Telecom Nancy

Protein Subcellular Localization

Rapport de projet - MOM2 Biomed

Moh-Ouali AMROUNI
Rayane BELKALEM

13 février 2026

Table des matières

Résumé

Ce rapport décrit un pipeline d'apprentissage automatique que nous avons construit pour prédire où se retrouvent les protéines procaryotes dans (ou en dehors de) la cellule. À partir de séquences brutes d'acides aminés, nous extrayons deux représentations complémentaires — l'une via ESM-C, qui capture des informations évolutives, l'autre via ProstT5 3Di, qui encode des caractéristiques structurelles — puis nous les fusionnons au travers d'une architecture Transformer–BiLSTM avant de classifier parmi six classes de localisation. Malgré une exécution entièrement sur CPU portable, le modèle atteint 87.41 % de précision, 80.05 % de F1 macro et un coefficient de corrélation de Matthews (MCC) de 0.8227 sur 2 382 protéines de test.

1 Introduction

1.1 Contexte et motivation

L'endroit où une protéine se situe dans (ou hors de) la cellule détermine en grande partie sa fonction. Chez les procaryotes, on distingue six compartiments principaux : le cytoplasme, la membrane cytoplasmique, le périplasme, la membrane externe, l'espace extracellulaire et la paroi cellulaire. Chacun de ces compartiments est lié à des fonctions biologiques bien spécifiques, ce qui rend la connaissance de la localisation d'une protéine particulièrement utile — par exemple pour identifier de nouvelles cibles thérapeutiques ou comprendre le fonctionnement d'un pathogène.

Déterminer expérimentalement cette localisation reste long et coûteux. C'est pourquoi les approches computationnelles, qui prédisent la localisation directement à partir de la séquence d'acides aminés, ont pris beaucoup d'ampleur. Les modèles de langage protéique (pLM), entraînés sur d'énormes bases de données de séquences, produisent des embeddings suffisamment riches pour supporter une classification précise, sans avoir besoin de construire manuellement des descripteurs.

1.2 Position du problème

Notre objectif est simple : étant donné une séquence protéique procaryote au format FASTA, prédire à quel compartiment elle appartient parmi les six. Le projet s'accompagne de trois contraintes pratiques :

1. utiliser au moins deux modèles d'embeddings pré-entraînés (ESM-C et ProstT5) ;
2. aller au-delà d'un simple MLP — il faut une architecture plus intéressante ;
3. tout faire tourner sur du matériel grand public, uniquement en CPU (un portable ASUS TUF 15).

1.3 Organisation du rapport

La suite de ce rapport s'organise comme suit. La section 2 présente le jeu de données et ce que nous en avons appris lors de l'exploration. La section 3 détaille la méthodologie complète, de l'extraction des embeddings à la procédure d'entraînement. La section 4 présente les résultats de l'évaluation, et la section 5 discute ce que ces chiffres nous apprennent vraiment. Enfin, la section 6 donne les commandes exactes pour reproduire notre travail, et la section 7 conclut avec des pistes d'amélioration.

2 Données

2.1 Source et format

Les données proviennent du benchmark DeepLocPro et sont distribuées sous forme de fichier FASTA. Chaque en-tête encode quatre champs séparés par des pipes :

```
>PROTEIN_ID|LOCATION|GRAM_TYPE|PARTITION
```

Le fichier principal avec lequel nous travaillons (`graphpart_set.fasta`) contient **11 906** séquences protéiques réparties en six classes de localisation. Nous parsons ces en-têtes vers un fichier `metadata.csv` avec les colonnes `protein_id`, `sequence`, `label`, `gram_type` et `partition`, ce qui facilite grandement le traitement en aval.

2.2 Analyse exploratoire des données

Avant de construire le moindre modèle, nous avons exploré le jeu de données pour bien comprendre à quoi nous avions affaire. Trois aspects ressortent : le déséquilibre des classes, la grande variabilité des longueurs de séquences, et la composition en type Gram.

