons tracé un parcours à travers les méandres de la collecte et de la préparation des données, une phase fondatrice qui jette les bases de notre projet. La diversité des sources que nous avons explorées, des articles de presse aux API, a enrichi notre corpus de manière significative

# Création et évaluation des modèles

1. Introduction :

Dans cette étape cruciale de notre projet, nous avons développé et évalué trois modèles majeurs pour la génération automatique de résumés de texte : BART avec Fine-tuning, Pegasus avec Fine-tuning, et LSTM. Chacun de ces modèles a été sélectionné pour ses caractéristiques distinctes, et nous entreprenons une évaluation rigoureuse de leur performance sur notre corpus préparé

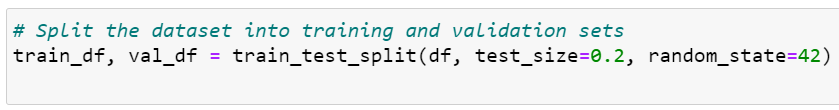
BART, en tant que modèle de transformer pré-entraîné, excelle dans la capture de la séquence bidirectionnelle, Pegasus, également basé sur une architecture transformer, se distingue par son attention sur la paraphrase, et LSTM, bien que plus ancien, conserve sa pertinence pour certains scénarios

1. Configuration des Modèles :

### Modèle Pegasus :

Pegasus est un modèle de traitement du langage naturel (NLP) développé par Google. Il appartient à la famille des modèles de transformer et afin de l’implémenter on doit passer par les étapes suivantes :

