Rapport Projet 2

Exploration des modèles de langue

Groupe n°3

MTI881 - Traitement du langage naturel appliqué

Étudiants

Tess Czaplinski,
Nicolas Salvan,
Ahmadou Moustapha Beye

Encadrants

Pierre-André Ménard,
Sylvie Ratté

Table des matières

Introduction 3
Étape 1 : Modèle de référence avec MedMention 4
1.1 Données utilisées A RELIRE 4
1.2 Modèle et entraînement 6
1.3 Résultats et analyse 9
Étape 2 : Ajout des annotations du cours 14
2.1 Données utilisées 14
2.3 Modèle et entraînement 15
2.3 Résultats et analyse 17
Etape 3 : Modifier l'entraînement pour utiliser un modèle génératif 18
3.1 Description du modèle 18
3.2 Modèle et entraînement 18
3.3 Résultats 21
Etape 4 : Thème libre 22
4.1 Choix de la méthode 22
4.2 Mise-en-œuvre 22
4.3 Utilisation Optuna 23
Discussion 25
Conclusion 26
Annexes 27
Étape 1 27

Introduction

Ce rapport de projet s'inscrit dans la continuité des phases d'annotation de texte médicaux et d'adjudication de données réalisées précédemment.

Cette seconde partie de projet concerne l'entraînement d'un modèle de langues, destiné à annoter des textes médicaux avec les types de concepts UMLS. L'objectif principal est d'optimiser l'entraînement afin d'obtenir un modèle à la fois rapide mais aussi précis et capable de maintenir une qualité d'annotation conforme à ce qui serait attendu dans le domaine biomédical.

L'une des utilités de ce projet vise à automatiser l'annotation de texte pour réduire la phase d'annotation, souvent chronophage dans le monde médical et qui est réservée à des experts qui n'ont pas le temps pour ce type de tâche.

Nous nous appuierons notamment sur des données annotées et curées obtenues par le projet précédent. Nous tâcherons d'identifier les erreurs d'annotation récurrentes qu'effectue le modèle et nos propos seront illustrés par des graphes et des exemples d'erreurs typiques.

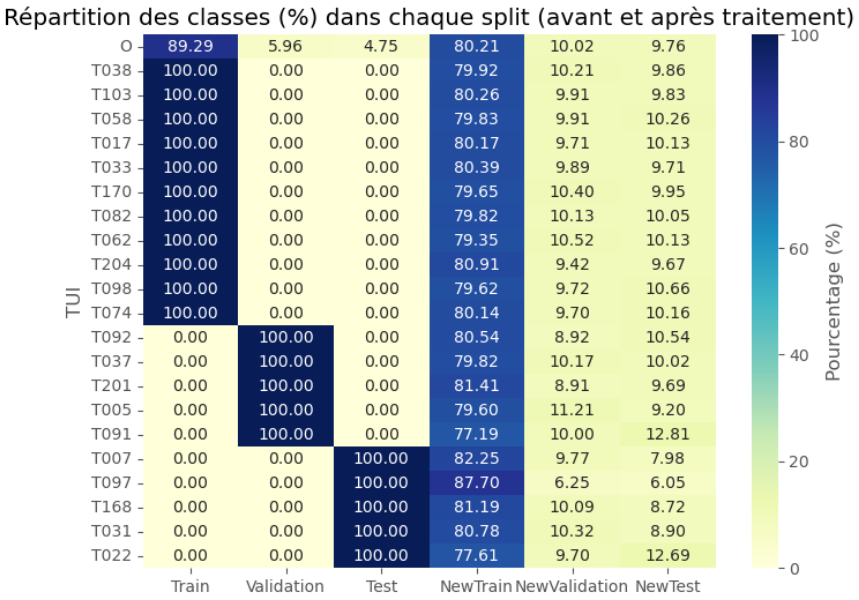
Pour tester le code de ce projet et consulter en détail les modifications apportées au code original, veuillez-vous référer au fichier README.md situé à la racine du dépôt.

Étape 1 : Modèle de référence avec MedMention

1.1 Données utilisées A RELIRE

Nous avons utilisé l'ensemble de données MedMentions-ZS. Ce jeu de données est une version modifiée de MedMentions qui permet l'apprentissage en zero-shot. Il est très utilisé dans des tâches de reconnaissance d'entités nommées (NER) biomédicales. Ce dataset est au format BIO2, où chaque token (colonne token) est associé à une étiquette ner_tag (colonne ner_tag). En comparant les données de MedMentions et de CONLL2003 qui est le jeu de données de base à notre disposition, on remarque qu'ils sont très différents. Tout d'abord il y a beaucoup plus de classes (5 classes pour CONLL2003, 22 classes pour MedMentions). Cela implique un fine-tuning plus complexe. Ensuite, on a pu observer que MedMentions n'était pas exhaustif : toutes les classes possibles ne sont pas présentes dans le jeu de données (il manque des TUI, qui sont des identifiants de catégorie sémantique UMLS). Enfin, le split de test et de validation par défaut de Medmentions ne contient pas toutes les classes de l'entraînement contrairement à CONLL2003. Nous pouvons observer cela dans les tableaux suivants.

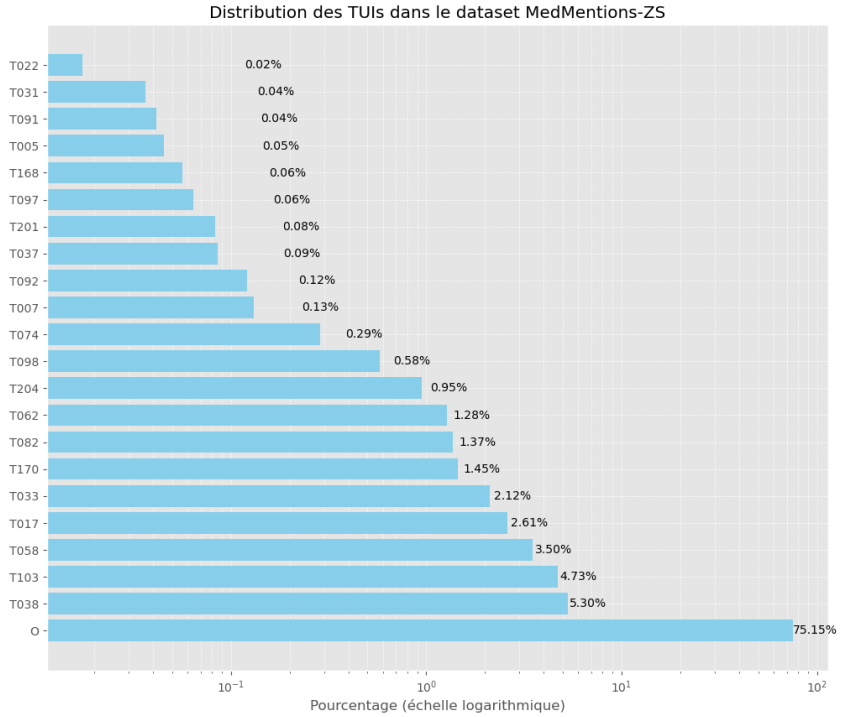
Nous avons donc réalisé un sur-échantillonnage. Avec la bibliothèque **scikit-multilearn**, on peut réaliser une division en sous-ensemble d'entraînement, de validation et de test de façon itérative tout en conservant la même proportion de classes dans chaque ensemble et en conservant l'intégrité des phrases. On a réussi à implémenter une division aléatoire qui conserve les proportions de classe.



(Figure 1) *Répartition des classes dans chaque division (en pourcent) avec à gauche la séparation par défaut et à droite, la séparation avec 80%-10%-10%. La division par défaut des données favorise le Zéro-Shot, certaines classes étant absentes de l'ensemble d'entraînement. Cela a conduit à un F1 score nul lors des premiers tests,*

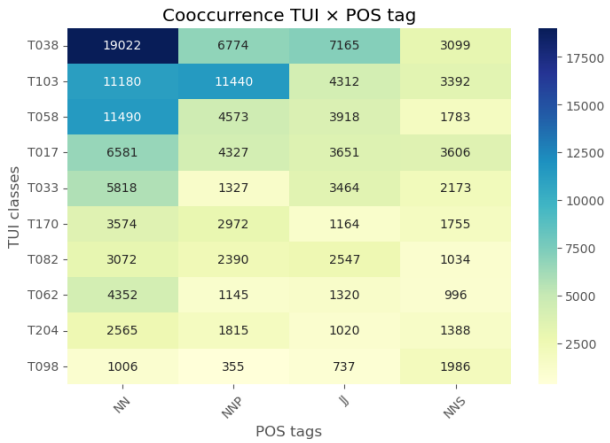
contrairement au jeu ConLL2003. Le rééquilibrage a permis une répartition plus stable des TUIs, essentielle pour permettre au modèle d'apprendre et de prédire toutes les classes.