Distribution des classes. La première chose qui saute aux yeux, c'est le fort déséquilibre. Près de 58 % des protéines sont cytoplasmiques, et en y ajoutant les protéines de membrane cytoplasmique, on couvre déjà presque 80 % du jeu de données. À l'autre bout, Cell Wall ne compte que 87 échantillons — moins de 1 %. Ce type de déséquilibre doit être traité avec soin pendant l'entraînement, sous peine que le modèle apprenne simplement à prédire la classe majoritaire.

Classe	Nombre	Proportion
Cytoplasmic	6 885	57.8 %
Cytoplasmic Membrane	2 535	21.3 %
Extracellular	1 077	9.0 %
Outer Membrane	756	6.4 %
Periplasmic	566	4.8 %
Cell Wall	87	0.7 %

Table 1 – Distribution des classes dans le jeu de données complet.

Longueur des séquences. Les longueurs de protéines varient énormément — de 8 acides aminés seulement jusqu'à plus de 5 600. Le gros des données se situe entre 150 et 800 résidus environ, avec une médiane autour de 399 et une moyenne de 438. La distribution présente une longue traîne à droite, ce qui a des implications pratiques : les séquences longues consomment davantage de mémoire et de temps lors de l'extraction des embeddings. Nous avons choisi `max_len = 1000` comme seuil ; en regardant les percentiles (P95 = 878, P99 = 1 296), cette valeur conserve la grande majorité des séquences tout en restant calculable sur CPU.

Type Gram. La majorité des protéines proviennent de bactéries Gram-négatives (8 417), suivies des Gram-positives (3 206), avec une petite fraction d'archées (283). C'est un point à noter, car les compartiments membrane externe et périplasme sont spécifiques aux organismes Gram-négatifs — la répartition des types Gram influence donc directement la distribution des classes que l'on observe.

