Rapport Projet d’outil de programmation à L’IA : Spam Detector

Etudiant : Thibaut LONGCHAMPS

Enseignant : Medhi AMMI

Détails du cheminement, des choix techniques et analyse des résultats

1. Section : Load raw data :

- **Concaténer les données brutes :** À l'origine, le dataset est fourni divisé en un ensemble d'entraînement et un ensemble de test sur le site de Hugging Face. J'ai choisi de concaténer ces ensembles pour obtenir un dataset complet, ce qui me permet d'appliquer le prétraitement en une seule étape. Cela me donne également la flexibilité de personnaliser la division des données selon mes besoins.

2. Section : EDA

- **Distribution du label :** Il est essentiel d’analyser l’équilibre dans la distribution du label. En effet la plupart des algorithmes d’apprentissage supervisés suppose une répartition égale entre les deux classes. Dans le cas d’un déséquilibre. Le modèle pourrait favoriser la classe majoritaire, car elle contribue davantage à minimiser l'erreur globale. Ainsi les métriques globales comme l’accuracy pourraient être trompeuses (dummy model). Ici, le label est équilibré, nous n’aurons pas besoin d’avoir recours à des techniques de data augmentation.

- **Preprocessing NLTK** : La librairie NLTK est un outil très puissant qui nous permet de tokenizer les données selon différentes échelles (caractères, mots, phrases). Ce qui facilite l’exploration et l’analyse des données (outliers, statistiques de distribution...)

- **Extraction des emoji :** Les séquences comme “\U0001F600-\U0001F64F” sont des encodages standard unicodes qui spécifient des plages de caractères correspondant à des emojis ou des symboles particuliers. Chaque emoji a un code unique (appelé code point) Sous un format “\U “suivi de 8 chiffres hexadécimaux, qui représentent un type d’emoji ou symbole. Ces unicodes alliés aux regex.compile permet de pouvoir manipuler les emojis et symbole de manière plus efficace, et est déterminant pour démontrer que la majorité des emojis se trouvent dans les textes classés comme spam. C'est une information très précieuse que nous devrons exploiter dans la construction de notre modèle.

3. Section : Data preprocessing, NLP

- **Gestions des valeurs aberrantes :** Ici, la méthode du z-score est utilisée (3 * std à la moyenne). Leur suppression est essentielle afin d'éviter les biais d'apprentissage (overfitting), les données bruitées (données non représentatives des comportements réelles), fausser la moyenne (asymétrie positive ou négative), augmentation les temps de calcules (padding démesuré)

- **Nettoyage des données :** Les expressions régulières (regex) sont un outil puissant qui, grâce à l'utilisation de métacaractères, permettent de transformer un texte brut en un format propre et structuré, afin de réaliser d'autres preprocessing (tokenisation...) et les transformer dans un format adapté à un modèle.

- **Spacy librairie pour le NLP :** nous utiliserons la bibliothèque spacy, qui nous permet d'effectuer diverses tâches de traitement linguistique. Cette bibliothèque nous permet de :
  - Tokeniser, C'est à dire découper un texte en sous-section et le placer dans une liste afin de faciliter les futurs traitements linguistiques
  - Charger un dictionnaire un anglais qui nous permettra de trier les tokens selon leur type grammatical
  - Supprimer les stops words, les mots qui n'ont pas d'intérêts informationnels
  - Lemmatiser, garder la racine des mots

  À ce stade les données sont nettoyées et détokenisées. La suite du preprocessing se fera grâce à la librairie tensorflow

- **Tensorflow librairie pour NLP :** Cette librairie va nous permettre :

- via la librairie keras de tokeniser chaque liste de notre dataset et grâce à l'argument num_words, définir la taille de notre vocabulaire (2000 mots les plus fréquents dans nos données). Un token "out of vocab" sera attribué à chaque token n'appartenant pas à ce dictionnaire (OOV ; out of vocab)

- Transformer les tokens en séquences de nombre entier en utilisant la clé pour chaque token défini dans notre dictionnaire. Certaines séquences sont dépourvues de mot appartenant au dictionnaire, ce qui renverra une liste vide. Leur suppression est indispensable. Des traitements similaires similaire seront effectués sur la colonne des emojis. Il faudra veiller à ce que les index des deux vocabulaires ne se chevauchent pas

