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

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.

I. Collecte et préparation des données

1. 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.

2. 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.

```python
def scrape_news(url, max_articles=None):
    response = requests.get(url)
    soup = BeautifulSoup(response.content, 'html.parser')
    articles = []
    for link in soup.find_all('a', href=True):
        if max_articles and len(articles) >= max_articles:
            break
        article_url = link['href']
        if article_url.startswith(('http://', 'https://')):
            try:
                article = Article(article_url)
                article.download()
                article.parse()
                content = article.text
                summary = article.summary
                articles.append({'Content': content, 'Summary': summary})
            except Exception as e:
                print(f"Error processing {article_url}: {e}")
    return articles
```

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

```
urls=["https://edition.cnn.com/2020/03/03/weather/nashville-tornado-tuesday/index.html",
     "https://edition.cnn.com/2023/12/16/us/tennessee-tornado-baby-found-alive/index.html",
      "https://edition.cnn.com/2020/03/03/weather/nashville-tornado-tuesday/index.html",
      "https://edition.cnn.com/2023/12/17/weather/northeast-storms-flooding-travel-monday/index.html",
      "https://edition.cnn.com/travel/gallery/modern-ships-sails-spc-intl/index.html",
      "https://edition.cnn.com/2023/11/29/africa/nigeria-women-health-risks-as-equals-intl-cmda/index.html",
      "https://edition.cnn.com/2023/12/18/africa/egypt-sisi-election-intl/index.html",
      "https://edition.cnn.com/2023/12/14/europe/hamas-leader-turkey-october-7-erdogan-intl/index.html",
      "https://edition.cnn.com/2023/11/30/politics/biden-administration-israel-war-hamas/index.html",
      "https://edition.cnn.com/2023/12/08/politics/post-war-gaza-plans/index.html",
      "https://edition.cnn.com/2023/12/07/opinions/israel-hamas-gaza-media-press-prusher/index.html",
      "https://edition.cnn.com/middleeast/live-news/israel-hamas-war-gaza-news-12-17-23/index.html",
      "https://edition.cnn.com/2023/11/08/americas/demonstrators-killed-mining-protests-panama/index.html",
      "https://edition.cnn.com/2023/12/08/middleeast/israel-un-diplomatic-showdown-hamas-war-mime-intl/index.html",
      "https://edition.cnn.com/2023/01/19/middleeast/iran-man-kills-wife-sentenced-intl/index.html?dicbo=v2-PT5ggt6&iid=ob_locked
      "https://edition.cnn.com/2023/12/05/china/mayday-lip-sync-investigation-china-intl-hnk/index.html",
      "https://edition.cnn.com/2023/12/08/asia/taiwan-intelligence-china-leaders-meeting-election-interference-intl-hnk/index.htm
      "https://edition.cnn.com/2023/12/15/economy/china-economy-data-november-hnk-intl/index.html",
      "https://edition.cnn.com/2023/12/18/tech/crypto-super-pac/index.html",
      "https://edition.cnn.com/2023/12/18/business/railroads-southern-border/index.html",
      "https://edition.cnn.com/2023/12/18/tech/apple-halt-sales-apple-watches/index.html",
      "https://edition.cnn.com/2023/12/12/media/elon-musk-x-conspiratorial-rabbit-hole/index.html",
      "https://edition.cnn.com/2023/12/18/business/trevor-milton-nikola-sentenced/index.html"
     ]
```

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 » .

```
def url_text(url):
    article = Article(url)
    article.download()
    article.parse()
    Content = article.text
    article.nlp()
    Summary = article.summary
    return Content , Summary
data=[url_text(url) for url in urls]
```

```
columns_names=['Content','Summary']
df_1= pd.DataFrame(data=data,columns=columns_names)
```

FIGURE 3 : RESUME ET STOCKAGE DES RESULTATS

- **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 .

```python
er = EventRegistry(apiKey="be8fb320-15de-4f70-897c-73febb47a3f9")
# spécifier l'URI de la location
usUri = er.getLocationUri("USA")

q = QueryArticlesIter(
    keywords=QueryItems.OR(["Palestine","Israel"]),
    minSentiment=0.4,
    sourceLocationUri=usUri,
    dataType=["news", "blog"]
)
# Liste pour stocker les articles
articles_list = []

# avoir en max 500 articles ou blog .
for art in q.execQuery(er, sortBy="date", maxItems=500):
    articles_list.append(art)
```

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.

```python
output_file_path = "Data/Scraped_Data/artificial_intelligence.json"
with open(output_file_path, 'w', encoding='utf-8') as json_file:
    json.dump(articles_list, json_file, ensure_ascii=False, indent=4)

print(f"Articles saved to {output_file_path}")
```