Nous avons donc réalisé un entraînement avec 80% des données dans l'entraînement, et 10% pour la validation et 10% pour le test, avec 768150 phrases en tout. En tout, il y a 43 classes en prenant en compte le format BIO2, et la classe « O » est clairement majoritaire.



(Figure 2) *Distribution des classes dans l'ensemble de données. On voit que la classe « O » est majoritaire. Les deux autres classes les plus majoritaires sont T038 (Biologic Function) et T103 (Chemical). La classe avec le plus faible effectif est T022 (Body System).*

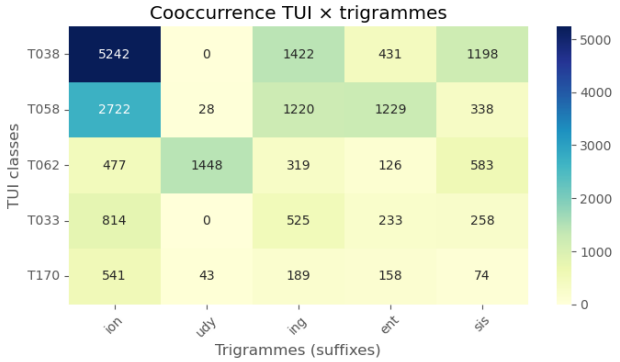
Nous avons également analysé les catégories lexicales en effectuant du POS-tagging avec la bibliothèque **NLTK**.



(Figure 3) *Cooccurrence des TUI et des POS tags (catégories sémantiques) les plus communes (NN=Nom commun, NNP=Nom propre, JJ=adjectif, NNS=Noms pluriels). On remarque que certaines classes sont dominées par des noms communs (NN), ce qui peut indiquer beaucoup de vocabulaire peu spécifique (« inflammation », ...). On remarque que la classe T103 – Chemical contient beaucoup de noms propres, ce qui pourrait indiquer la présence de molécules ou*

de protéines dans les données. Les classes contenant des adjectifs seraient celles qui utilisent beaucoup de modificateurs, comme T103 notamment.

Nous avons continué avec l'analyse morpho-lexicale, par le biais de l'analyse des trigrammes de suffixes qui pourraient nous indiquer la catégorie des mots.



(Figure 4) *Analyse morpho-lexicale des classes majoritaires. La classe T062 (Research Activity) indique peut-être une sur-présence du terme « study » dans le jeu de données. De même pour la catégorie T058 (Health Care Activity), une sur-présence du terme « patient » pourrait biaiser le modèle par la suite.*

1.2 Modèle et entraînement

Nous avons utilisé le modèle BERT-BASE-UNCASED pour cette phase. Il s'agit d'un modèle basé sur des *transformers* entraîné sur du texte en anglais pour des tâches de MLM (Masked Language Modeling) et NSP (Next Sentence Prediction). De plus, nous avons exploré un certain espace d'hyperparamètres afin d'optimiser notre modèle pour la reconnaissance d'entités médicales. Le tableau 1 qui suit en présente une liste non-exhaustive.

Hyperparamètres testés	Valeurs testées
Nombre d'époques *(num_train_epochs)*	1, 3, 5, 10, 15, 30
Accumulation de gradient *(gradient_accumulation_steps)*	1, 2
Taux d'apprentissage *(learning_rate)*	1e-4 , 5e-5, 1e-5, 1e-6, 1e-7
Type de planificateur de taux d'apprentissage *(lr_scheduler_type)*	linear, reduce_lr_on_plateau, constant, cosine
Taille de batch d'entraînement *(per_device_train_batch_size)*	8, 16, 32
Taille de phrase *(max_seq_length)*	64, 128, *None*

(Tableau 1) *Liste (non-exhaustive) des hyperparamètres testés lors de la phase 1.*

Ainsi, le nombre d'époques permet de contrôler le temps d'apprentissage, tandis que le taux d'apprentissage influe sur la rapidité et la stabilité de la convergence. L'accumulation de gradients, quant à elle, nous a permis de simuler un batch-size plus élevé sans dépasser les contraintes matérielles, et la taille de batch ainsi que la longueur

maximale des séquences ont été choisies pour conserver l'information pertinente tout en maîtrisant les ressources computationnelles. Ces réglages ont été sélectionnés pour atteindre notre objectif d'obtenir un modèle performant et stable sur des textes médicaux.

Notre stratégie d'entraînement a consisté à entraîner sur un nombre faible d'époques pour choisir les hyperparamètres par RandomSearch, en sélectionnant les modèles avec les meilleurs scores F1-global et les courbes d'apprentissage les plus prometteuses (pas de sur ou sous apprentissage). Une fois cette sélection réalisée, on entraîne le modèle avec un learning rate plutôt élevé sur plus d'époques (10+) et on diagnostique les résultats d'entraînement avec les scores par classes. Enfin, on continue de faire du fine-tuning en partant du meilleur des 3 checkpoints des modèles les plus prometteurs selon le F1 global sur l'ensemble de validation. Comme il s'agit d'un processus manuel, nous avons plus de contrôle sur les objectifs que l'entraînement doit atteindre.

Après sélection, nous avons analysé les performances de 3 modèles avec des configurations différentes :

- Le **Modèle 1** (job 1717) qui utilise un learning rate de 1e-5 avec un scheduler linéaire et un batch size de 16 sur 15 époques, a permis d'observer une descente progressive de la loss, mais la convergence reste relativement lente.

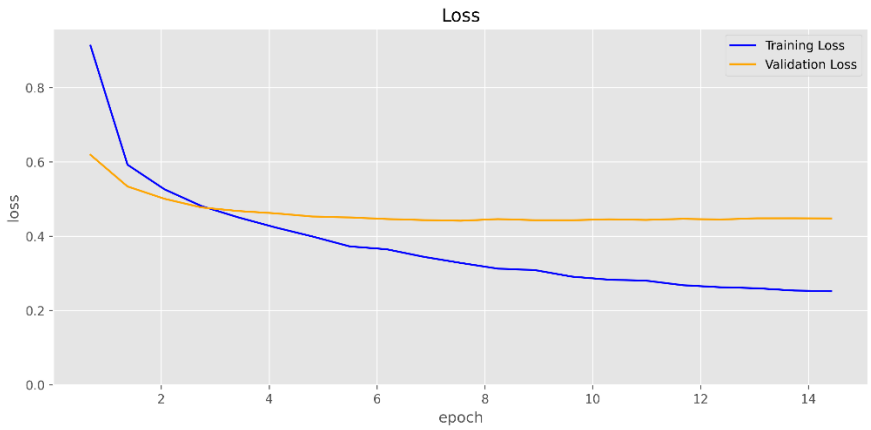


Figure 5 - Loss du job 1717

- Le **Modèle 2** (job 1713) se distingue par un taux encore plus conservateur (1e-6) et un batch size réduit (8), nécessitant 30 époques. Cette configuration met en évidence l'impact négatif d'un learning rate trop faible, entraînant un apprentissage trop lent malgré un nombre d'époques prolongé.

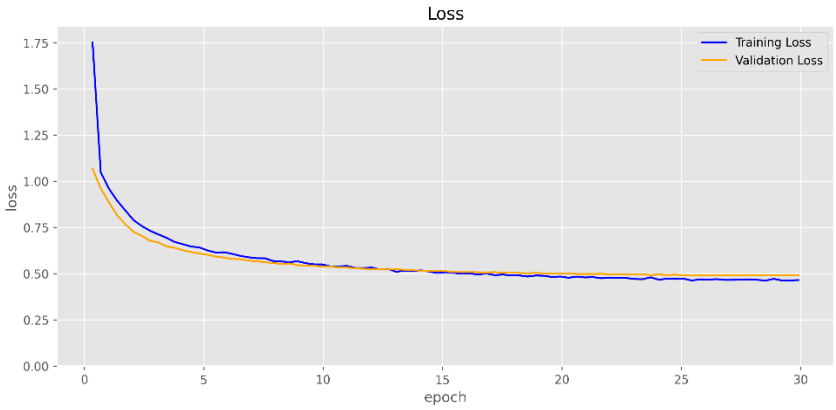


Figure 6 - Loss du job 1713

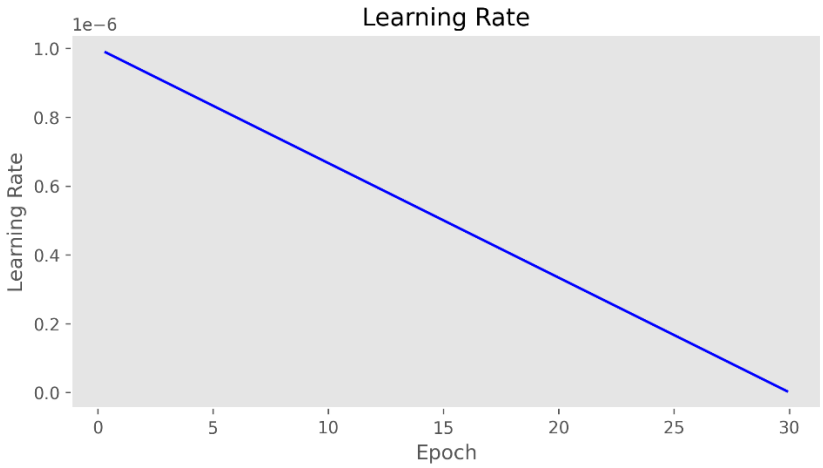


Figure 7 - Learning rate du job 1713

- En revanche, le **Modèle 3** (job 1731) – utilisant un learning rate de 5e-5 associé à un scheduler de type reduce_lr_on_plateau et un batch size de 16 sur 15 époques – offre une convergence plus rapide et stable dans l'ensemble, même si l'analyse visuelle (voir Figures 1, 3 et 4) révèle des signes d'overfitting à partir de certaines époques.