- Harmoniser les tailles de séquence, en ajoutant un token supplémentaire dans le vocabulaire pour les 0 du padding. Les deux vocabulaires seront combinés pour le model avec une architecture unifiée

- **Train test split :** J'ai pris le parti de faire une répartition 80/20 et d'ajuster l'argument stratify sur le label, afin que les classes splitées restent équilibrées malgré le côté aléatoire de l'échantillonnage. Un ensemble de validation a aussi été défini avec les mêmes proportions. Cet ensemble nous permettra de prévenir l'overfitting en surveillant les métriques pendant l'entraînement pour chaque époch et surveiller l'epoch pendant laquelle le modèle atteindra la convergence.

4. Section : Unified architecture model

- **Combiner les ensembles de train, val et test :** Combiner les séquences encodées de texte et d'emojis pour chaque ligne des ensembles d'entraînement, de validation, et de test et les convertir en liste. Cette opération est utile pour préparer les données textuelles et emojis dans un format compatible avec les modèles de deep learning, et ainsi prendre en compte simultanément les informations textuelles et les emojis dans l'apprentissage.

- **Padding :** Après avoir combiné les séquences, il est nécessaire d'harmoniser leur taille. L'ensemble des séquences sera fixé sur la taille de 55. Pour toutes les séquences avec une taille initialement inférieur, des 0 seront ajoutées, pour les séquences initialement supérieures elles seront tronquées.

- **Taille des séquences :** J'ai fait le choix d'opter pour une taille de séquence maximale égal au 95 percentile. Cette approche permet d'éviter des remplissages (padding) excessivement longs pour les séquences qui n'en ont pas besoin. Ainsi, nous pouvons traiter la majorité des données de manière efficace tout en minimisant le remplissage inutile.

- **Tensor_slices** : Crée des datasets adaptés à tensorflow et permet de diviser les données en lots de taille batch_size (ici, 32). Les lots facilitent l'entraînement par mini-batch, optimisant la gestion des ressources mémoire et accélérant l'entraînement. Le .shuffle() sera exclusivement appliqué sur l'ensemble d'entraînement, ce qui évitera les patterns liés à l'ordre des données, ce qui améliore la capacité du modèle à généraliser.

- **Embedding** : chaque token sera en transformé en un vecteur de 128 valeurs numériques dans un espace de dimension 128. (shape input : (None, longueur séquence : 55, embedding : 128)

Architecture globale :

```
Input shape : (None, 55)
Shape after Embedding : (None, 55, 128)
Shape after LSTM bidirectionnel : (None, 128)
Shape after Dense (64 neurons) : (None, 64)
Shape after Dropout : (None, 64)
Shape after dense (1 neuron) : (None, 1)

Model: "functional"
```

Layer (type)	Output Shape	Param #	Connected to
input_combined (InputLayer)	(None, 55)	0	-
embedding (Embedding)	(None, 55, 128)	301,568	input_combined[0…
not_equal (NotEqual)	(None, 55)	0	input_combined[0…
bidirectional (Bidirectional)	(None, 128)	98,816	embedding[0][0], not_equal[0][0]
dense (Dense)	(None, 64)	8,256	bidirectional[0]…
dropout (Dropout)	(None, 64)	0	dense[0][0]
output (Dense)	(None, 1)	65	dropout[0][0]

```
Total params: 408,705 (1.56 MB)

Trainable params: 408,705 (1.56 MB)

Non-trainable params: 0 (0.00 B)
```

- **Input :** Une séquence de longueur fixe (max_len_combined_opti : taille de séquence réduite au 95eme percentile), qui correspond à la séquence combinée de texte et emojis après encodage et padding.

- **Embedding Layer :** Cette couche convertit les indices (entiers) en vecteurs denses de taille fixe (embedding_dim=128). L'argument mask_zero=True indique que les positions avec des zéros (padding) sont ignorées dans les calculs.

- **La couche LSTM bidirectionnelle :** traite la séquence dans les deux directions (avant et arrière). Les séquences textuelles ont souvent une dépendance contextuelle dans les deux sens. Le B_LSTM capture mieux ces relations contextuelles par rapport à un modèle simple ou unidirectionnel.

  Paramètres :
  - lstm_units=64 : Nombre de neurones dans chaque direction.
  - dropout=0.3 : Régularisation pour éviter l'overfitting en désactivant aléatoirement 30 % des neurones.
  - return_sequences=False : Ne retourne que la dernière sortie de la séquence (vecteur global de la séquence).