FIGURE 5 : SAUVEGARDE DES FICHIERS DE DONNEES

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

```python
from transformers import T5Tokenizer, T5ForConditionalGeneration
def generate_summary(text):
    model_name = "t5-small"
    tokenizer = T5Tokenizer.from_pretrained(model_name)
    model = T5ForConditionalGeneration.from_pretrained(model_name)
    inputs = tokenizer("summarize: " + text, return_tensors="pt", max_length=512, truncation=True)
    summary_ids = model.generate(inputs['input_ids'], max_length=150, length_penalty=2.0, num_beams=4, early_stopping=True)
    summary = tokenizer.decode(summary_ids[0], skip_special_tokens=True)
    return summary
```

FIGURE 6 : AJOUT DES RESUMES

- **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.

```
df_3 = pd.read_excel("Enrechissement_Data/df_3.xlsx")
```

FIGURE 7 : DONNEES 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.

	Summary	Text
0	it s heartbreaking. initial surveys indicated ef-3 tornado damage in east nashville nashville s donelson neighborhood and in mt as of tuesday evening 18 people were reported dead in putnam county...	severe storms and at least one tornado slammed through central tennessee killing 24 people and wrecking hundreds of buildings hours before dawn on tuesday it s tragic day in our state gov bill le...
1	by the time the tornado siren sounded one week ago the roof of sydney moore s mobile home in clarksville tennessee had been sheared off and her 4-month-old son lord sucked up by the twister moment...	by the time the tornado siren sounded one week ago the roof of sydney moore s mobile home in clarksville tennessee had been sheared off and her 4-month-old son lord sucked up by the twister there...
2	it s heartbreaking. initial surveys indicated ef-3 tornado damage in east nashville nashville s donelson neighborhood and in mt as of tuesday evening 18 people were reported dead in putnam county...	severe storms and at least one tornado slammed through central tennessee killing 24 people and wrecking hundreds of buildings hours before dawn on tuesday it s tragic day in our state gov bill le...
3	the storm dumped 4 inches of rain over much of the northeast within 24 hours with reports of more than five inches of water reported just northwest of new york city tariq	a powerful storm that s been slamming the northeast has washed out roads altered travel plans for thousands of people and knocked out power to nearly half of maine and dangerous flood

FIGURE 8 : DATAFRAME DU CORPUS

3. 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.

```python
def preprocess_text(text):
    text = str(text)
    text = text.lower()
    # Supprimer les tabulations, les retours à la ligne et les retour chariot.
    text = re.sub("(\\t)", ' ', str(text))
    text = re.sub("(\\r)", ' ', str(text))
    text = re.sub("(\\n)", ' ', str(text))
    # Supprimer les guillemets doubles du texte.
    text = re.sub('"', '', text)
    # Remplacer les contractions par leur forme étendue
    text = ' '.join([contraction_mapping[t] if t in contraction_mapping else t for t in text.split(" ")])
    # Supprimer les possessifs "'s" des mots.
    text = re.sub(r"'s\b","",text)
    # Supprimer _ s'il se produit plus d'une fois consécutivement
    text = re.sub("(__+)", ' ', str(text))
    # Supprimer - s'il se produit plus d'une fois consécutivement
    text = re.sub("(--+)", ' ', str(text))
    # Supprimer ~ s'il se produit plus d'une fois consécutivement
    text = re.sub("(~~+)", ' ', str(text))
    # Supprimer + s'il se produit plus d'une fois consécutivement
    text = re.sub("(\+\++)", ' ', str(text))
    # Supprimer . s'il se produit plus d'une fois consécutivement
    text = re.sub("(\.\.+)", ' ', str(text))
    # Supprimer les caractères - <>()|&©ø\[\]\'\",;?~*!
    text = re.sub(r"[<>()|&©ø\[\]\'\",;?~*!]", ' ', str(text))
    # Supprimer mailto:
    text = re.sub("(mailto:)", ' ', str(text))
    # Supprimer \x9* dans le texte
    text = re.sub(r"(\\x9\d)", ' ', str(text))
    # Remplacer les numéros INC par INC_NUM
    text = re.sub("([iI][nN][cC]\d+)", 'INC_NUM', str(text))
    # Remplacer CM# et CHG# par CM_NUM
    text = re.sub("([cC][mM]\d+)|([cC][hH][gG]\d+)", 'CM_NUM', str(text))
    # Supprimer la ponctuation à la fin d'un mot
    text = re.sub("(\.\s+)", ' ', str(text))
    text = re.sub("(\-\s+)", ' ', str(text))
    text = re.sub("(\:\s+)", ' ', str(text))
    # Remplacer toute URL par le nom de domaine uniquement
    try:
        url = re.search(r'((https*:\/*)([^\/\s]+))(.[^\s]+)', str(text))
        repl_url = url.group(3)
        text = re.sub(r'((https*:\/*)([^\/\s]+))(.[^\s]+)', repl_url, str(text))
    except:
        pass
    # Supprimer les espaces multiples
    text = re.sub("(\s+)", ' ', str(text))
    # Supprimer le caractère unique entre deux espaces
    text = re.sub("(\s+.\s+)", ' ', str(text))
    return text
```