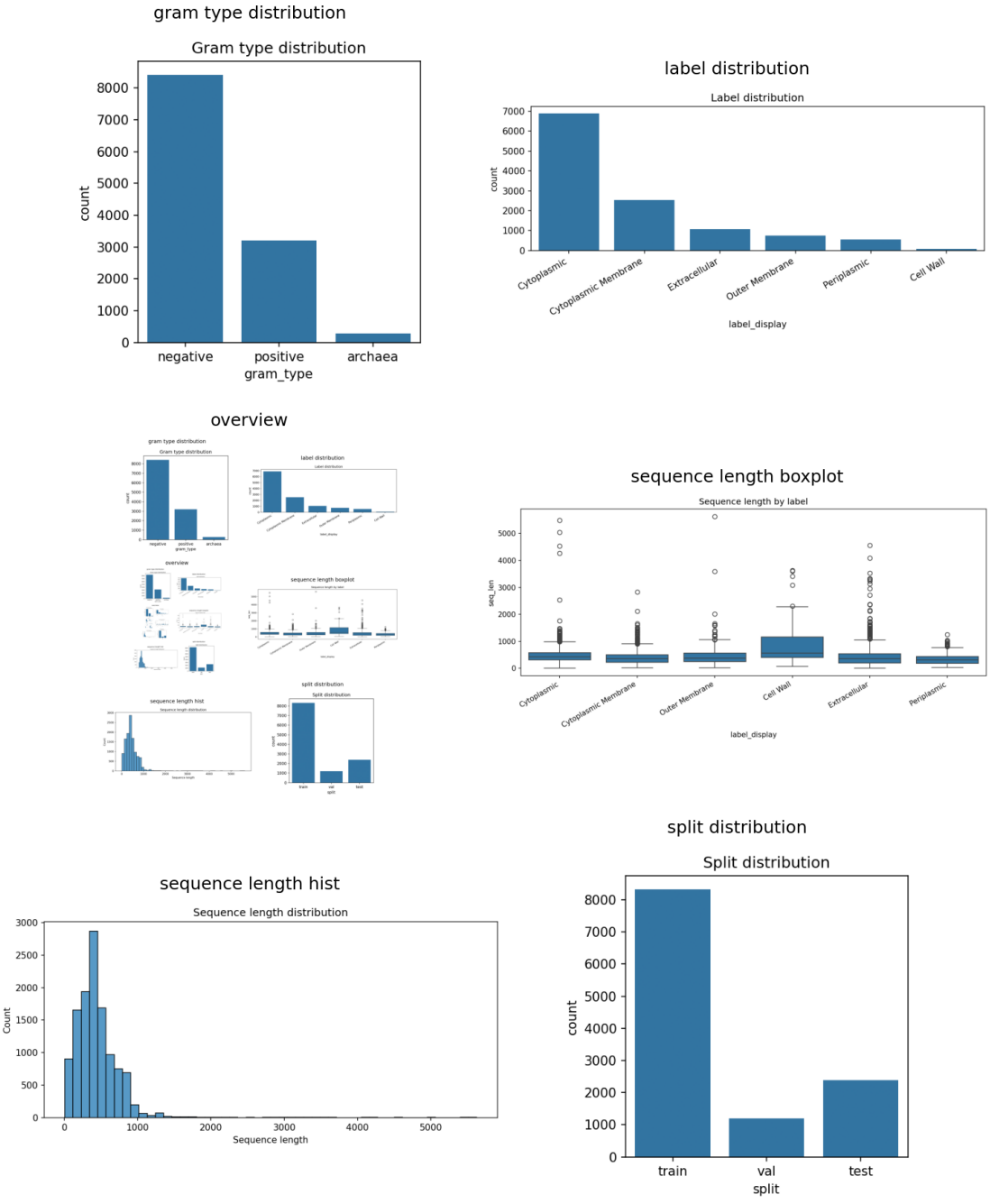

FIGURE 1 – Vue d'ensemble du jeu de données : distribution des classes, longueurs de séquences et tailles des splits.

La figure 1 rassemble plusieurs vues du jeu de données sur une même page. Le diagramme en barres des classes rend le déséquilibre immédiatement visible — la barre Cytoplasmic écrase toutes les autres. Le sous-graphe des longueurs montre bien le pro-

fil à longue traîne dont on a parlé : la plupart des protéines se concentrent entre 100 et 700 aa, mais quelques-unes dépassent largement les 2 000. Le diagramme des splits confirme que notre découpage stratifié maintient bien les proportions de classes dans chaque sous-ensemble.

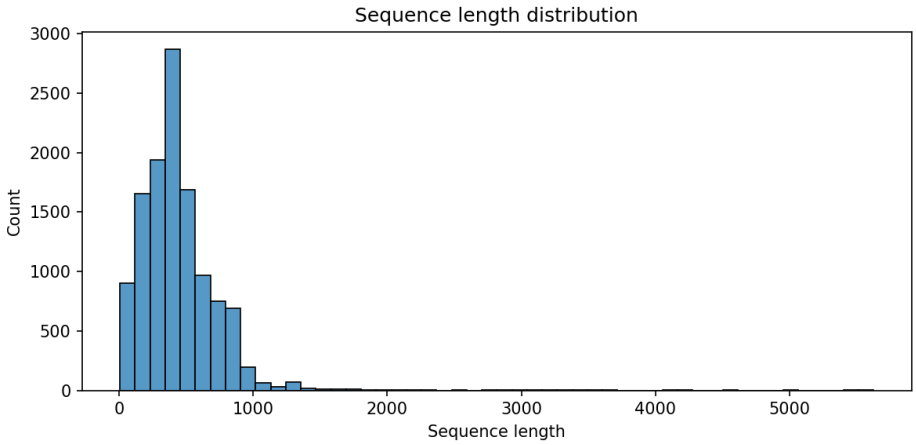

FIGURE 2 – Histogramme des longueurs de séquences protéiques (acides aminés).

La figure 2 se concentre sur la distribution des longueurs. L'histogramme confirme un profil asymétrique à droite, avec un pic autour de 300–500 aa. On observe une chute rapide après environ 800 aa, et seule une traîne fine dépasse les 1 000. Cela nous a confortés dans le choix de `max_len = 1000` : on ne perd qu'environ 3 % des séquences, un compromis acceptable au vu des économies de mémoire sur CPU.