- **Dense layer :** Permet au modèle d'apprendre des relations non linéaires dans les caractéristiques extraites par le LSTM, et réduit la dimensionnalité tout en conservant les informations importantes.
  - Une couche dense avec 64 neurones et une activation Relu.

- **Dropout Layer :** Une couche de régularisation qui désactive aléatoirement 50 % des neurones pour éviter le surapprentissage.

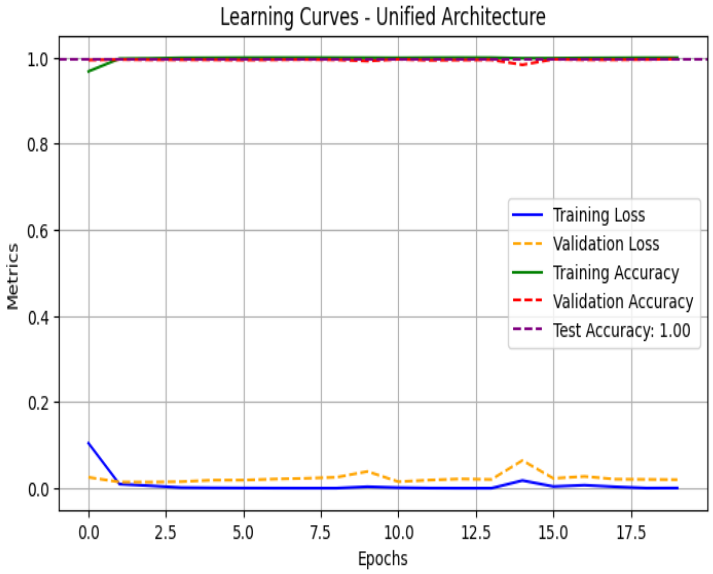
- **Sortie** : Une probabilité entre 0 et 1 (activation sigmoïde), pour effectuer une classification binaire.

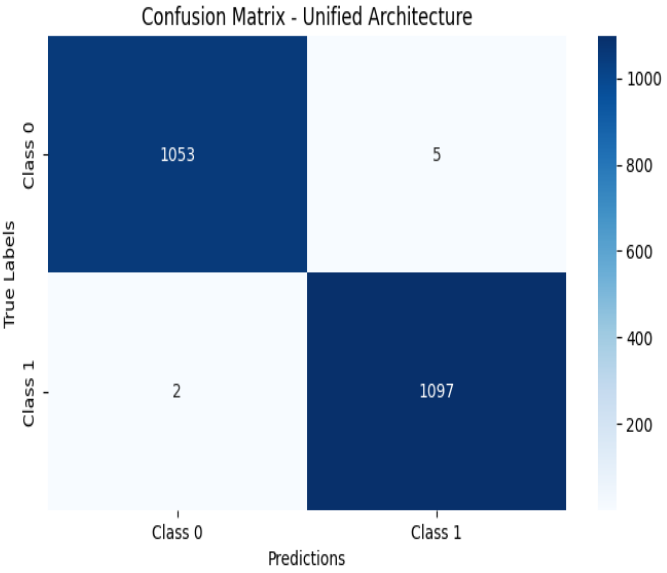
Compilation du modèle unifié

- **Optimiser Adam :** Permet d'ajuster dynamiquement le learning rate pour chaque paramètre

- **Loss :** Pendant la compilation nous définirons également la fonction de perte à optimiser, en l'occurence ici, nous avons choisi la Binary Crossentropy, adaptée aux classifications binaires
  .
- **Métriques :** métriques à monitorer pendant l'entraînement du modèle

Evaluation des résultats :






```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.99      0.99      1058
           1       0.99      1.00      0.99      1099

    accuracy                           0.99      2157
   macro avg       0.99      0.99      0.99      2157
weighted avg       0.99      0.99      0.99      2157
```

Test loss : 0.0318

Test set accuracy: 0.9935

Analyse des performances du modèle BiLSTM Unifié :

Courbes d'apprentissage :

- **Perte (Loss)** : Les courbes de pertes d'entraînement et de validation diminuent rapidement, atteignant des valeurs proches de zéro dès la 2ème époch, ce qui indique une bonne convergence sans signes de surapprentissage (overfitting)

- **L'Accuracy** : Les précisions d'entraînement et de validation atteignent quasiment 100 % dès les premières epochs. Démontrant un excellent ajustement du modèle.
  L'Accuracy sur le test set (données inconnues du modèle) est également excellente avec 0.9935 % de données affectées à la bonne classe.