FIGURE 9 : FONCTION DE TRAITEMENT DE DONNEES

- **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 :

```python
corpus = corpus.drop(["Content", "Summary"], axis=1)
```

```python
corpus.replace('', np.nan, inplace=True)
corpus.dropna(axis=0,inplace=True)
```

FIGURE 10 : TRAITEMENT CORPUS

4. 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

II. 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

2. 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 :

a. Diviser les données en test et entraînement :

```
# Split the dataset into training and validation sets
train_df, val_df = train_test_split(df, test_size=0.2, random_state=42)
```

Figure 11 : Split dataset

b. Initialiser le tokeniser de pegasus

```
# Initialize the PEGASUS tokenizer
tokenizer = PegasusTokenizer.from_pretrained('google/pegasus-large')
```

Figure 12 : Pegasus Tokenizer

c. Tokeniser des données d'entrainement

```python
# Tokenize the training data
train_content = list(train_df['Content'])
train_summary = list(train_df['Summary'])

# Ensure the content is in string format
train_content = [str(content) for content in train_content]

train_data = tokenizer(
    train_content,
    truncation=True,
    padding=True,
    max_length=1000,
    return_tensors='pt',
    verbose=True
)

train_summary_data = tokenizer(
    train_summary,
    truncation=True,
    padding=True,
    max_length=200,  # Adjust the max_length as needed for summaries
    return_tensors='pt',
    verbose=True
)
```

Figure 13 : Pegasus tokenization

d. 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

```python
from torch.utils.data import DataLoader, Dataset

class CustomDataset(Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
        item['labels'] = torch.tensor(self.labels['input_ids'][idx])  # Assuming 'labels' is the key for summary input_ids
        return item

    def __len__(self):
        return len(self.encodings['input_ids'])
```

Figure 14 : Dataset personalisé

e. Création des instances du dataset personnalisé *CustomDataset*

```
# Create CustomDataset instances for both training and validation
train_dataset = CustomDataset(train_data, train_summary_data)
val_dataset = CustomDataset(val_data, val_summary_data)
```

Figure 15 : Création de Custom Dataset

f. **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.**

```
# Create DataLoader instances
train_loader = DataLoader(train_dataset, batch_size=2, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=2, shuffle=False)
```

Figure 16 : Création de l'instance DataLoader

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

```
model = PegasusForConditionalGeneration.from_pretrained('google/pegasus-large')
```

Figure 17 : Initialisation de Pegasus

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

```
training_args = TrainingArguments(
    output_dir='./results',
    per_device_train_batch_size=1,
    per_device_eval_batch_size=1,
    num_train_epochs=3,
    save_steps=10_000,
    save_total_limit=2,
)
```

Figure 18 : Paramètres d'entrainement Pegasus

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

```
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
)
```

Figure 19 : Lancer Pegasus Trainer

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

```
# Save the fine-tuned model
model.save_pretrained('/kaggle/working/models/fine_tuned_pegasus1')
```

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.

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

```
# Split the data into train, validation, and test sets
train, temp_data = train_test_split(df, test_size=0.2, random_state=42)
val, test = train_test_split(temp_data, test_size=0.5, random_state=42)
```

Figure 21 : split data pour Bart

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

```
# Transforming dataframes into datasets
train_ds = Dataset.from_pandas(train)
test_ds = Dataset.from_pandas(test)
val_ds = Dataset.from_pandas(val)

# Visualizing results
print(train_ds)
print('\n' * 2)
print(test_ds)
print('\n' * 2)
print(val_ds)
```

Figure 22 : Transformer Dataset en Dataframe

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

```
checkpoint = 'facebook/bart-large-xsum' # Model
tokenizer = BartTokenizer.from_pretrained(checkpoint) # Loading Tokenizer
```

Figure 23 : Configuration de modèle Bart

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

```
model = BartForConditionalGeneration.from_pretrained(checkpoint) # Loading Model
```

Figure 24 : Instance de la classe BartConditionalGeneration

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

```
def preprocess_function(examples):
    inputs = [doc for doc in examples["Content_preprocessed"]]
    model_inputs = tokenizer(inputs, max_length=1024, truncation=True)

    # Setup the tokenizer for targets
    with tokenizer.as_target_tokenizer():
        labels = tokenizer(examples["Summary_preprocessed"], max_length=128, truncation=True)

    model_inputs["labels"] = labels["input_ids"]
    return model_inputs
```