2.3 Stratégie de découpage des données

Le jeu de données est divisé en **train** (8 333 séquences, 70 %), **validation** (1 191, 10 %) et **test** (2 382, 20 %), avec stratification sur la classe afin que chaque sous-ensemble reflète à peu près les proportions globales.

Un point subtil mérite d'être mentionné : des protéines avec des séquences très similaires peuvent se retrouver des deux côtés d'un split, ce qui gonfle artificiellement les performances. Pour s'en prémunir, nous avons implémenté une étape optionnelle de clustering via MMseqs2 à 30 % d'identité de séquence. Lorsqu'elle est activée, les clusters entiers restent dans le même split, empêchant les protéines quasi-identiques de fuiter entre les partitions.

Interprétation de la figure des splits. La figure 3 montre les tailles des splits obtenus. Les barres confirment le ratio 70/10/20, et grâce à la stratification, l'équilibre des classes est préservé dans chaque sous-ensemble. Le jeu de test (2 382 échantillons) est suffisamment grand pour obtenir des métriques par classe significatives, même pour les classes les plus petites.

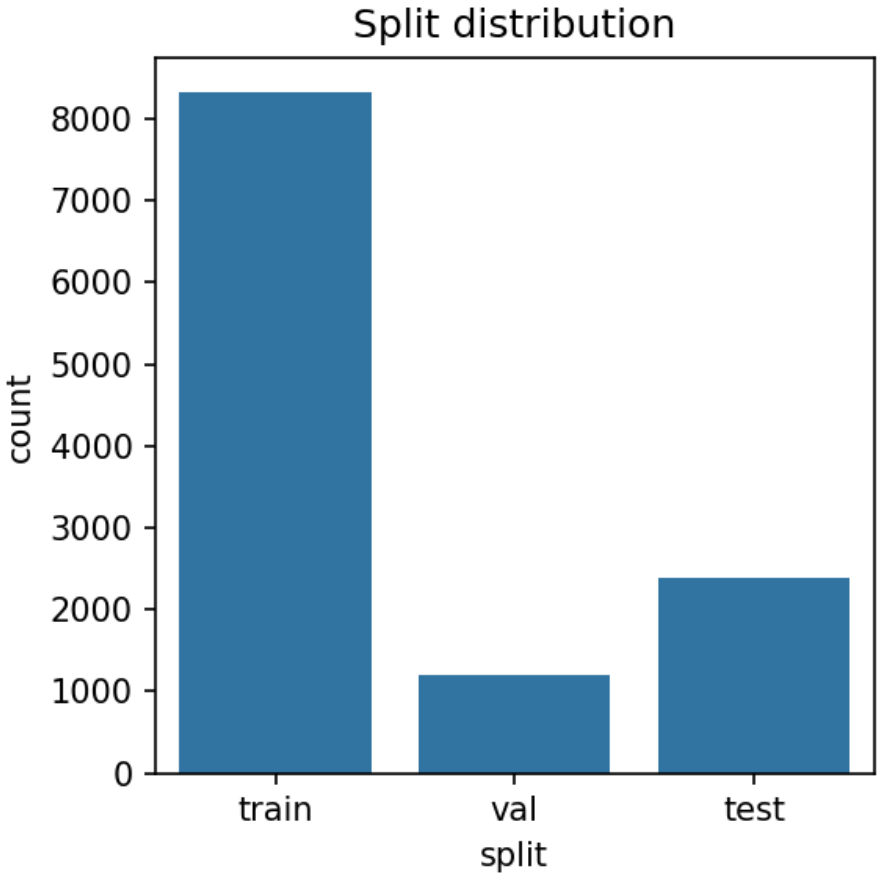

FIGURE 3 – Tailles des sous-ensembles train / validation / test.

3 Méthodologie

3.1 Vue d'ensemble du pipeline

À haut niveau, le pipeline se déroule en cinq étapes :

1. **Préparation des métadonnées** — on parse les en-têtes FASTA vers un CSV propre.
2. **Création des splits** — partitionnement stratifié et (optionnellement) sans fuite.
3. **Extraction des embeddings** — on passe ESM-C et ProstT5 sur chaque séquence.
4. **Entraînement** — on ajuste notre modèle de fusion Transformer–BiLSTM.
5. **Évaluation** — on mesure la précision, le F1 macro et le MCC sur le split test.