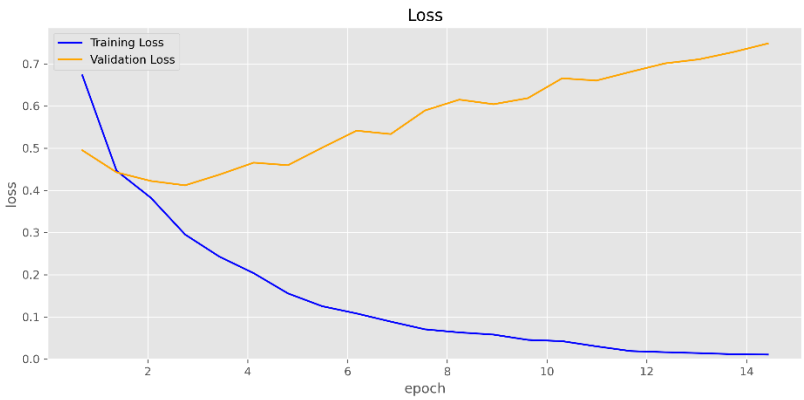


Figure 8 - Loss du job 1731

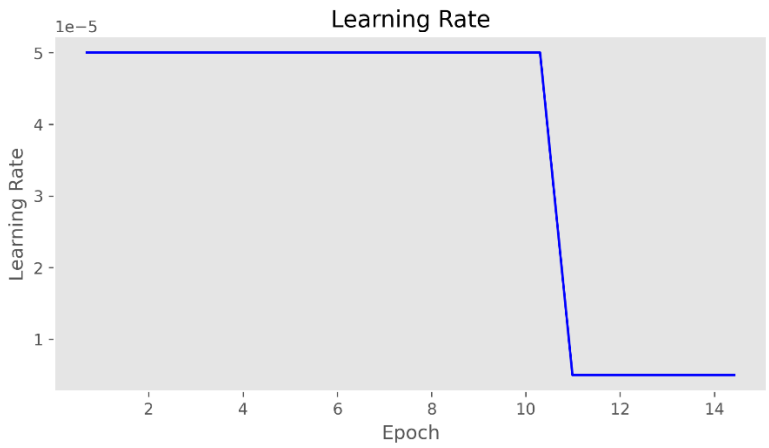


Figure 9 - Learning rate du job 1731

Bien que le job 1731 présente des signes d'overfitting (avec une divergence notable entre la loss d'entraînement et de validation à partir de la 4ème époque), il offre néanmoins de meilleures performances globales et des courbes d'apprentissage plus prometteuses que les autres configurations testées. Il constitue ainsi le meilleur compromis entre rapidité de convergence et performance, tout en soulignant la nécessité d'intégrer des mécanismes supplémentaires (comme l'early stopping ou la régularisation) afin de limiter le surapprentissage lors du fine-tuning ultérieur.

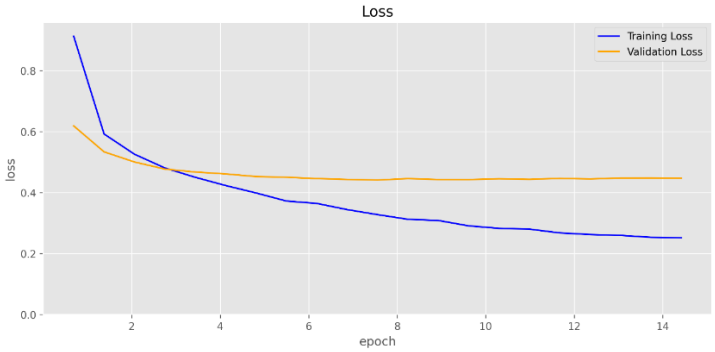
1.3 Résultats et analyse

Performances globales : Nous avons analysé les performances globales en utilisant les métriques d'exactitude *(accuracy)*, précision, rappel *(recall)* F1-score. Le modèle qui a montré les meilleures performances moyennes est le troisième, avec un score F1 de 66.3% et une accuracy de 88.7%. Néanmoins, il nous faut nuancer ces scores globaux, étant donné la majorité écrasante de classe "O".

Métrique globale	Modèle 1	Modèle 2	Modèle 3
Accuracy	0.868	0.850	0.887
Precision	0.578	0.523	0.651
Recall	0.628	0.551	0.676
F1	0.602	0.537	0.663

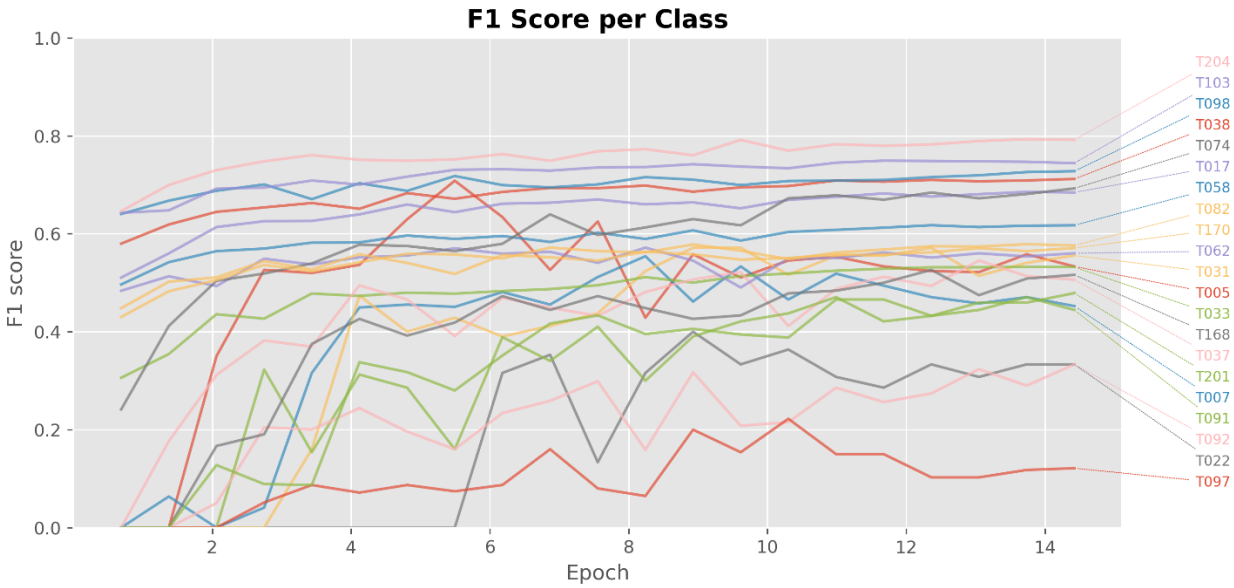
***(Tableau 2)** Métriques globales des meilleurs modèles trouvés par RandomSearch*

Performances de l'entraînement : En observant les courbes de perte d'entraînement et de validation des modèles 1 et 2 (voir Annexes), on observe un overfitting pour le modèle 1 et un underfitting pour les modèles 2 et 3, ce qui est une conséquence directe du déséquilibre de classes.



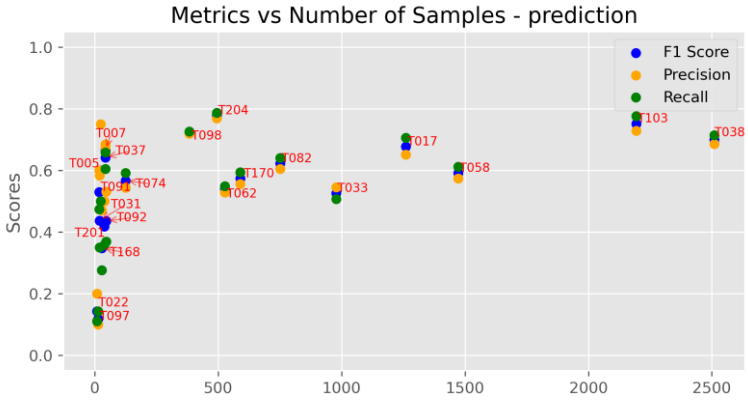
***(Figure** 10) Entraînement du modèle 1. On observe un over-fitting des données à partir de l'epoch 4 environ : les courbes d'entraînement et de validation sont fortement différentes.*

Scores par classe : Nous avons analysé les scores par classe du Modèle 3, qui a obtenu les meilleures performances globales.