1. **Diviser les données en test et entraînement :**



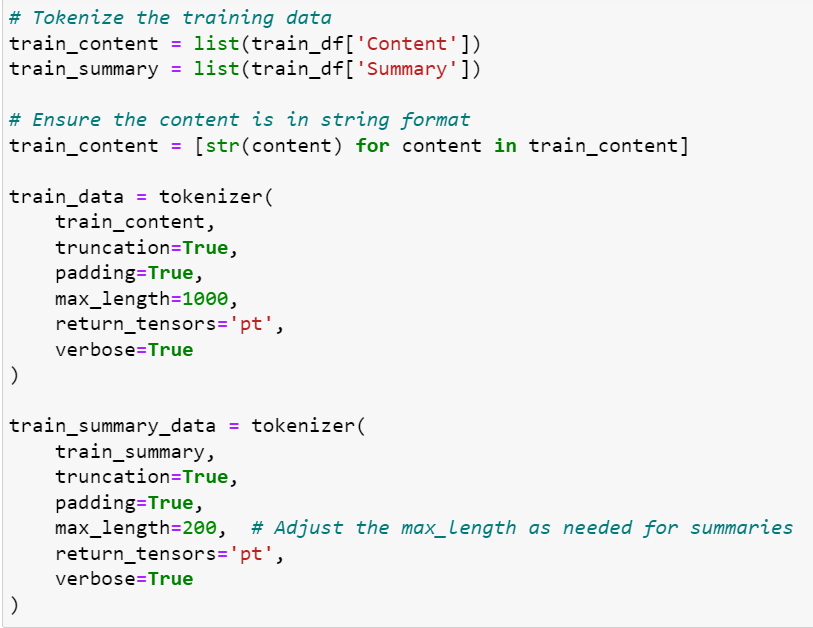
**Figure 11 : Split dataset**

1. **Initialiser le tokeniser de pegasus**



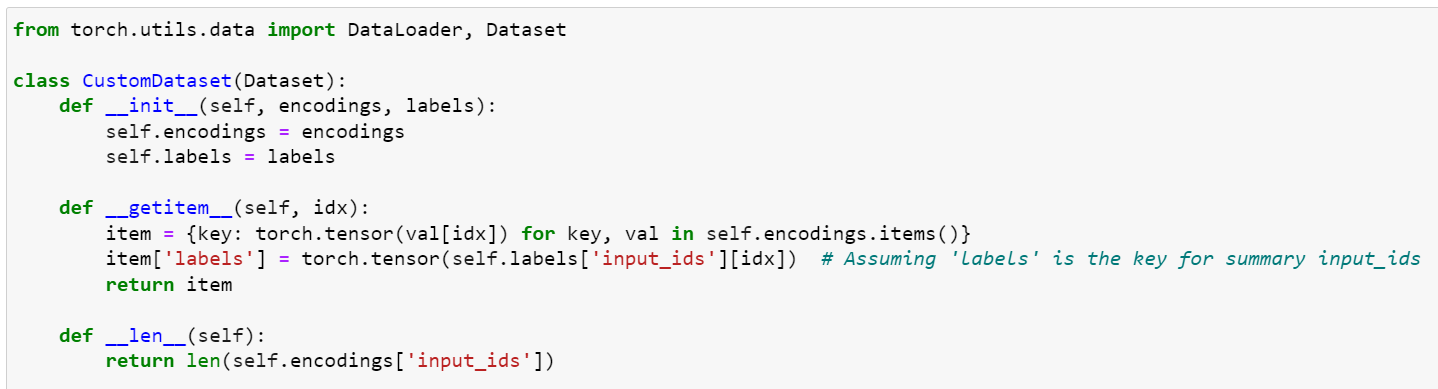
**Figure 12 : Pegasus Tokenizer**

1. **Tokeniser des données d’entrainement**



**Figure 13 : Pegasus tokenization**

1. **Définit une classe de dataset personnalisée pour gérer les données tokenisées et les labels, La création d'une classe de dataset personnalisée permet de personnaliser le chargement des données et d'assurer une intégration harmonieuse avec l'architecture spécifique de Pegasus**



**Figure 14 : Dataset personalisé**

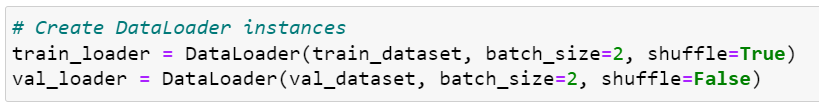
1. **Création des instances du dataset personnalisé *CustomDataset***

Une image contenant texte, Police, capture d’écran, ligne

Description générée automatiquement

**Figure 15 : Création de Custom Dataset**

1. **Création des instances DataLoader pour le chargement efficace des lots pendant l'entraînement et la validation. Ces instances DataLoader sont généralement utilisées pour itérer sur des lots de données pendant le processus d'entraînement ou de validation.**



**Figure 16 : Création de l'instance DataLoader**

1. **Initialisation du modèle de Pegasus :**



**Figure 17 : Initialisation de Pegasus**

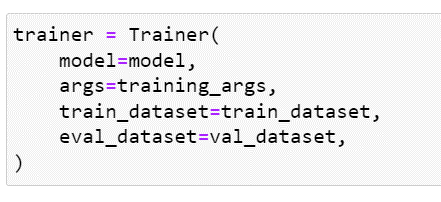
1. **Configuration des paramètres d'entraînement tels que le répertoire de sortie, la taille du lot, le nombre d'époques, etc.**

Une image contenant texte, Police, capture d’écran, nombre

Description générée automatiquement

**Figure 18 : Paramètres d'entrainement Pegasus**

1. **Initialise le Trainer avec le modèle, les arguments d'entraînement et les datasets.**



**Figure 19 : Lancer Pegasus Trainer**

1. **Sauvegarde du model fine-tuné :**

Une image contenant texte, Police, capture d’écran, ligne

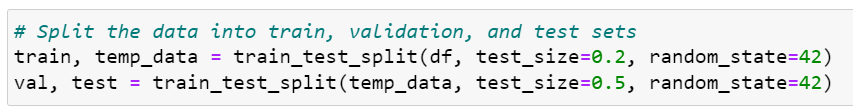
Description générée automatiquement

**Figure 20 : Sauvgard de modèle pegasus ajusté**

### Modèle Bart :

BART, ou Bidirectional and Auto-Regressive Transformers, est un modèle de traitement du langage naturel (NLP) qui appartient à la famille des transformers, afin de l’implémenter on doit procéder comme suit.