Chacune de ces étapes est détaillée ci-dessous.

3.2 Extraction des embeddings

On utilise deux modèles de langage protéique pré-entraînés, chacun capturant un type d'information différent sur la protéine.

3.2.1 ESM-C — embeddings évolutifs

ESM-C 300 M est un modèle d'EvolutionaryScale qui apprend des représentations à partir de larges collections de séquences protéiques alignées. Il est particulièrement doué pour capter les signaux de co-évolution — des résidus qui tendent à muter ensemble, ce qui reflète souvent des contraintes structurelles ou fonctionnelles.

En pratique, on tokenise chaque protéine, on la passe dans le modèle, puis on agrège les états cachés par résidus avec un pooling attentionnel pondéré. On obtient ainsi un unique vecteur de **960 dimensions** par protéine. Sur CPU, on traite les séquences par lots de 16, ce qui garde la consommation mémoire raisonnable.

3.2.2 ProstT5 3Di — embeddings structurels

ProstT5 vient de Rostlab et s'appuie sur l'encodeur T5. Son originalité réside dans l'alphabet structurel 3Di : plutôt que de prédire l'acide aminé suivant, il apprend à traduire les séquences en états structurels.

On préfixe chaque séquence avec le token `<AA2fold>` (les acides aminés sont séparés par des espaces), on la passe dans l'encodeur, puis on moyenne les états cachés des positions d'acides aminés — en excluant le token de préfixe et le padding. Le résultat est un vecteur de **1024 dimensions** par protéine.

3.2.3 Contraintes de calcul

Un défi pratique majeur : nous n'avions pas de GPU. Tout a tourné sur un portable ASUS TUF 15, CPU uniquement.

ProstT5 est assez gourmand en mémoire, on a donc dû activer le *disk offload* via la bibliothèque `accelerate` et se limiter à traiter une séquence à la fois (`batch = 1`). On a aussi plafonné le budget mémoire à 6 Go pour éviter le swapping. ESM-C est plus léger et peut gérer des lots de 16 sans problème.

On a fixé `max_len = 1000` acides aminés pour les deux modèles. Comme l'analyse exploratoire l'a montré, cela couvre environ 97 % du jeu de données — un bon compromis entre conservation de l'information et temps de calcul. Les poids des modèles Hugging Face sont cachés localement après le premier téléchargement.

3.3 Architecture du modèle

Nous utilisons ce que nous appelons le `TransformerLSTMFusionModel`. L'idée est de prendre les deux vecteurs d'embeddings de taille fixe, de laisser un Transformer déterminer comment ils interagissent, puis d'affiner le tout avec un BiLSTM avant la classification finale. On décrit chaque composant ci-dessous.

Le modèle propose une architecture plus originale que l'article : embeddings → Transformer → BiLSTM → MLP.

— projection linéaire des deux embeddings vers une dimension commune ;
— traitement des modalités comme tokens d'un Transformer encoder ;
— passage dans un BiLSTM pour modéliser les interactions ordonnées entre tokens ;
— fusion CLS + attention-pooling LSTM puis MLP de classification.

3.3.1 Couches de projection

Puisque les deux embeddings vivent dans des espaces différents (960 D pour ESM-C, 1024 D pour ProstT5), on les projette d'abord dans un espace commun de 512 dimensions. Chaque projection se compose de :

$$\text{Linear} \;\rightarrow\; \text{LayerNorm} \;\rightarrow\; \text{GELU} \;\rightarrow\; \text{Dropout}(0.3)$$

On a choisi GELU plutôt que ReLU parce que ça fonctionne généralement mieux dans les architectures de type Transformer.

3.3.2 Encodeur Transformer

Une fois les deux embeddings dans le même espace, on les traite comme des tokens dans une courte séquence. On ajoute un token apprenable `[CLS]` en tête, ce qui donne une entrée de 3 tokens : $[\texttt{CLS},\ \mathbf{e}_{\text{ESM-C}},\ \mathbf{e}_{\text{ProstT5}}]$.