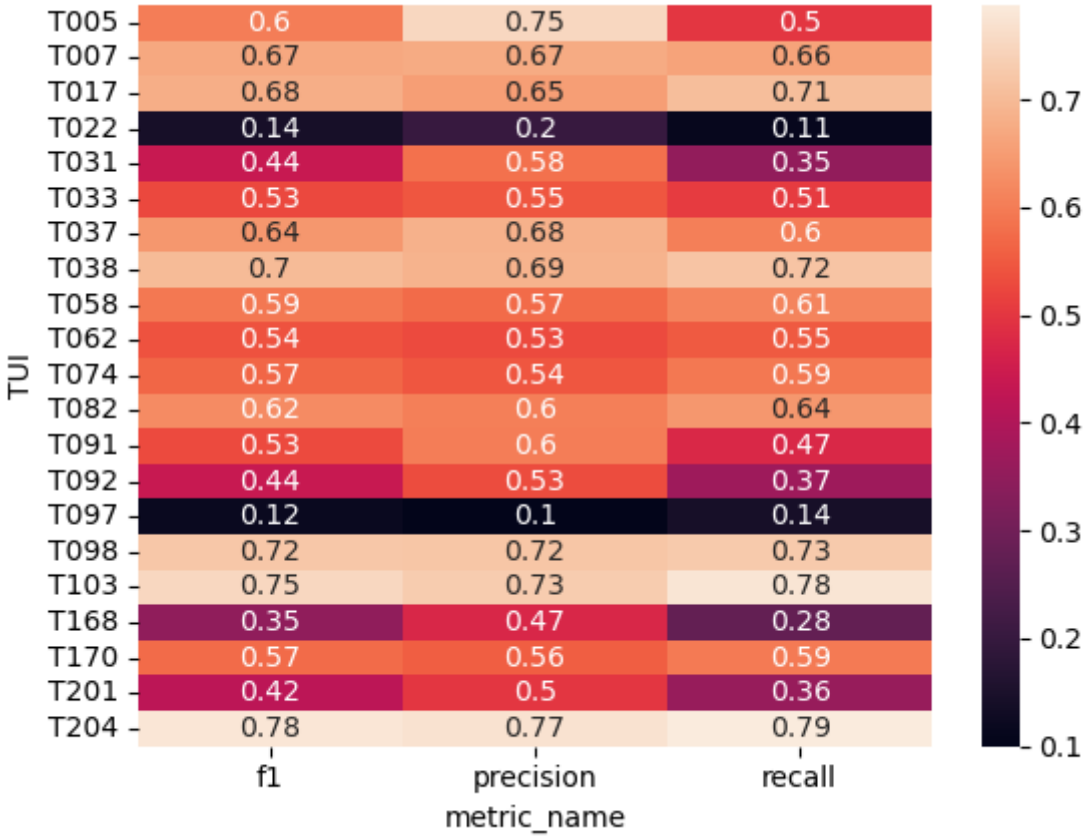


(Figure 11) Évolution des F1 scores de validation par classe. On observe que les classes les mieux prédites sont T204 (Eukaryote), T103 *(Chemical) et T098* (Population Group). La classe avec le moins bon score est T097 *(Profession or occupationnal group)*.

Dans la plupart de nos entraînements, nous avons observé une forte corrélation entre les métriques et l'effectif de la classe dans les données d'entraînement. Cela peut s'expliquer par tes biais d'entraînement et le manque de pondération dans la fonction de coût.



(Figure 12) Score de métriques en fonction de l'effectif. On observe une tendance générale où plus une classe est représentée, meilleurs sont ses scores.



Impact :

Bien que la convergence apparente soit rapide au début, les oscillations provoquées par un learning rate trop élevé mènent à une perte d'efficacité dans l'apprentissage (F1 global et autres métriques), comme indiqué dans la figure 10 - Métriques globales.

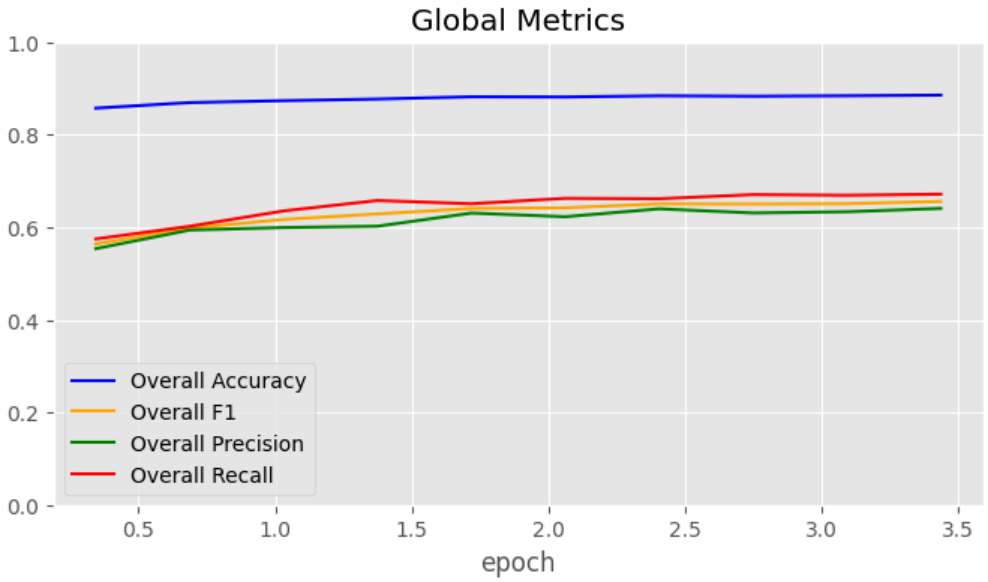


Figure 13 10 - Métriques globales

Conclusion sur le taux d'apprentissage :

La configuration baseline (5e-5) avec un scheduler cosine apparaît comme la plus équilibrée ; elle offre la stabilité nécessaire tout en assurant une bonne vitesse de convergence. Un LR trop bas ralentit l'apprentissage, tandis qu'un LR trop élevé induit une instabilité qui se traduit par des oscillations de la loss et des métriques moins fiables.

Autres hyperparamètres et leur impact

En plus du taux d'apprentissage, nous avons examiné l'impact des autres hyperparamètres tels que le batch size, le nombre d'epochs, et la longueur maximale des séquences :

- Batch size (per_device_train_batch_size & per_device_eval_batch_size) :
  Un batch size de 16 permet d'avoir un bon compromis entre la stabilité des gradients et l'utilisation efficace de la mémoire GPU. Les courbes de loss et les métriques globales indiquent une bonne régularité et une convergence satisfaisante sous cette configuration.

- Nombre d'epochs (num_train_epochs) :
  Bien que nous ayons testé 5 epochs, l'analyse des courbes de loss et des métriques montre que la convergence est généralement atteinte autour de la 3e

ou 4e epoch. Une stratégie d'early stopping pourrait être envisagée pour éviter le surapprentissage et économiser du temps.

- Longueur maximale des séquences (max_seq_length) :
  La valeur de 128 tokens permet de capturer suffisamment d'information dans des titres ou des extraits courts, tout en réduisant la charge computationnelle.

Synthèse et recommandations

En résumé, les analyses montrent que :

Le baseline (5e-5) est la configuration la plus stable et performante, avec une convergence progressive et des métriques globales satisfaisantes. En revanche, un taux d'apprentissage trop bas (1e-5) ralentit la convergence et risque de conduire à une performance sous-optimale.

D'un autre côté, un taux d'apprentissage trop élevé (2e-4) induit une instabilité manifeste dans la descente de la loss et des fluctuations erratiques des métriques.

Les autres hyperparamètres, tels que le batch size, le nombre d'epochs et la longueur maximale des séquences, sont correctement choisis et permettent une convergence équilibrée pour notre modèle.