1. **Diviser les données en entraînnement et test.**



**Figure 21 : split data pour Bart**

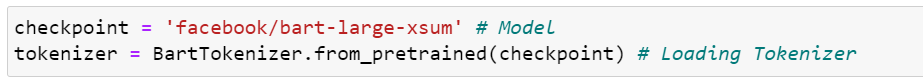
1. **Transformer les datasets en dataframes :**

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

**Figure 22 : Transformer Dataset en Dataframe**

1. **Configuration du modèle pré-entraîné BART de Facebook et son tokenizer associé, permettant ainsi de prétraiter les données et d'utiliser le modèle**



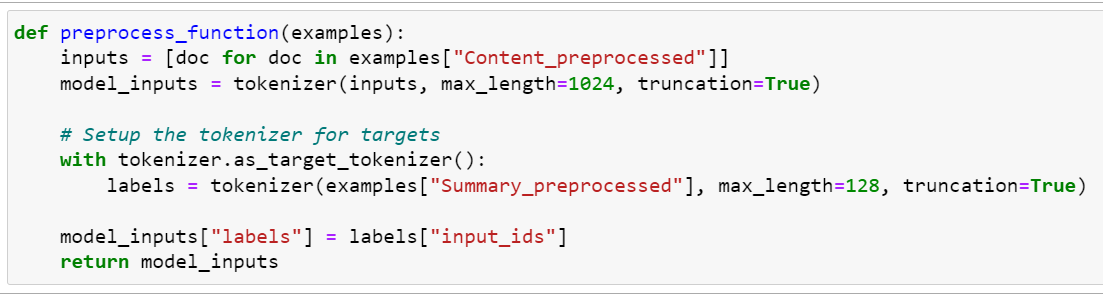
**Figure 23 : Configuration de modèle Bart**

1. **Création d’une instance du modèle BART pré-entraîné pour la génération conditionnelle de séquences.**



**Figure 24 : Instance de la classe BartConditionalGeneration**

1. **Cette fonction sert à tokeniser en utilisant un tokenizer spécifique avec des contraintes de longueur**



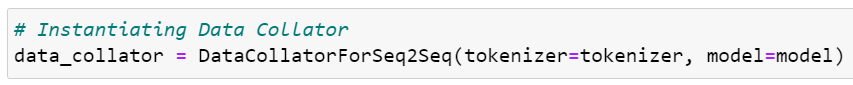
**Figure 25 : Tokenisation de bart**

1. **Ce code applique la fonction de prétraitement preprocess function aux ensembles de données d'entraînement, de test et de validation.**



**Figure 26 : Appliquer la fonction preprocess**

1. **le *DataCollator* est conçu pour prendre des exemples de données dans le format attendu par un modèle de séquence à séquence, tel que BART, et préparer ces données pour l'entraînement.**



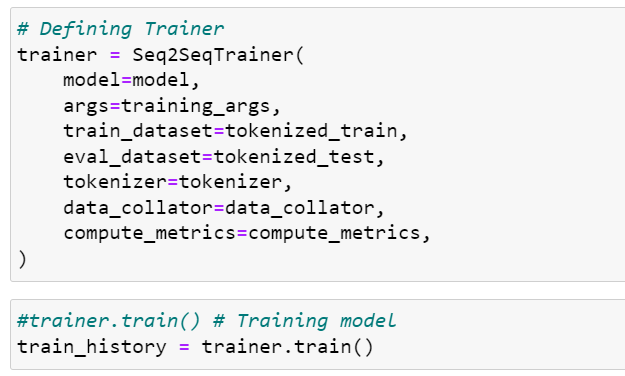
**Figure 27 : Data Collator Bart**

1. **Définition des paramètres d'entraînement pour le modèle**



**Figure 28 : Parametres d'entrainement de Bart**

1. **Entraînement du modèle pour le fine-tuning**



**Figure 29 : Lancement de Bart Trainer**

### Modèle LSTM :

Les LSTM, ou "Long Short-Term Memory", sont un type de réseau de neurones récurrents (RNN) conçus pour traiter les problèmes liés à la dépendance à long terme dans la séquence de données. Pour l’implementer on doit procéder comme suit.