Pour aider le modèle à distinguer les tokens, on ajoute des embeddings de type de token apprenables (un par position). La séquence passe ensuite dans un encodeur Transformer à 2 couches, 4 têtes d'attention et activation GELU, suivi d'une normalisation de couche.

Le rôle du Transformer ici est de permettre aux deux modalités de s'attendre mutuellement — en gros, d'apprendre comment les informations évolutives et structurelles doivent être combinées pour chaque protéine.

3.3.3 BiLSTM et pooling attentionnel

Après le Transformer, on passe la séquence complète des 3 tokens encodés dans un LSTM bidirectionnel à une couche (taille cachée 256, soit une sortie de 512 D à cause des deux directions).

On applique ensuite un mécanisme de pooling attentionnel apprenable sur les sorties du LSTM :

$$\alpha_t = \text{softmax}\Big(\mathbf{w}^\top \tanh(\mathbf{W}\,\mathbf{h}_t)\Big), \qquad \mathbf{s} = \sum_t \alpha_t\,\mathbf{h}_t$$

L'intuition est que certains tokens sont plus informatifs que d'autres, et le modèle devrait apprendre lesquels privilégier. Le vecteur agrégé par attention $\mathbf{s}$ (512 D) est concaténé avec la sortie `[CLS]` du Transformer (également 512 D), produisant une représentation de 1024 dimensions.

3.3.4 Tête de classification

Enfin, le vecteur fusionné de 1024 D passe dans un MLP à trois couches :

$$1024 \xrightarrow{\text{ReLU}} 512 \xrightarrow{\text{ReLU}} 256 \rightarrow 6$$

avec du dropout entre chaque couche. La sortie est constituée de six logits bruts — un par classe de localisation.

Résumé de l'architecture

Le tableau 2 rassemble tous les choix architecturaux importants. Le modèle est volontairement compact (∼964 K paramètres) pour tenir confortablement sur CPU.

Composant	Configuration
Dimension d'entrée ESM-C	960
Dimension d'entrée ProstT5	1024
Dimension cachée (d)	512
Têtes Transformer	4
Couches Transformer	2
Activation Transformer	GELU
Dimension cachée LSTM	256
Couches LSTM	1 (bidirectionnel)
Dimension pooling attentionnel	512
MLP de classification	$1024 \to 512 \to 256 \to 6$
Dropout (toutes les couches)	0.3
Nb. approx. de paramètres	$\sim$964 K

TABLE 2 – Configuration de l'architecture TransformerLSTMFusionModel.

3.4 Procédure d'entraînement

3.4.1 Fonction de perte et gestion du déséquilibre

Vu à quel point la distribution des classes est déséquilibrée (Cell Wall représente moins de 1 % des données), une cross-entropy naïve verrait à peine les classes minoritaires. On attaque le problème sous deux angles :

— **Perte focale** (Lin et al., 2017) avec $\gamma = 2.0$: cette variante de la cross-entropy réduit le poids des exemples déjà bien classifiés et met davantage de gradient sur les cas difficiles. Formellement :

$$\mathcal{L}_{\text{FL}} = -\alpha_t \, (1 - p_t)^{\gamma} \, \log(p_t)$$

— **Échantillonnage équilibré** : un `WeightedRandomSampler` sur-échantillonne les classes minoritaires pour que chaque batch voie un mélange de classes à peu près uniforme, indépendamment de la distribution réelle.

Les poids de classe α_t sont fixés à l'inverse de la fréquence de chaque classe dans le jeu d'entraînement.

3.4.2 Optimiseur et taux d'apprentissage

On utilise AdamW avec un taux d'apprentissage initial de 10^{-4} et une régularisation de poids de 0.01. Le taux d'apprentissage suit un recuit cosinus, décroissant progressivement jusqu'à 10^{-6}, avec un échauffement linéaire de 5 époques au début pour laisser le modèle se stabiliser avant que la décroissance ne commence.