Étape 2 : Ajout des annotations du cours

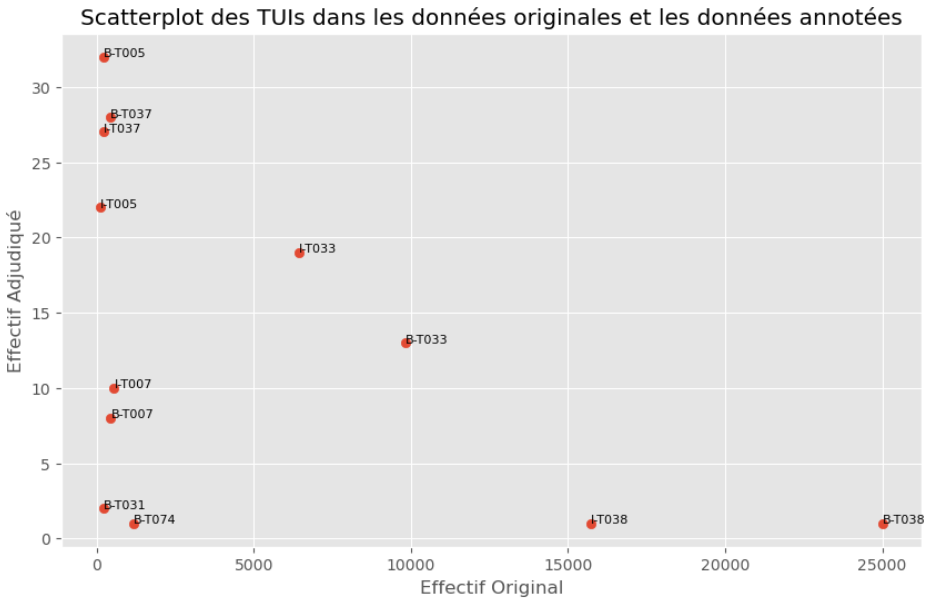
2.1 Données utilisées

Nous avons utilisé les données issues de la phase de curation du Projet 1 du cours, correspondant à des annotations de résumés d'articles médicaux. Les annotations des équipes 1, 2, 3, 4, 6 ont été retenues et nous avons écarté celles de l'équipe 5 étant donné leur structure, qui ne présentait pas de curation.

Les données sélectionnées ont ainsi été parsées avant d'être placées dans une base de données unifiée nommée dataset_concat.json. Lors de ce traitement de concaténation et parsing, toutes les annotations de type CUI ont été converties en leur équivalent TUI à l'aide de l'API UMLS. Les données annotées pour lesquelles l'API ne trouve pas de TUI sont ignorées par le modèle et sont annotées "IGN".

Remarque: Il est essentiel de ne pas les annoter "O" puisqu'il s'agit tout de même de données médicales, identifiées par les annotateurs, et qui ne peuvent donc pas être considérées comme des entités extérieures au domaine médical.

Grâce à ce traitement, les données ont pu être fusionnées avec celles du jeu de données MedMentions. Les résultats de cette analyse sont présentés dans la figure suivante.

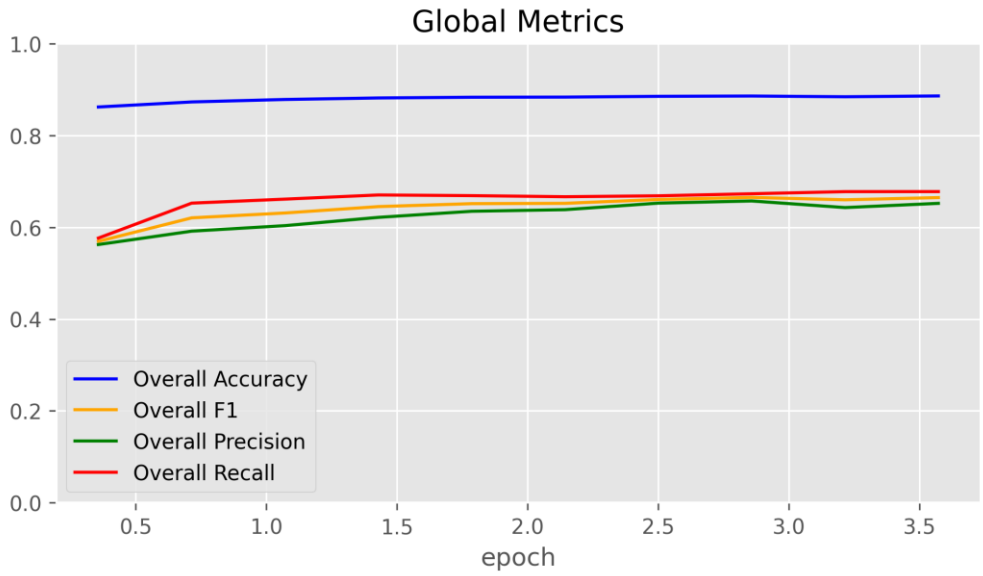


(Figure 14) Nuage de points de l'effectif des classes dans les données ajoutées en fonction de l'effectif présent dans MedMentions. On voit que les données adjudiquées complètent assez bien les classes en sous-effectif dans MedMentions, comme T005, T037 et T007 (en haut à gauche du graphe).

Même si les données ajoutées permettent de compléter certaines classes, il faut remettre en perspective le fait qu'elles représentent un effectif très faible par rapport aux classes présentes dans le jeu de données original. De plus, il est possible que certaines annotations ne soient pas complètes, étant donné que peu de groupes ont considéré d'annoter toutes les catégories UMLS présentes dans MedMentions.

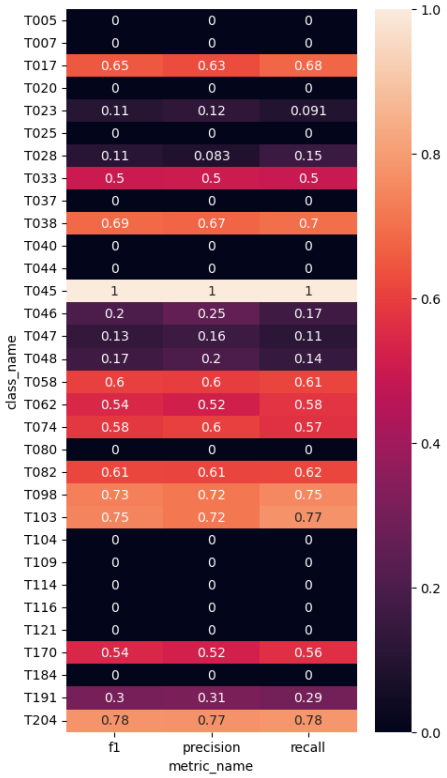
2.3 Modèle et entraînement

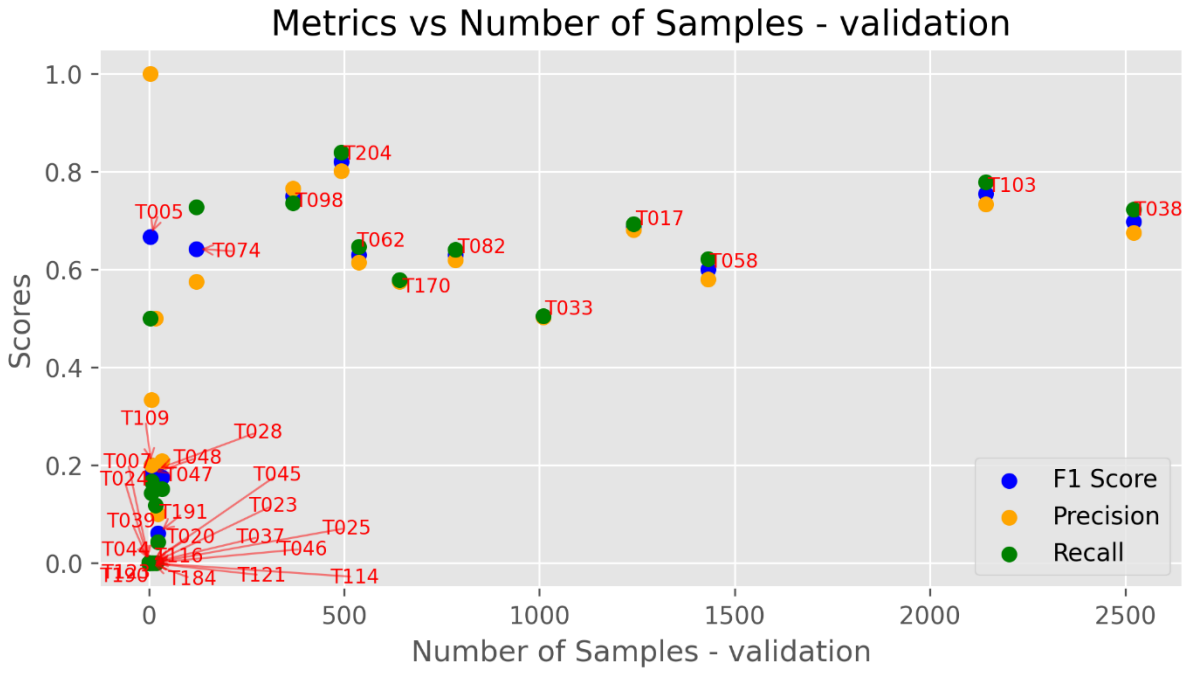
Nous avons testé d'entraîner notre modèle sur les hyperparamètres définis par défaut.