Les courbes d'accuracy de l'ensemble d'entraînement et de validation se confondent avec la droite d'accuracy de l'ensemble test car le modèle fait très peu d'erreurs de classification.

Matrice de confusion :

- **Classe 0 not_spam :** 1053 prédictions correctes et 2 erreurs (faux négatifs).
- **Classe 1 spam** : 1097 prédictions correctes et 5 erreurs (faux positifs).

Précision : Proportion des prédictions positives correctes parmi toutes les prédictions positives

Rappel (Sensibilité) : Proportion des prédictions positives correctes parmi toutes les instances positives réelles.

F1-Score : Équilibre entre précision et rappel (moyenne harmonique)

Ces trois métriques qu'il est également important de monitorer ont des résultats excellents.

Conclusion : Le modèle atteint de très bonnes performances. Le modèle commet des erreurs de prédiction négligeables (5 spams mal classifiés contre 2 pour les non spam), montrant une excellente capacité du modèle à distinguer les deux classes.

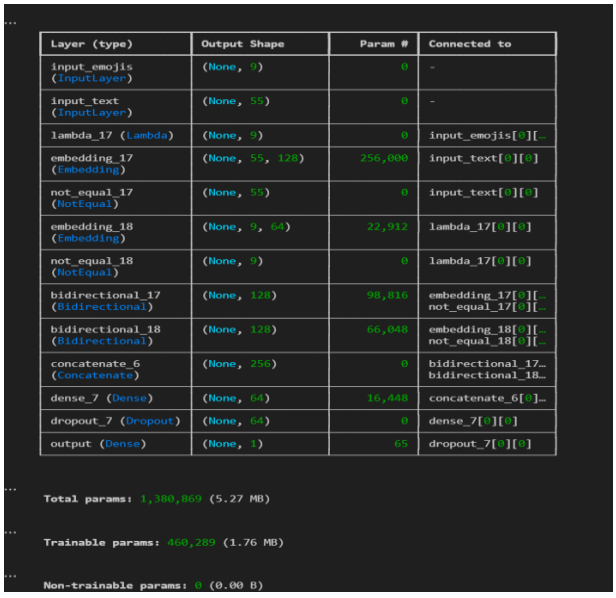
5. Section : Multimodal model

- Pour ce modèle les séquences textuelles et emoji seront traités séparément, avec une shape de séquence adaptée. La méthode du 95ème percentile sera également appliquée, mais cette fois personnalisée à chaque input. Chaque modalité (texte, emojis) est traitée indépendamment avec ses propres branches dans le réseau (Embedding, LSTM, etc.), puis les caractéristiques sont fusionnées (via une concaténation) avant la classification. Cela permet de préserver et exploiter les spécificités de chaque modalité.

- Autre particularité, c'est l'inclusion d'une couche Lambda dans les modèles multimodaux. La couche Lambda a été incluse pour éviter les confusions entre les indices du vocabulaire de l'input texte et de l'input emojis. Cette modification permet de résoudre un problème technique lié à la couche Embedding, qui ne permet pas d'utiliser des indices supérieurs à la taille de son vocabulaire (input_dim: 358). Les indices d'emoji qui était dans la plage 2000 – 2355 pour le modèle unifié, sont maintenant sur la plage 0-358 (padding + oov) sans risque de confusions avec les indices des textes encodés qui vont de 0 à 1999 (+ padding + oov).