3.4.3 Régularisation

On applique du dropout à 0.3 à peu près partout : dans les couches de projection, à l'intérieur de l'encodeur Transformer et dans la tête de classification. Les normes de gradient sont écrêtées à 1.0 pour éviter les instabilités. L'entraînement dure au maximum 100 époques, mais en pratique l'arrêt prématuré (patience de 15 époques, surveillé sur le F1 macro de validation) se déclenche bien avant.

Hyperparamètres d'entraînement

Hyperparamètre	Valeur
Optimiseur	AdamW
Taux d'apprentissage	10^{-4}
Régularisation de poids	0.01
Betas	(0.9, 0.999)
Planificateur	Recuit cosinus
Époques d'échauffement	5
Taux d'apprentissage minimal	10^{-6}
Époques max	100
Taille de batch	32
Fonction de perte	Perte focale ($\gamma = 2.0$)
Poids de classe	Inverse de la fréquence
Échantillonnage équilibré	Oui
Écrêtage du gradient	1.0 (norme max)
Arrêt prématuré	Patience 15 (F1 macro)
Graine aléatoire	42

TABLE 3 – Hyperparamètres d'entraînement.

4 Résultats

L'entraînement s'est arrêté via l'arrêt prématuré, et nous avons évalué le meilleur checkpoint sur le jeu de test (2 382 protéines).

4.1 Métriques globales

	Précision	F1 macro	MCC
Jeu de test	0.8741	0.8005	0.8227

TABLE 4 – Métriques d'évaluation globales sur le split test.

Dans l'ensemble, les chiffres sont encourageants. 87 % de précision, c'est solide, et le MCC de 0.82 nous indique que le modèle ne se contente pas de surfer sur la classe majoritaire. Le F1 macro est un peu plus bas à 80 %, ce qui est attendu puisqu'il donne un poids égal à toutes les classes, y compris les minoritaires difficiles.

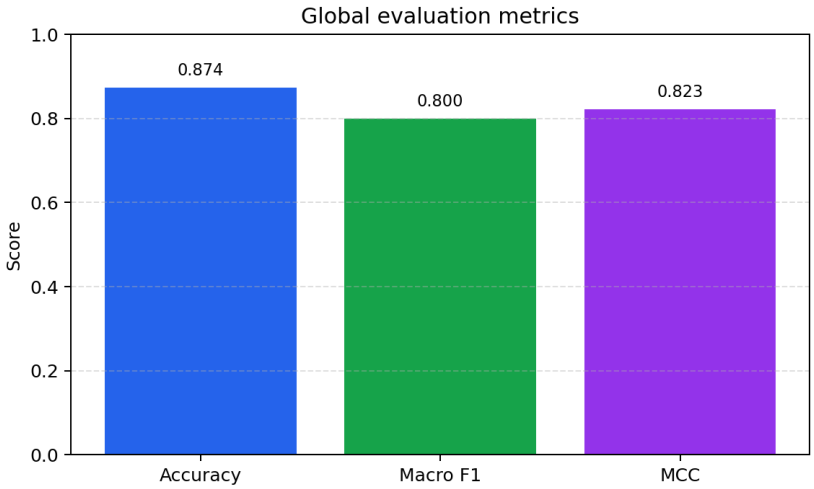

FIGURE 4 – Diagramme en barres des métriques globales. Les trois métriques dépassent 0.80, la précision étant légèrement en tête du fait de l'effet de la classe majoritaire.

4.2 Analyse par classe

Classe	Précision	Rappel	F1
Cytoplasmic	0.901	0.846	0.873
Cytoplasmic Membrane	0.500	0.750	0.600
Periplasmic	0.933	0.925	0.929
Outer Membrane	0.776	0.839	0.806
Extracellular	0.706	0.818	0.758
Cell Wall	0.857	0.818	0.837

TABLE 5 – Précision, rappel et F1 par classe sur le jeu de test.