Pour les métriques globales, la précision globale reste constamment élevée, indiquant une bonne performance générale du modèle. Le F1 Score, la précision et le rappel montrent des tendances stables avec de légères améliorations au fil des époques. Ces résultats suggèrent que le modèle maintient une performance équilibrée et stable, même s'il y a encore beaucoup de marge pour améliorer le F1 Score et le rappel pour certaines catégories rares.

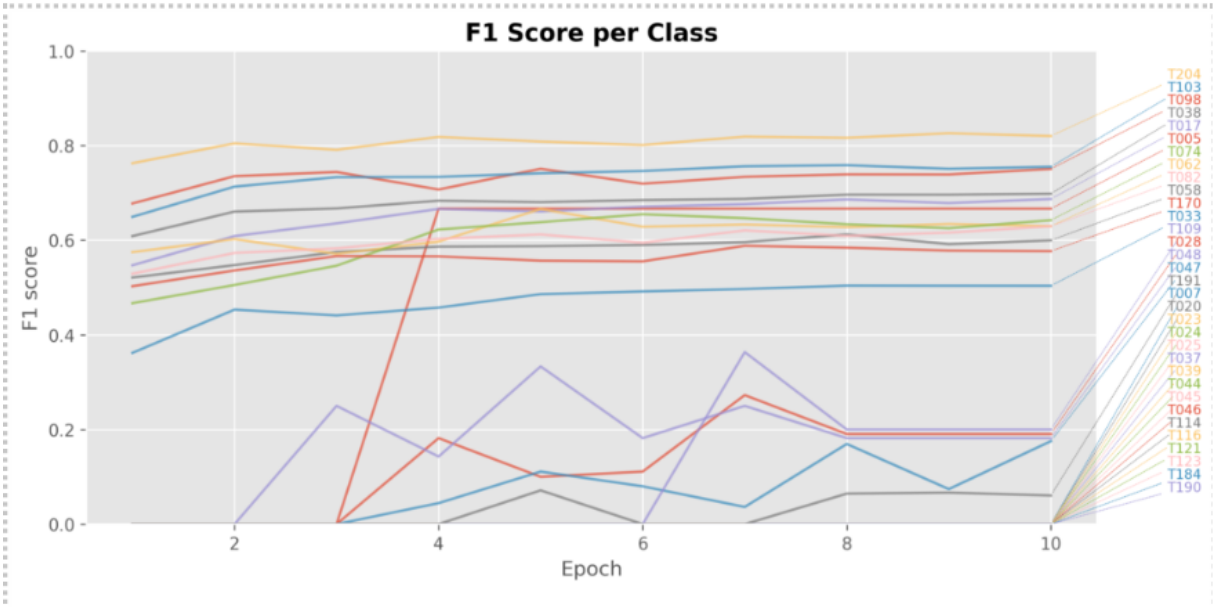
En incluant les nouvelles données, on a observé une baisse légère des performances globales excluant l'accuracy. Cela peut s'expliquer par l'amélioration significative qu'a apporté le lr_scheduler dans l'étape 1.





En observant les métriques par classe, le modèle BERT montre des performances variées selon les catégories, ce qui est attendu étant donné la diversité des données biomédicales. Les catégories TUI avec plus d'échantillons tendent à avoir des métriques plus stables et plus élevées. Ainsi, on observe que les catégories avec un nombre élevé d'échantillons (comme celles autour de 2500 échantillons) montrent des

performances en général meilleures. Cela suggère que le modèle bénéficie de plus de données pour apprendre les caractéristiques de ces catégories.



Sur ce graphe, les lignes représentent différentes classes, avec des performances variées. Certaines classes ont des F1 Scores élevés et stables, tandis que d'autres montrent des variations ou des scores plus bas. Globalement, le modèle démontre une capacité à bien prédire certaines classes, mais il y a des disparités notables dans la performance entre les classes, indiquant des domaines potentiels pour l'amélioration.

2.3 Résultats et analyse

Le modèle BERT uncased capture correctement les classes fréquentes, mais échoue sur les classes rares, ce qui s’explique notamment par notre jeu de données qui reste déséquilibrés (certains labels très fréquents comme T038, T103, tandis que d’autres sont quasi absents), ce qui est normal sans stratégies de rééchantillonnage, d'augmentation de données, ou de pondération de la perte.

Etape 3 : Modifier l’entraînement pour utiliser un modèle génératif

3.1 Description du modèle

Pour cette étape, nous testons l’utilisation du modèle GPT-Neo 1.3B. Il est entraîné pour générer du texte de manière auto-régressive, contrairement à BERT-base-uncased qui est un modèle *encoder-only,* créé pour fonctionner de manière bidirectionnelle sur un texte.

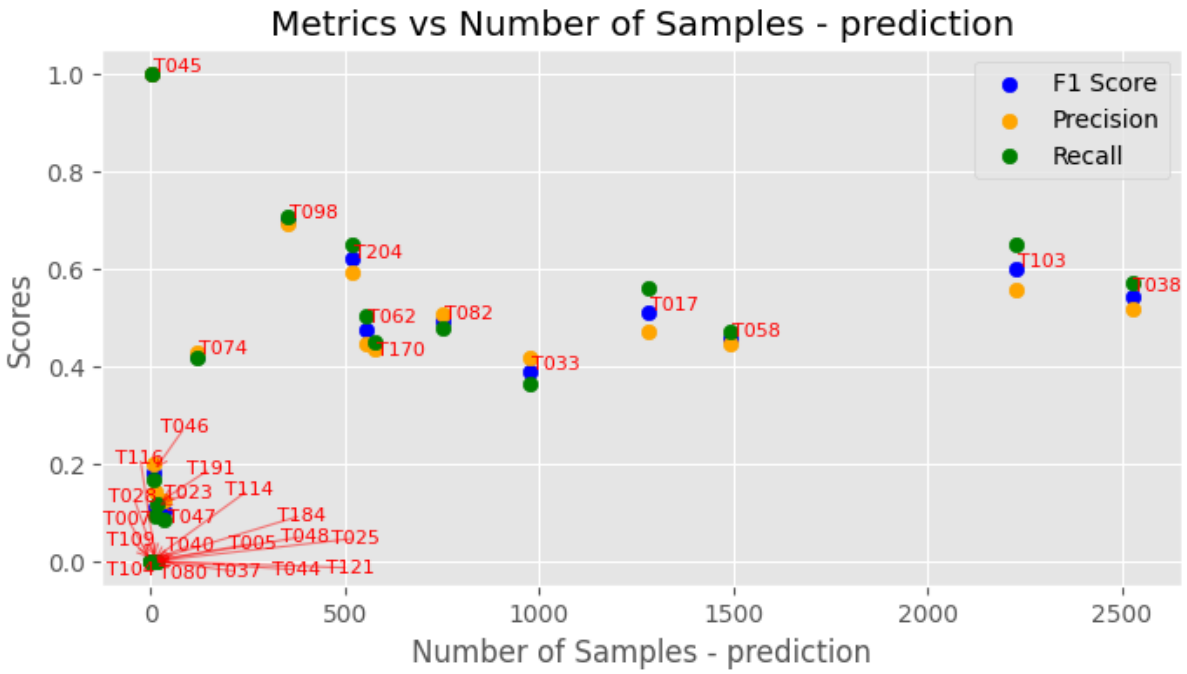
D'un côté, tandis que BERT permet de travailler spécifiquement sur des tâches de classification ou d’annotation, GPT-Neo génère des séquences mot par mot.

3.2 Modèle et entraînement

Pour faire tourner le script, nous utilisons la bibliothèque accelerate pour simplifier l’exécution de GPT-Neo 1.3B d’EleutherAI, car il s’agit d’un modèle volumineux et accelerate permet une optimisation de la mémoire et du chargement pour éviter les dépassements et maximiser les performances.

Nous avons remarqué que ce modèle prend beaucoup plus de temps à l’entraînement par rapport à Bert (). Cela s’explique notamment par le fait que GPT-Neo est un modèle decoder-only auto-régressif, qui ne peut pas paralléliser la prédiction des tokens de sortie comme le fait BERT, qui est encoder-only et traite tous les tokens en parallèle pour l’étiquetage.

Il a aussi fallu adapter GPT Neo à nos données, notamment en instaurant un padding par défaut et en modifiant l’alignement avec “add_prefix_space”, car GPT-Neo nécessite un préfixe d’espace pour traiter correctement les tokens, contrairement à BERT-base-uncased.