1. **Séparation des données en train et test :**



**Figure 30 : LSTM split**

1. **Tokenisation :**

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

**Figure 31 : LSTM tokenization**

Une image contenant texte, Police, capture d’écran, blanc

Description générée automatiquement

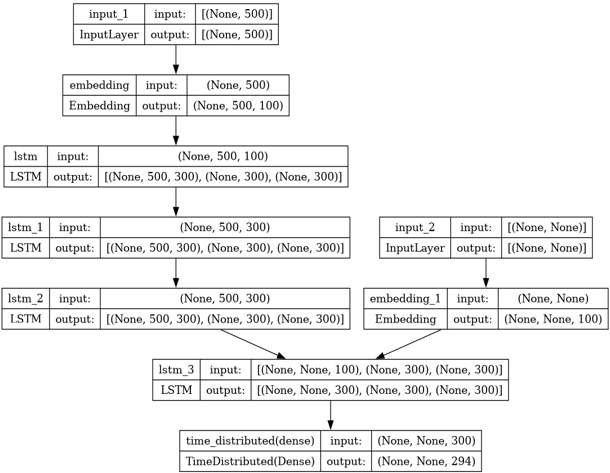
**Figure 32 : texts to sequence**

1. **Modele LSTM :**



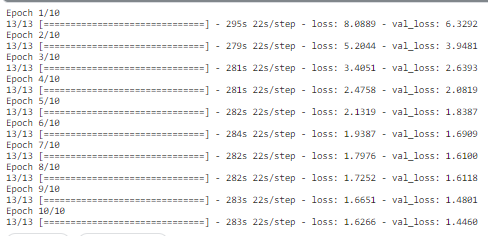
**Figure 33 : Architecture de LSTM**

1. **Architecture des modèles :**



**Figure 34 : Plot de Lstm Architecture**

1. **Entrainement du modèle**

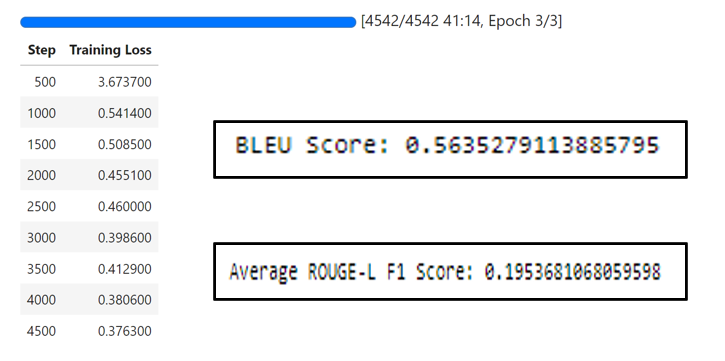


**Figure 35 : Entrainement de LSTM**

1. Evaluation des résultats :

### Modèle Pegasus :

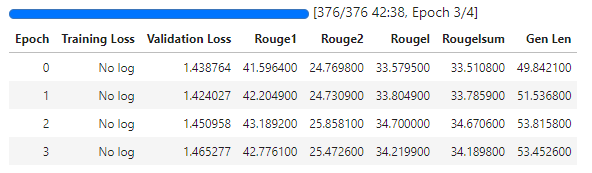
Le modèle pegasus a donné les résultats suivants pour les scores BLEU et ROUGE



**Figure 36: Evaluation Pegasus**

### Modèle BART :

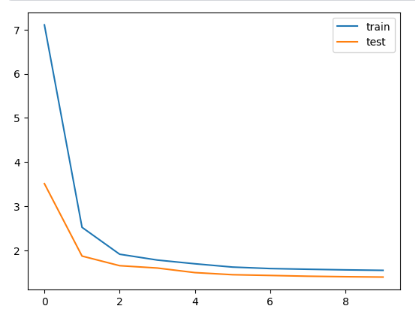
Le modèle BART nous a donné **des résultats suivants** :



**Figure 37 : Evaluation Bart**

### Modèle LSTM :

Le modèle LSTM nous a donné le résulat suivant



**Figure 38 : Evaluation LSTM**

1. Conclusion :

Dans ce chapitre on a découvert les différents résultats que nos modèles ont donné et on enregitré le modèle BART vu le fait qu’il a donné de bon résultats.

# Amélioration des performances :

1. Introduction :

Pour l’amélioration des performances on a opté pour l’ajout des fonctionnalités à nos modèles afin de booster leurs compétences.

1. LSTM : Attention Layer

Pour l’amélioration éviter le problème de convergence de modèle LSTM on a ajouté une couche d’attention comme le montre la figure suivante :

Une image contenant texte, diagramme, Police, Parallèle

Description générée automatiquement

**Figure 39 : LSTM avec Attention Layer**

Mais malgré cela le modèle se converge vers l’output : the the the the the the ….

1. BART : NER

Afin d’améliorer les performances de modèle Bart, on a opter par l’ajout de la fonctionnalité de NER en utilisant la bibliothèque standard Spacy .