Figure 25 : Tokenisation de bart

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

```
# Applying preprocess_function to the datasets
tokenized_train = train_ds.map(preprocess_function, batched=True,
                               remove_columns=[ 'Content_preprocessed', 'Summary_preprocessed']) # Removing features

tokenized_test = test_ds.map(preprocess_function, batched=True,
                               remove_columns=[ 'Content_preprocessed', 'Summary_preprocessed']) # Removing features

tokenized_val = val_ds.map(preprocess_function, batched=True,
                               remove_columns=['Content_preprocessed', 'Summary_preprocessed']) # Removing features

# Printing results
print('\n' * 3)
print('Preprocessed Training Dataset:\n')
print(tokenized_train)
print('\n' * 2)
print('Preprocessed Test Dataset:\n')
print(tokenized_test)
print('\n' * 2)
print('Preprocessed Validation Dataset:\n')
print(tokenized_val)
```

Figure 26 : Appliquer la fonction preprocess

g. **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.**

```
# Instantiating Data Collator
data_collator = DataCollatorForSeq2Seq(tokenizer=tokenizer, model=model)
```

Figure 27 : Data Collator Bart

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

```
training_args = Seq2SeqTrainingArguments(
    output_dir = 'bart_samsum',
    evaluation_strategy = "epoch",
    save_strategy = 'epoch',
    load_best_model_at_end = True,
    metric_for_best_model = 'eval_loss',
    seed = seed,
    learning_rate=2e-5,
    per_device_train_batch_size=2,
    per_device_eval_batch_size=2,
    gradient_accumulation_steps=4,
    weight_decay=0.01,
    save_total_limit=1,
    num_train_epochs=4,
    predict_with_generate=True,
    fp16=True,
    report_to="none"
)
```

Figure 28 : Parametres d'entrainement de Bart

i. Entraînement du modèle pour le fine-tuning

```
# Defining Trainer
trainer = Seq2SeqTrainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_train,
    eval_dataset=tokenized_test,
    tokenizer=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
)
```

```
#trainer.train() # Training model
train_history = trainer.train()
```

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.

a. Séparation des données en train et test :

```
x_tr,x_val,y_tr,y_val=train_test_split(np.array(df['text']),np.array(df['summary']),test_size=0.1,random_state=0,shuffle=True)
```

Figure 30 : LSTM split

b. Tokenisation :

```
y_tokenizer = Tokenizer(num_words=tot_cnt-cnt)
y_tokenizer.fit_on_texts(list(y_tr))
```

Figure 31 : LSTM tokenization

```
y_tr_seq    =   y_tokenizer.texts_to_sequences(y_tr)
y_val_seq   =   y_tokenizer.texts_to_sequences(y_val)
```

Figure 32 : texts to sequence

c. Modele LSTM :

```
from keras import backend as K
K.clear_session()

latent_dim = 100
embedding_dim=50
# Encoder
encoder_inputs = Input(shape=(max_text_len,))
#embedding layer
enc_emb =  Embedding(x_voc, embedding_dim,trainable=True)(encoder_inputs)
#encoder lstm 1
encoder_lstm1 = LSTM(latent_dim,return_sequences=True,return_state=True,dropout=0.4,recurrent_dropout=0.4)
encoder_output1, state_h1, state_c1 = encoder_lstm1(enc_emb)
#encoder lstm 2
encoder_lstm2 = LSTM(latent_dim,return_sequences=True,return_state=True,dropout=0.4,recurrent_dropout=0.4)
encoder_output2, state_h2, state_c2 = encoder_lstm2(encoder_output1)
#encoder lstm 3
encoder_lstm3=LSTM(latent_dim, return_state=True, return_sequences=True,dropout=0.4,recurrent_dropout=0.4)
encoder_outputs, state_h, state_c= encoder_lstm3(encoder_output2)
# Set up the decoder, using `encoder_states` as initial state.
decoder_inputs = Input(shape=(None,))
#embedding layer
dec_emb_layer = Embedding(y_voc, embedding_dim,trainable=True)
dec_emb = dec_emb_layer(decoder_inputs)
# Decoder LSTM
decoder_lstm = LSTM(latent_dim, return_sequences=True, return_state=True,dropout=0.4,recurrent_dropout=0.2)
decoder_outputs,decoder_fwd_state, decoder_back_state = decoder_lstm(dec_emb,initial_state=[state_h, state_c])
#dense layer
decoder_dense =  TimeDistributed(Dense(y_voc, activation='softmax'))
decoder_outputs = decoder_dense(decoder_outputs)
# Define the model
model_lstm = Model([encoder_inputs, decoder_inputs], decoder_outputs)
model_lstm.summary()
```

Figure 33 : Architecture de LSTM

d. Architecture des modèles :