Le modèle basé sur GPT-Neo montre des performances variées selon les catégories.
On voit que certaines catégories comme T045 et T098 ont des métriques élevées, indiquant une bonne performance.
Mais d'autres catégories comme T114 et T103 ont des métriques plus faibles, suggérant que le modèle a du mal à bien prédire ces catégories.
Les catégories avec plus d'échantillons tendent à avoir des performances plus stables, ce qui est typique car plus de données permettent au modèle de mieux apprendre les caractéristiques de ces catégories.

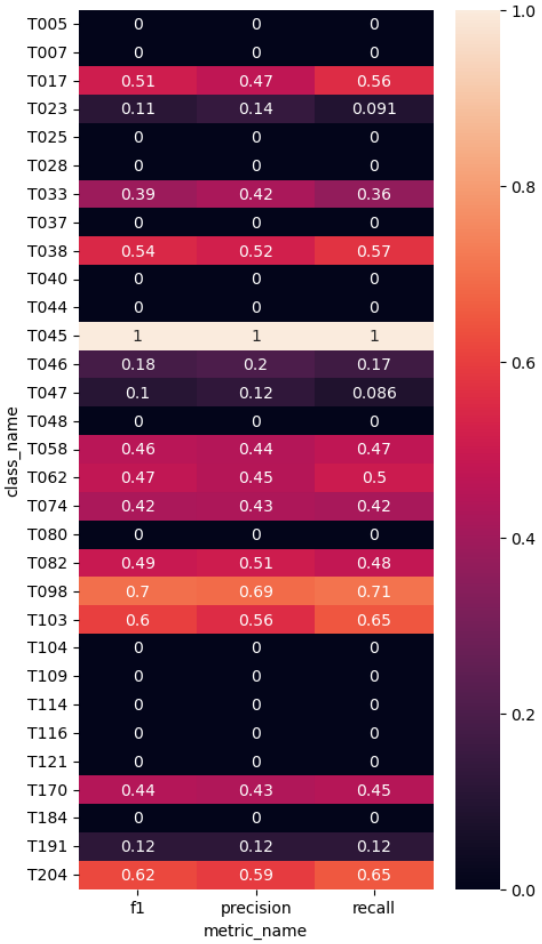
Comparaison métriques évaluation et prédiction de BERT et GPT-Neo

Métrique	BERT (uncased)	GPT-Neo
Eval Loss	0.63	1.23
Eval Accuracy	88.62%	85.88%
Eval F1	0.665	0.533
Eval Precision	0.652	0.516
Eval Recall	0.678	0.552
Predict Loss	0.66	1.26
Predict Accuracy	87.96%	85.22%
Predict F1	0.647	0.519

Predict Precision	0.634	0.500	
Predict Recall	0.662	0.539	
Train Loss		0.195	0.161
Train Samples/sec		283.77	12.15
Train Steps/sec		17.74	1.52

Ainsi, BERT bat GPT-Neo sur toutes les métriques de prédiction, en particulier sur le F1-score, accuracy, precision, et recall. De plus, BERT possède une meilleure généralisation et une efficacité bien plus grande à l'évaluation. D'un autre côté, GPT-Neo s'entraîne plus lentement (on a un énorme écart au niveau de samples_per_second). La plus faible précision et F1 de Neo peuvent aussi s'expliquer par le fait qu'il préfère générer du texte qui a du sens globalement, plutôt que de bien séparer les différentes classes de manière précise, ce qui est pourtant nécessaire pour une tâche de classification.

3.3 Résultats



(Figure 15) *Résultats par classe. On observe que beaucoup de classes sont totalement absentes des prédictions du modèle, ce qui indique des biais très importants. Malgré son nombre beaucoup plus important de paramètres que BERT, on obtient de meilleures performances pour les modèles précédents.*

Avec ce modèle, on obtient donc de moins bons résultats qu'avec le modèle Bert, qui est plus adapté dans un cadre de données médicales dont il faut capturer la structure contextuelle. À l'inverse, utiliser GPT-Neo dans ce contexte nécessite un formatage spécifique des entrées/sorties pour forcer le modèle à produire des paires token/étiquette, avec souvent moins de précision sur la structure des entités médicales.

Cela peut s'expliquer par un prétraining qui vaut uniquement sur la modélisation du langage causal (prédiction du token suivant), sans objectif bidirectionnel. Ainsi, cette tâche d'entrainement est moins adaptés aux tâches nécessitant une compréhension complète du contexte car il ne voit que le contexte gauche.

Etape 4 : Thème libre

4.1 Choix de la méthode

Pour cette étape, nous nous sommes d'abord concentrés sur l'optimisation des hyperparamètres et sur la recherche d'une loss d'entraînement qui puisse améliorer le modèle. Etant donné que GPT Néo prend beaucoup de temps à s'exécuter et ne nous a pas donné les résultats espérés, nous avons choisi de garder le modèle BERT.

D'un côté, optimiser les hyperparamètres est pertinent dans notre contexte, car on va pouvoir améliorer les performances du modèle en ajustant les paramètres qui influencent la précision et la recall, ce qui est nécessaire car les résultats actuels montrent des f1-scores très disparates entre TUIs (nous avons par exemple des f1-scores proches de 0 pour plusieurs classes). Cette optimisation pourrait réduire ces déséquilibres et augmenter l'efficacité du modèle sur plus de classes.

D'un autre côté, changer la loss utilisée par défaut permet de mieux gérer les déséquilibres de classes, par exemple en attribuant plus de poids aux classes rares, ce qui aide le modèle à mieux les apprendre et à éviter de favoriser excessivement les classes dominantes 'O'.

4.2 Mise-en-œuvre

Nous avons commencé par tester la loss pondérée pour rééquilibrer les classes et forcer le modèle à se concentrer sur des classes moins fréquentes.

Nous avons ensuite testé la focal loss qui affine la weighted loss en réduisant l'impact des exemples faciles, ce qui permet de focaliser l'apprentissage sur les exemples difficiles et les classes rares.

Méthode de Perte	Predict Loss	Overall Accuracy	F1 Score	Precision	Recall
Weighted Loss	1.706	0.817	0.567	0.466	**0.723**
Focal Loss	0.241	0.245	0.165	0.104	0.399
Balanced Focal Loss	0.953	0.879	**0.650**	**0.637**	0.664

La comparaison numérique de ces trois fonctions de perte montre que la balanced focal loss est efficace car elle permet d'augmenter fortement la précision du modèle, et ainsi augmenter le f1score. En revanche, la loss pondérée permet de rééquilibrer les classes du modèle, mais elle perd en précision puisque le modèle est peu entraîné sur ces classes-là. Cette étude met en évidence que la stratégie optimale n'est pas nécessairement de chercher à tout prix à compenser le déséquilibre, mais plutôt de trouver un bon compromis entre pondération et focus sur les exemples difficiles.

Ainsi, la fonction de perte balancée indique les meilleures métriques de performance, ce qui est cohérent ce qui est cohérent puisqu'elle combine un ajustement dynamique des pondérations avec un mécanisme de focalisation sur les erreurs, tout en tenant compte de la distribution réelle des classes lors de l'apprentissage. A noter que l'on observe un léger surajustement dans les graphiques pour cette loss, mais elle permet d'obtenir un bon compromis global.

4.3 Utilisation Optuna

Nous avons aussi cherché à optimiser les hyperparamètres de notre modèle parmi : learning_rate, batch_size et num_train_epochs.Nous avons utilisé Optuna car Optuna permet d'automatiser la recherche des meilleurs hyperparamètres de façon efficace grâce à des algorithmes intelligents. Cet outil peut aussi arrêter automatiquement les essais peu prometteurs grâce au "pruning", ce qui économise du temps de calcul.

Nous avons pu tester Optuna sur le modèle de l'étape 1 et ainsi obtenir de meilleurs scores globaux sur les entrées suivantes :

hyperparams = {

"learning_rate": trial.suggest_float("learning_rate", 1e-5, 5e-5, log=True),

"per_device_train_batch_size": trial.suggest_categorical("batch_size", [8,16,32]),

"num_train_epochs": trial.suggest_int("num_train_epochs", 5)

}

Méthode de Perte	Predict Loss	Overall Accuracy	F1 Score	Precision	Recall
Hyperparamètres par défaut	0.46558502316474915	0.8754544248374685	0.630784708249497	0.6139534883720931	0.6485647788983708
Meilleurs hyperparamètres	0.7004675269126892	**0.8824333289107071**	**0.6537089123042694**	**0.6371522094926351**	**0.6711490388759 59**

Pour num_train_epochs = 5, les meilleurs hyperparamètres sont :

{"learning_rate": 4.422745210823006e-05, "batch_size": 16, "num_train_epochs": 5}

Ces paramètres nous permettent ainsi d'améliorer considérablement notre modèle.