A screen shot of a computer code

Description automatically generated

**Figure 40 : Fonction d'extraction des entités**

Ensuite on ajoute un input à notre architecture qui est l’input des entités généré par le module Spacy basique.

A close-up of a computer screen

Description automatically generated

**Figure 41 : Fonction d'ajout de NER au Bart**

On a remarqué que le modèle a été amélioré malgré le nombre limité de jeu de données de l’entrainement.

Une image contenant texte, capture d’écran, logiciel, affichage

Description générée automatiquement

**Figure 42 : Bart avec NER trainer**

Ainsi que le score BLEU a été amelioré :

Une image contenant texte, Police, ligne, capture d’écran

Description générée automatiquement

**Figure 43 : BLEU score de modèle Bart amelioré**

1. Conclusion :

On remarque que l’ajout d’autre fonctionnalités aux modèles peuvent amélioré les résultats comme il peut ne pas fonctionner, dans notre cas la fonctionnalité de NER qui est met en input au modèle Bart a aider notre modèle pour s’améliorer

# Déploiement du modèle

1. Introduction

À présent que nous avons parcouru le chemin exigeant de la collecte, de la préparation des données, de la création et de l'évaluation des modèles, il est temps de passer à l'étape passionnante du déploiement du modèle qui marque la concrétisation de nos efforts dans une application pratique et accessible à tous.

Notre objectif est de créer une interface utilisateur avec Flask conviviale qui permettra aux utilisateurs de bénéficier des capacités de notre modèle de résumé de texte. Que ce soit sur une plateforme mobile ou web, nous visons à fournir une expérience fluide et intuitive, où la complexité algorithmique se dissimule derrière une interface ergonomique.

Note application suit une architecture micro-service, ce qui signifie qu'elle est composée de plusieurs services indépendants qui interagissent entre eux pour fournir une fonctionnalité globale. Dans votre cas, les deux principaux services sont le frontend écrit en JavaScript (utilisant Flask pour le backend en Python) et le modèle bart qui est entrainé sur notre corpus scrapé pour la génération de résumés.

Une image contenant texte, capture d’écran, Rectangle, Police

Description générée automatiquement

Figure 44 : Architecture de l'application

1. Frontend (JavaScript - home.js):

* **Interface Utilisateur :**

L'interface utilisateur est construite en utilisant HTML et stylée avec CSS. Elle permet aux utilisateurs de saisir du texte dans une zone de texte et de choisir entre deux onglets, "Paragraph" et "NER".

Une image contenant texte, capture d’écran, logiciel, Système d’exploitation

Description générée automatiquement

Figure 45 : Interface de l'application

* **Asynchrone Traitement :**

Lorsque l'utilisateur clique sur le bouton "Generate Summary", le code JavaScript réagit à cet événement et Le contenu de la zone de texte est extrait et envoyé de manière asynchrone au backend Flask pour générer un résumé.

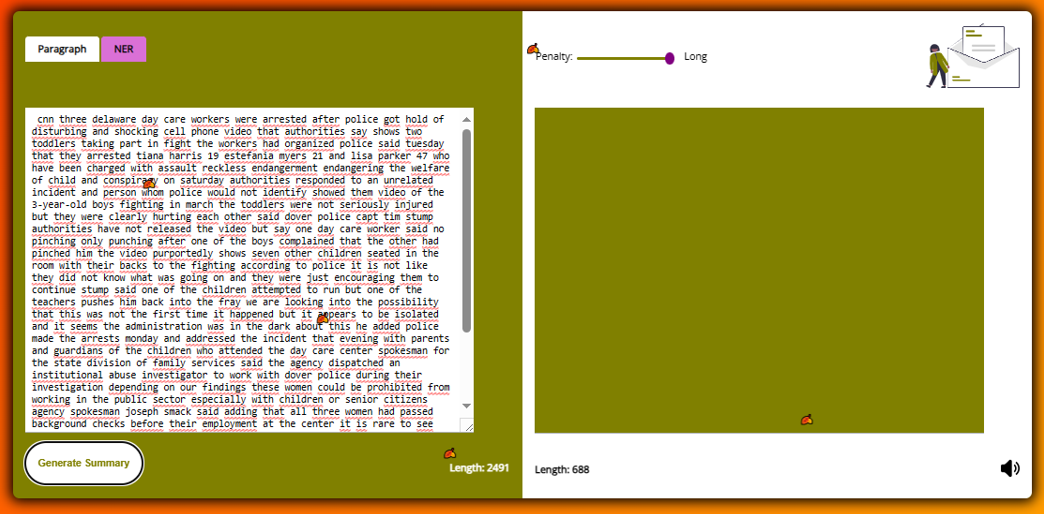


Figure 46 : Button generate summary

* **Affichage du Résumé :**

Une fois que le backend a généré le résumé, la réponse est récupérée et le résumé est affiché dans l'interface utilisateur.