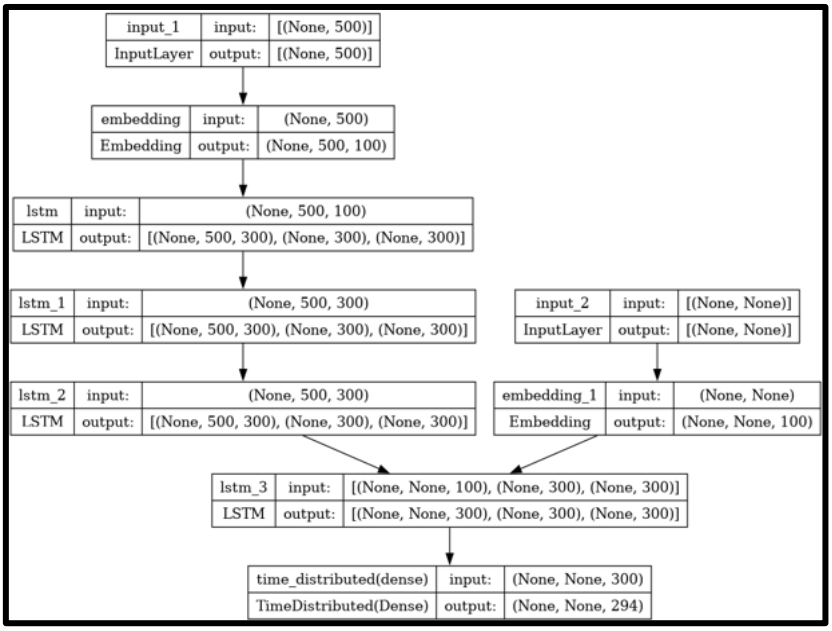

Figure 34 : Plot de Lstm Architecture

e. Entrainement du modèle

```
Epoch 1/10
13/13 [==============================] - 295s 22s/step - loss: 8.0889 - val_loss: 6.3292
Epoch 2/10
13/13 [==============================] - 279s 22s/step - loss: 5.2044 - val_loss: 3.9481
Epoch 3/10
13/13 [==============================] - 281s 22s/step - loss: 3.4051 - val_loss: 2.6393
Epoch 4/10
13/13 [==============================] - 281s 22s/step - loss: 2.4758 - val_loss: 2.0819
Epoch 5/10
13/13 [==============================] - 282s 22s/step - loss: 2.1319 - val_loss: 1.8387
Epoch 6/10
13/13 [==============================] - 284s 22s/step - loss: 1.9387 - val_loss: 1.6909
Epoch 7/10
13/13 [==============================] - 282s 22s/step - loss: 1.7976 - val_loss: 1.6100
Epoch 8/10
13/13 [==============================] - 282s 22s/step - loss: 1.7252 - val_loss: 1.6118
Epoch 9/10
13/13 [==============================] - 283s 22s/step - loss: 1.6651 - val_loss: 1.4801
Epoch 10/10
13/13 [==============================] - 283s 22s/step - loss: 1.6266 - val_loss: 1.4460
```

Figure 35 : Entrainement de LSTM

3. Evaluation des résultats :

- **Modèle Pegasus :**

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

[4542/4542 41:14, Epoch 3/3]

Step	Training Loss
500	3.673700
1000	0.541400
1500	0.508500
2000	0.455100
2500	0.460000
3000	0.398600
3500	0.412900
4000	0.380600
4500	0.376300

```
BLEU Score: 0.5635279113885795
```

```
Average ROUGE-L F1 Score: 0.1953681068059598
```

Figure 36: Evaluation Pegasus

- **Modèle BART :**

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

[376/376 42:38, Epoch 3/4]

Epoch	Training Loss	Validation Loss	Rouge1	Rouge2	Rougel	Rougelsum	Gen Len
0	No log	1.438764	41.596400	24.769800	33.579500	33.510800	49.842100
1	No log	1.424027	42.204900	24.730900	33.804900	33.785900	51.536800
2	No log	1.450958	43.189200	25.858100	34.700000	34.670600	53.815800
3	No log	1.465277	42.776100	25.472600	34.219900	34.189800	53.452600

Figure 37 : Evaluation Bart

- **Modèle LSTM :**

Le modèle LSTM nous a donné le résulat suivant

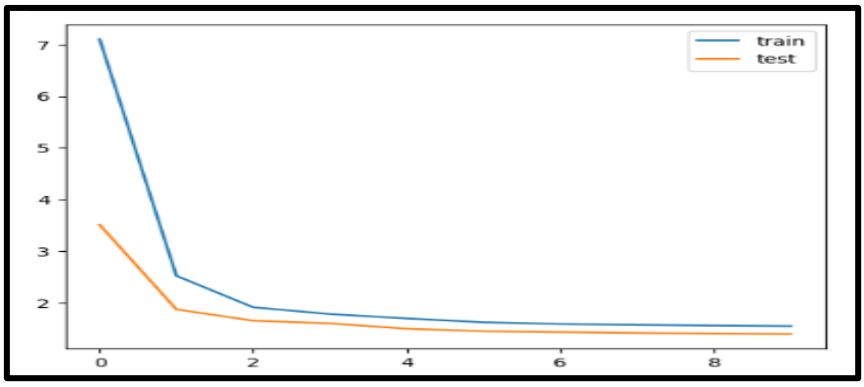

Figure 38 : Evaluation LSTM

4. 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.