  A l'exception de ces changements la structure du modèle reste la même afin de pouvoir comparer les résultats

Architecture globale :



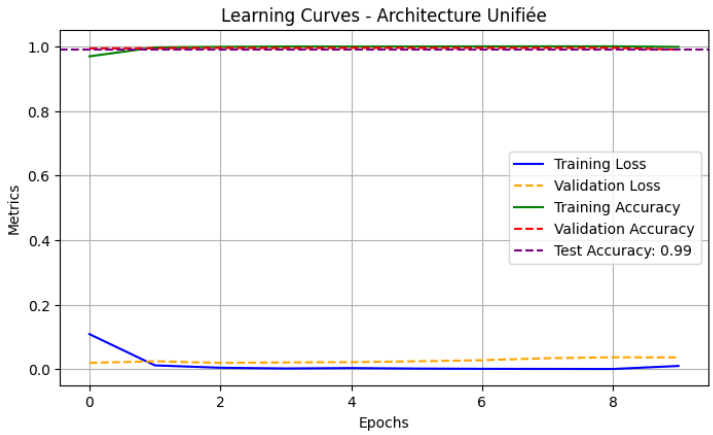
Layer (type)	Output Shape	Param #	Connected to
input_emojis (InputLayer)	(None, 9)	0	-
input_text (InputLayer)	(None, 55)	0	-
lambda_17 (Lambda)	(None, 9)	0	input_emojis[0][…
embedding_17 (Embedding)	(None, 55, 128)	256,000	input_text[0][0]
not_equal_17 (NotEqual)	(None, 55)	0	input_text[0][0]
embedding_18 (Embedding)	(None, 9, 64)	22,912	lambda_17[0][0]
not_equal_18 (NotEqual)	(None, 9)	0	lambda_17[0][0]
bidirectional_17 (Bidirectional)	(None, 128)	98,816	embedding_17[0][… not_equal_17[0][…
bidirectional_18 (Bidirectional)	(None, 128)	66,048	embedding_18[0][… not_equal_18[0][…
concatenate_6 (Concatenate)	(None, 256)	0	bidirectional_17… bidirectional_18…
dense_7 (Dense)	(None, 64)	16,448	concatenate_6[0]…
dropout_7 (Dropout)	(None, 64)	0	dense_7[0][0]
output (Dense)	(None, 1)	65	dropout_7[0][0]

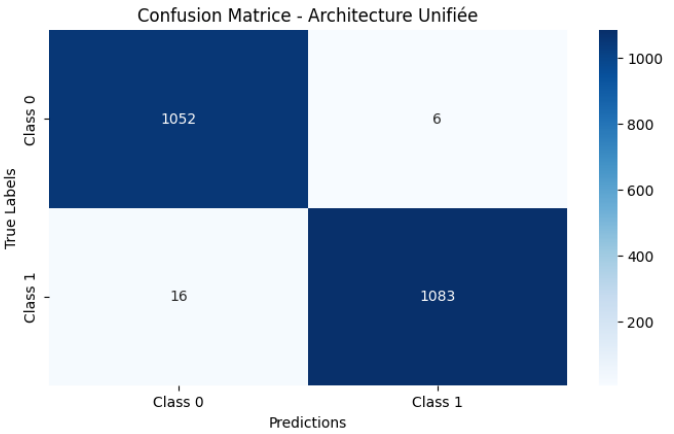
Total params: 1,380,869 (5.27 MB)

Trainable params: 460,289 (1.76 MB)

Non-trainable params: 0 (0.00 B)

Evaluation des résultats :






```
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      1058
           1       0.99      0.99      0.99      1099

    accuracy                           0.99      2157
   macro avg       0.99      0.99      0.99      2157
weighted avg       0.99      0.99      0.99      2157
```

Test set loss: 0.0288

Test set accuracy: 0.9898

La tendance générale montre que le comportement du modèle pendant l'entraînement est très similaire à celui de l'architecture unifiée. Les courbes de pertes pour l'entraînement et la validation convergent rapidement, rendant difficile l'identification des performances d'accuracy sur le training set et le validation set, car les deux courbes se superposent autour de 100 % dès la deuxième époque, même constat pour la droite représentant les résultats de prédiction (accuracy) sur le test.

Les performances du modèle multimodal, bien que globalement excellentes, sont légèrement inférieures à celles de l'architecture unifiée. Le modèle multimodal commet davantage d'erreurs dans la détection de la classe spam, avec 14 erreurs supplémentaires. En revanche, pour la classe non_spam, il n'enregistre que 6 erreurs, ce qui le rend presque aussi performant que le modèle unifié.