Une image contenant texte, capture d’écran, logiciel, Système d’exploitation

Description générée automatiquement

Figure 47 : Affichage de Résume

* **Speech Synthesis :**

Les utilisateurs peuvent également écouter le résumé généré en utilisant la fonction de synthèse vocale de l'API Web Speech en cliquant sur le Button

en bas à droite de la page.

1. Backend (Python – app.py) :

* **Serveur Flask :**

Flask est utilisé comme framework web pour le backend. Il gère les routes, les requêtes HTTP, et interagit avec le modèle de génération de résumés.

Une image contenant texte, capture d’écran, logiciel, Police

Description générée automatiquement

Figure 48 : Generation du resume

Ce code permet d’envoyer la requête pour obtenir le résumé

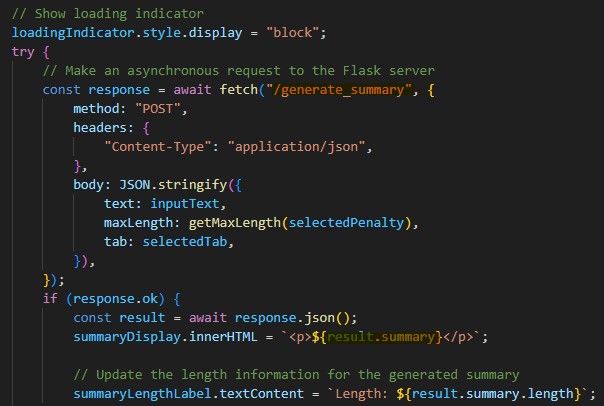


Figure 49 : Summary request

* **Modèle ESI\_Summerizer (bart fine tuned) :**

Le modèle BART (BartForConditionalGeneration) est chargé depuis le backend à l'aide de la bibliothèque Transformers, se situe dans le dossier Models/

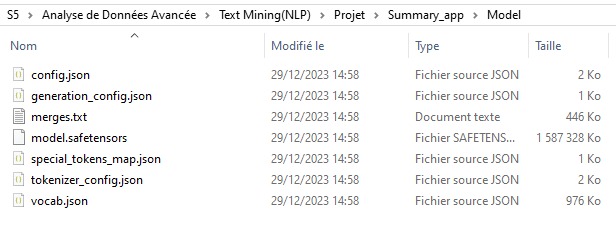


Figure 50 : Model

* **Réponse JSON :**

Le backend renvoie le résumé généré au format JSON en réponse à la requête du frontend.

1. Conclusion :

En résumé, l'application utilise un frontend interactif en JavaScript qui communique de manière asynchrone avec un backend Flask en Python. Le backend utilise un modèle Bart pour générer des résumés, et la communication entre les deux se fait via des requêtes HTTP. L'architecture micro-service offre une modularité et une scalabilité, permettant d'ajouter facilement de nouvelles fonctionnalités ou services à l'avenir

# Conclusion :

En conclusion, cette exploration de l'application de text summarization, alimentée par un modèle de deep learning formé sur des données d'articles de presse, a démontré la pertinence et le potentiel significatif de cette technologie dans le domaine de la gestion et garantir la transparence de l'information.

La capacité à extraire efficacement les éléments clés d'un texte, tout en préservant son sens original, offre des avantages considérables en termes de gain de temps et de facilitation de la compréhension. Cependant, des défis subsistent, tels que la nécessité de perfectionner davantage le modèle pour des contextes spécifiques et la vigilance quant à la préservation de la précision et de l'intégrité du contenu. En dépit de ces défis, l'application représente une avancée significative dans la quête d'outils intelligents visant à rendre l'information plus accessible et digestible.

Ce rapport a jeté les bases d'une compréhension approfondie de cette application, offrant une vision claire de ses avantages actuels et de son potentiel d'amélioration future dans le paysage dynamique de la technologie de l'information.