III. 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.

2. 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 :

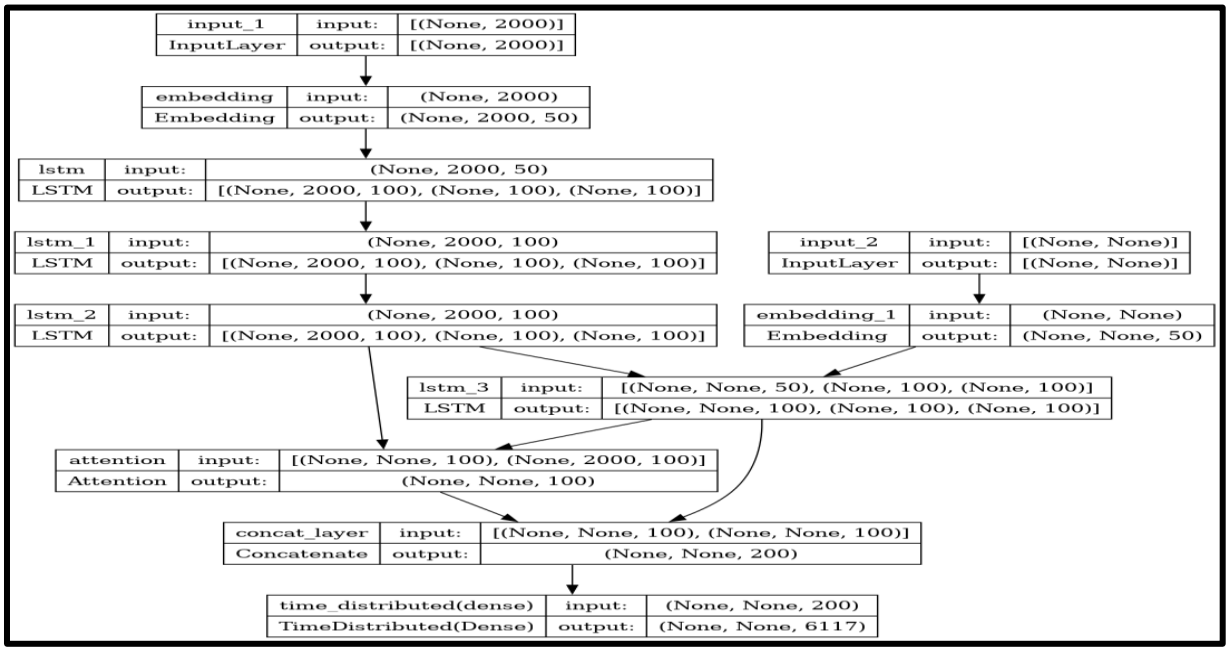

Figure 39 : LSTM avec Attention Layer

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

3. 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 .

```
def extract_entities(text):
    doc = nlp(text)
    entities = [ent.text for ent in doc.ents]
    return entities
# Assuming df is your DataFrame
df['Entities'] = df['Content'].apply(extract_entities)
```

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.

```
def preprocess_function(examples):
    inputs = [content + " " + entities for content, entities in zip(examples["Content"], examples["entities"])]
    model_inputs = tokenizer(inputs, max_length=1024, truncation=True)

    with tokenizer.as_target_tokenizer():
        labels = tokenizer(examples["Summary"], max_length=128, truncation=True)

    model_inputs["labels"] = labels["input_ids"]
    return model_inputs
```

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.

```
history = trainer.train()
```

[1701/1701 31:27, Epoch 8/9]

Epoch	Training Loss	Validation Loss
0	No log	3.210102
1	No log	3.324137
2	1.360500	3.270472
4	1.360500	3.506397
5	1.132700	3.594978
6	1.132700	3.670992
8	0.802400	3.785352

Figure 42 : Bart avec NER trainer

Ainsi que le score BLEU a été amelioré :

```
from nltk.translate import bleu_score
from rouge_score import rouge_scorer

bleu_score = bleu_score.corpus_bleu(references, predictions)
print("BLEU Score:", bleu_score)
```

```
BLEU Score: 0.5450022002835863
```