Discussion

Notre étude a permis d'identifier les points forts et les limitations inhérents aux méthodes d'annotation automatique de textes médicaux. En comparant différentes configurations d'hyperparamètres sur le modèle BERT-BASE-UNCASED, nous avons constaté que la configuration avec un taux d'apprentissage de 5e-5 associé à un scheduler cosine offrait le meilleur compromis entre stabilité et rapidité de convergence. Cette configuration a permis d'obtenir des métriques globales satisfaisantes, notamment en termes de F1-score et d'accuracy, malgré l'impact négatif de la surreprésentation de la classe « O ».

Cependant, cette approche présente plusieurs limites. D'une part, la distribution déséquilibrée du dataset – avec des classes rares ou même absentes dans certains splits – engendre des erreurs systématiques sur les catégories peu représentées. Ces erreurs, bien que compréhensibles d'un point de vue statistique, posent un risque majeur dans un contexte médical où une mauvaise annotation peut conduire à des erreurs sémantiques critiques. D'autre part, les imperfections dans les annotations initiales, notamment la non-exhaustivité des catégories UMLS et des erreurs dans le labelage, réduisent la capacité du modèle à généraliser de manière fiable.

Il est par ailleurs apparu que l'approche générative avec GPT-Neo ne parvient pas à rivaliser avec la performance de BERT dans cette tâche d'annotation. Tandis que GPT-Neo montre une capacité de génération de texte globale intéressante, son architecture auto-régressive ne facilite pas une prédiction fine et précise des étiquettes, ce qui se traduit par des scores de performance inférieurs.

Ces constats nous incitent à envisager plusieurs axes d'amélioration. Sur le plan des données, il serait pertinent de renforcer la qualité et la quantité des annotations, en particulier pour les classes sous-représentées, et d'envisager des stratégies de rééquilibrage telles que l'augmentation de données ou la pondération de la fonction de perte. Par ailleurs, l'implémentation d'algorithmes d'optimisation hyperparamétrique, par exemple via Optuna, pourrait permettre d'affiner davantage les réglages du modèle et ainsi d'améliorer sa robustesse. Enfin, des approches hybrides combinant les avantages d'architectures encoder-only et decoder-only pourraient être explorées pour pallier les limitations observées et améliorer la capture du contexte dans des domaines aussi sensibles que le médical.

Même si notre approche actuelle offre une base solide, ces réflexions soulignent l'importance d'une optimisation continue tant sur le plan des données que sur celui des stratégies d'entraînement pour assurer une annotation médicale de haute qualité, indispensable à la recherche d'information fiable dans ce domaine.

Conclusion

Ce projet nous a permis de démontrer la complexité inhérente au fine-tuning des modèles de langage pour l'annotation de textes médicaux. En expérimentant avec BERT-BASE-UNCASED et GPT-Neo, nous avons mis en évidence l'importance d'un réglage précis des hyperparamètres pour obtenir une convergence stable et de bonnes performances, ainsi que les défis liés au déséquilibre des classes dans les données. Notre travail démontre que, malgré des performances globales satisfaisantes avec BERT, des marges d'amélioration subsistent—notamment par l'enrichissement et le rééquilibrage du jeu de données ainsi que par l'optimisation de la fonction de perte.

Au-delà des résultats quantitatifs, ce projet a renforcé notre compréhension des mécanismes d'apprentissage et de la sensibilité des modèles aux variations des hyperparamètres. Pour aller plus loin, il conviendra d'explorer des stratégies automatisées d'optimisation et d'enrichir les données afin de réduire les erreurs d'annotation critiques dans un domaine aussi sensible que le médical.

En résumé, notre démarche ouvre des perspectives prometteuses pour l'amélioration continue des systèmes d'annotation automatisée, tout en soulignant la nécessité de poursuivre la recherche pour garantir une fiabilité optimale dans l'application aux données médicales.

Annexes

Étape 1

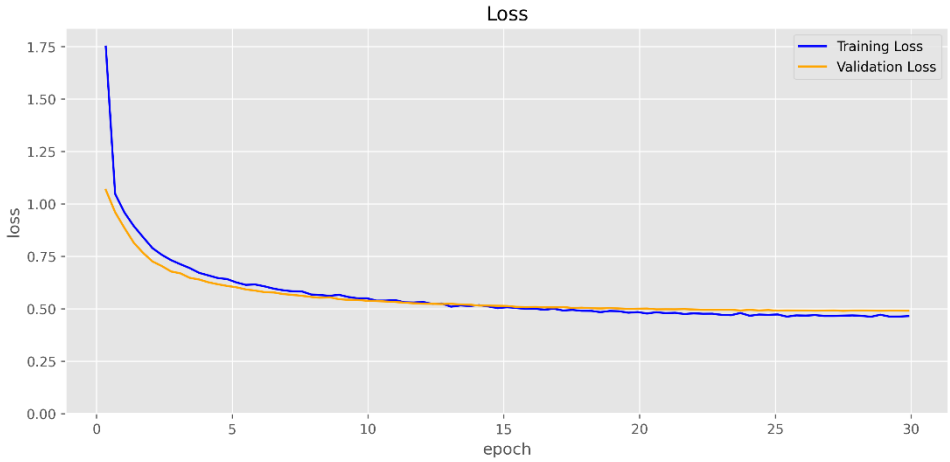
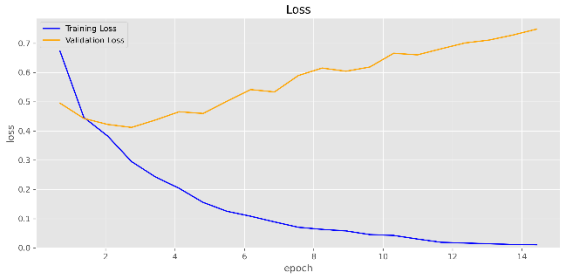
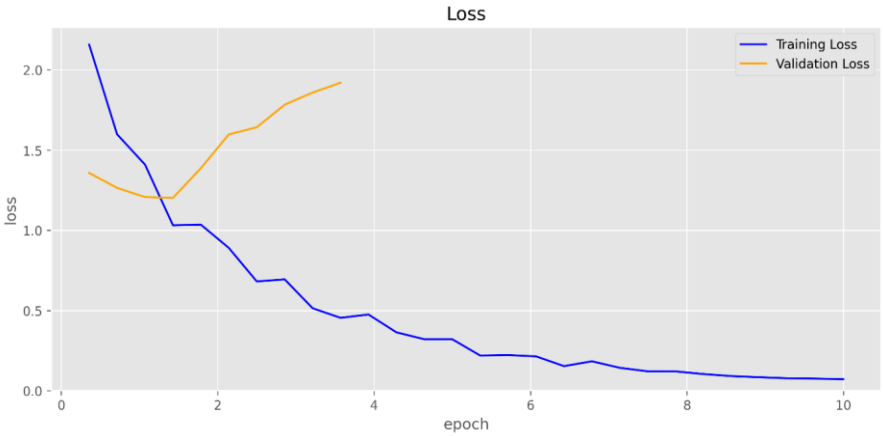


Figure 16 Entraînement du Modèle 2. On observe un under-fitting des données, car le coût d'entraînement ne semble pas converger vers 0 mais vers un plateau.

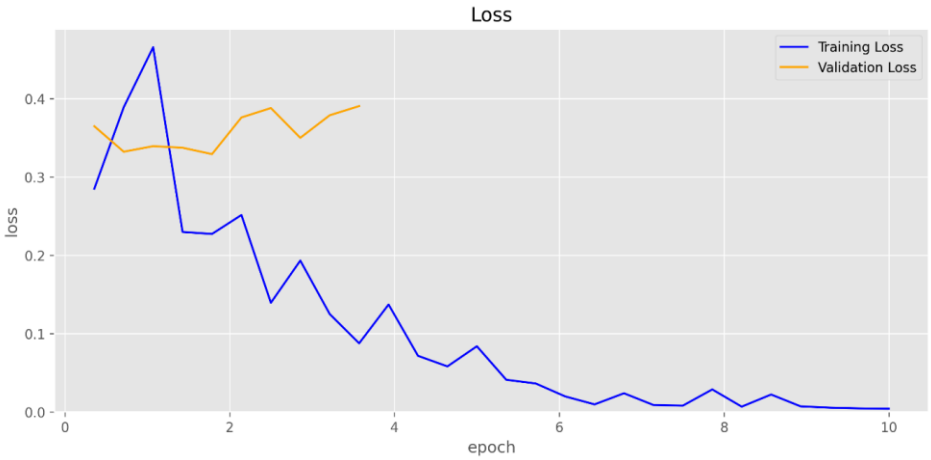


(Figure 17) Entraînement du modèle 3. On observe un over-fitting des données à partir de l'epoch 4 environ : les courbes d'entraînement et de validation sont fortement différentes.

2Weighted loss



Focal loss



Balanced loss