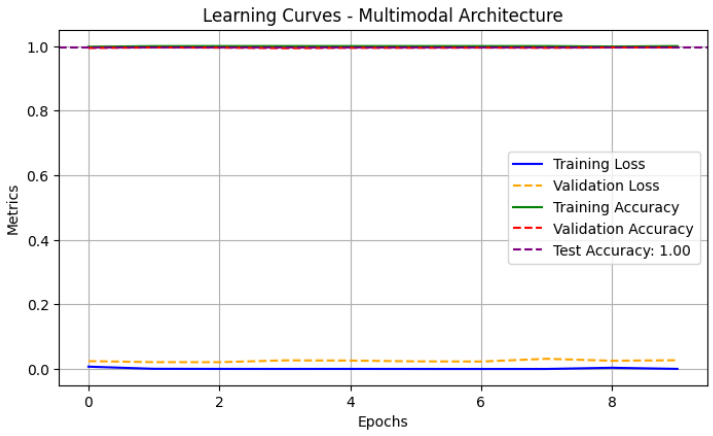
6. Section : Multimodal model avec optimisation des hyperparamètres

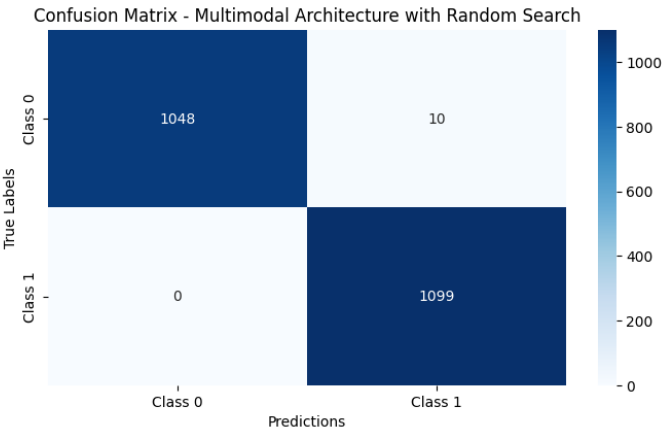
- Une recherche aléatoire (Random Search) est réalisée à l'aide de Keras Tuner (avec un modèle défini dans un HyperModel) pour optimiser les hyperparamètres du modèle multimodal. Cette recherche explore 15 configurations différentes afin de trouver les meilleures valeurs pour les hyperparamètres suivants :

  - Taille de l'embedding pour les textes et les emojis
  - Nombre de neurones dans la couche LSTM
  - Nombre de neurones dans la couche Dense
  - Taux de dropout
  - Pas du taux d'apprentissage (learning rate)

Nous gardons une structure de modèle identique.

Evaluation des résultats :






```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.99      1.00      1058
           1       0.99      1.00      1.00      1099

    accuracy                           1.00      2157
   macro avg       1.00      1.00      1.00      2157
weighted avg       1.00      1.00      1.00      2157
```

Loss sur le test set : 0.0312

Accuracy sur le test set : 0.9954

Le modèle multimodal optimisé pour les hyperparamètres affiche également d'excellentes performances globales. Il réussit à classifier parfaitement tous les Non-Spam (aucun faux négatif). Cependant, il montre une baisse de performance sur la classe Spam, avec 10 erreurs de classification.

7. Section : Choix du meilleur modèle et piste d'améliorations

Parmi l'ensemble de nos modèles, le modèle multimodal avec optimisation des hyperparamètres sera privilégié, car il est préférable de garantir que tous les e-mails légitimes soient correctement identifiés dans notre boîte de réception, même si cela implique qu'une petite proportion de spams puisse ne pas être détectée. Une meilleure détection des faux positifs sera privilégiée afin d'éviter que le modèle ne classe dans les spam un e-mail légitime. Cette erreur pourrait avoir des répercussions importantes pour l'utilisateur plutôt que l'inverse.

Pistes d'améliorations :

- Afin de rendre le modèle plus robuste, il serait intéressant de l'évaluer sur de nouvelles données afin de mesurer la capacité du modèle à généraliser ses prédictions

- Mettre en place une validation croisée pourrait améliorer la robustesse des résultats. La validation croisée utilise un ensemble de test différent à chaque itération, ce qui permet de mieux généraliser les performances du modèle en évitant qu'elles soient spécifiques à un seul découpage des données

- L'utilisation du transfert learning (état de l'art) pourrait permettre d'explorer d'autres structures de modèles plus complexes et approuvées scientifiquement afin d'optimiser les performances de prédiction. Intégrer un mécanisme d'attention au modèle BI_LSTM pourrait également être intéressant.