Figure 43 : BLEU score de modèle Bart amelioré

4. 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

IV. 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.

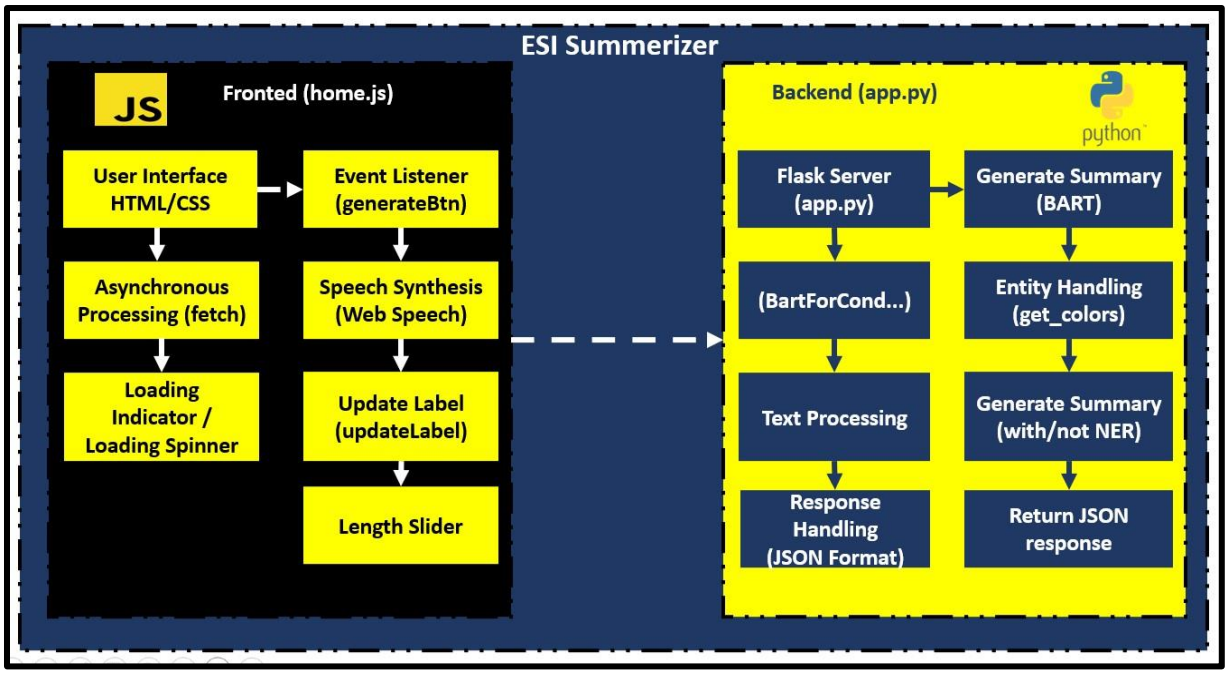

FIGURE 44 : ARCHITECTURE DE L'APPLICATION

2. 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".

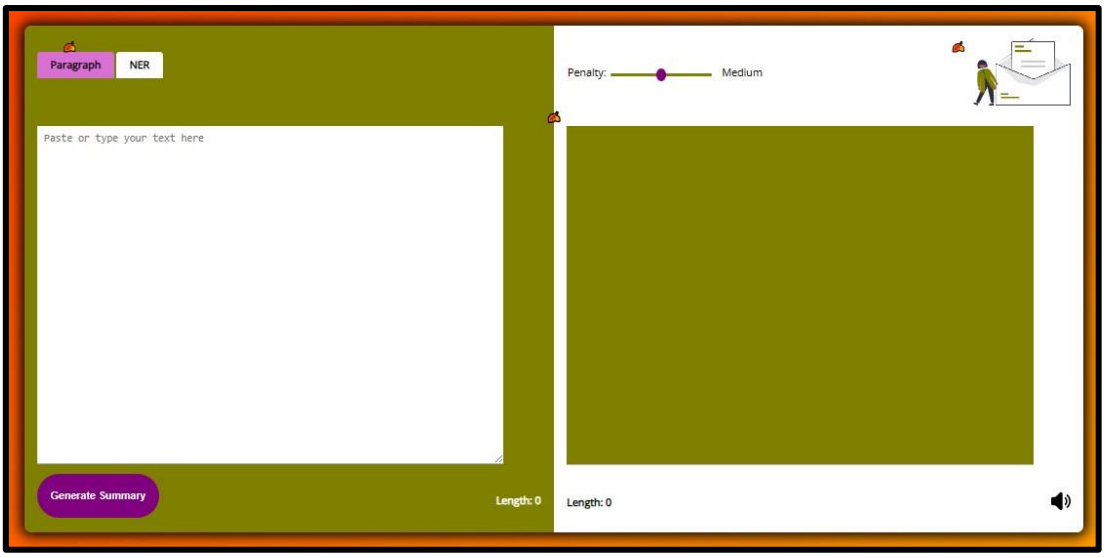

FIGURE 45 : INTERFACE DE L'APPLICATION

- **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.

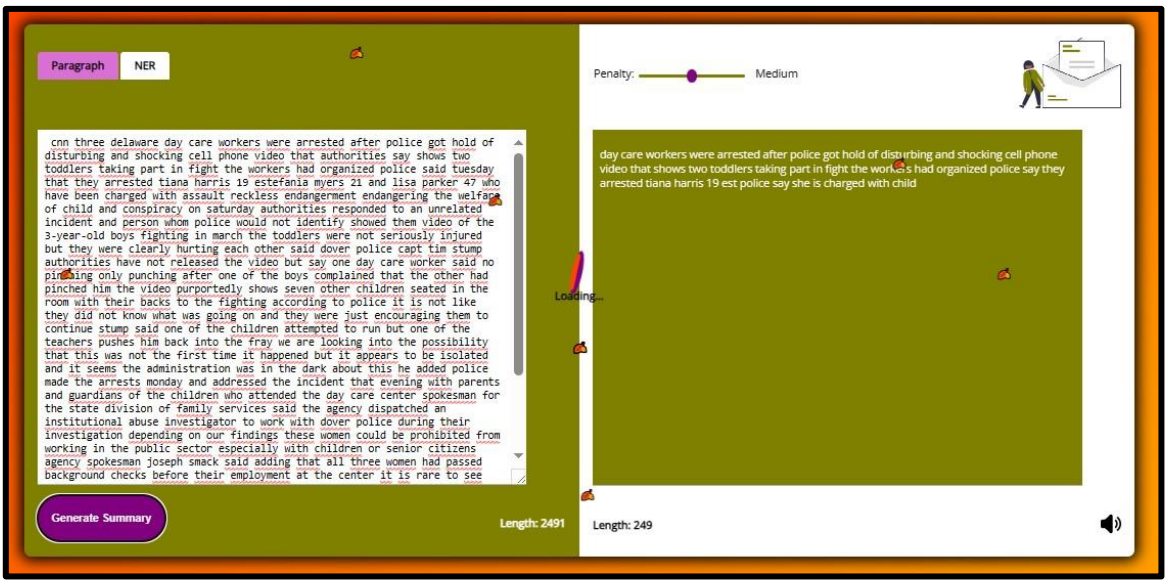

FIGURE 47 : AFFICHAGE DE RESUME

- **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.

3. 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.

```python
@app.route('/generate_summary', methods=['POST'])
def generate_summary():
    data = request.get_json()
    text = data['text']
    max_length = data['maxLength']
    selected_tab = data['tab']
    if selected_tab == 'paragraph':
        # Generate summary using the original function
        summary = generate_summary_function(text, model, tokenizer, max_length)
    elif selected_tab == 'ner':
        # Generate summary using the function with NER
        summary = generate_summary_function_with_ner(text, model, tokenizer, max_length)
    else:
        # Handle other tabs as needed
        summary = "Invalid tab selection"
    return jsonify({'summary': summary})
```

FIGURE 48 : GENERATION DU RESUME

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

```javascript
// Show loading indicator
loadingIndicator.style.display = "block";
try {
    // Make an asynchronous request to the Flask server
    const response = await fetch("/generate_summary", {
        method: "POST",
        headers: {
            "Content-Type": "application/json",
        },
        body: JSON.stringify({
            text: inputText,
            maxLength: getMaxLength(selectedPenalty),
            tab: selectedTab,
        }),
    });
    if (response.ok) {
        const result = await response.json();
        summaryDisplay.innerHTML = `<p>${result.summary}</p>`;

        // Update the length information for the generated summary
        summaryLengthLabel.textContent = `Length: ${result.summary.length}`;
```

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/

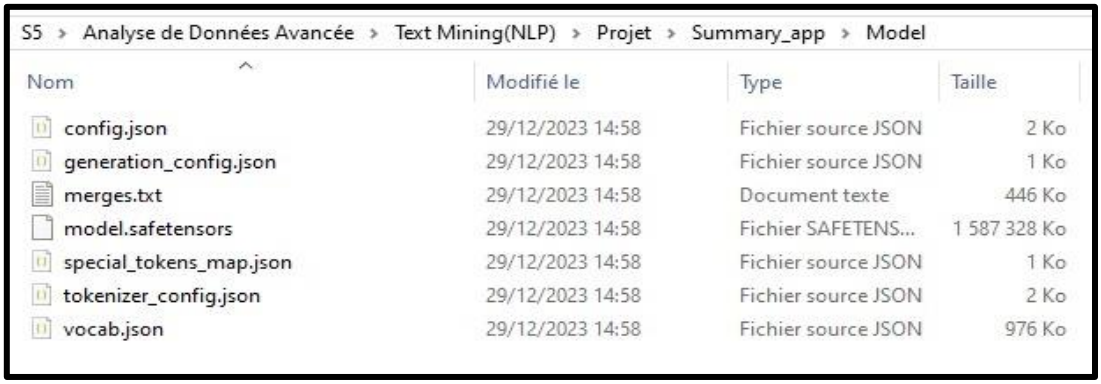

FIGURE 50 : MODEL

- **Réponse JSON :**

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

4. 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.