En regardant les chiffres par classe, le résultat le plus frappant est Periplasmic avec un F1 de 0.93 — cela suggère que les protéines périplasmiques ont des signatures assez distinctes que l'approche double-embedding capture bien. Cell Wall est aussi une bonne surprise à 0.84, sachant qu'il n'y a que 87 exemples d'entraînement. En revanche, Cytoplasmic Membrane est en retrait à 0.60. Nous pensons que cela vient du fait que les protéines transmembranaires partagent des caractéristiques avec les deux classes cytoplasmique et membranaire, ce qui rend leur séparation plus délicate.

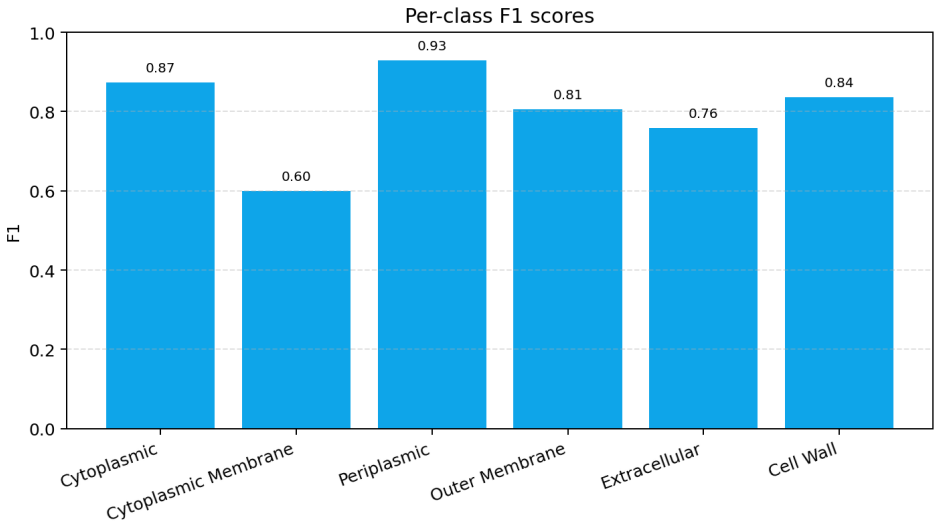

FIGURE 5 – F1 par classe de localisation. Periplasmic est en tête tandis que Cytoplasmic Membrane ressort clairement comme la classe la plus difficile.

4.3 Matrice de confusion

FIGURE 6 – Matrice de confusion

5 Discussion

5.1 Ce que les chiffres nous disent

À première vue, 87 % de précision peut sembler suffisant, mais il vaut la peine d'y regarder de plus près. Comme les protéines cytoplasmiques représentent près de 58 % du jeu de test, un modèle qui prédit toujours « Cytoplasmic » obtiendrait déjà environ 58 % de précision sans rien apprendre d'utile. Le fait que notre F1 macro atteigne 80 % et que le MCC soit à 0.82 raconte une histoire plus complète : le modèle discrimine réellement les six classes, il ne se contente pas de miser sur la majoritaire.

L'écart entre la précision (87 %) et le F1 macro (80 %) est en soi parlant — il vient des classes faibles qui tirent la moyenne macro vers le bas, exactement ce à quoi on s'attend avec un tel déséquilibre.

5.2 Éclairage par classe

Quelques classes méritent qu'on s'y attarde :

— **Periplasmic** ($F1 = 0.929$) : le meilleur résultat, et de loin. Les protéines périplasmiques des bactéries Gram-négatives possèdent typiquement des peptides signal et des motifs structurels spécifiques qui laissent visiblement une empreinte nette dans les deux types d'embeddings.

— **Cell Wall** ($F1 = 0.837$) : un résultat rassurant. Ça montre qu'une stratégie d'entraînement bien choisie peut compenser des effectifs très réduits.

— **Cytoplasmic Membrane** ($F1 = 0.600$) : le point faible évident. C'est la deuxième plus grande classe (2 535 échantillons), donc le problème n'est pas le manque de données. L'explication probable est biologique : les protéines transmembranaires chevauchent la membrane, ce qui signifie qu'elles ont des domaines cytoplasmiques importants. Au niveau des embeddings, cela brouille la frontière entre « cytoplasmique » et « membranaire », rendant les deux classes difficiles à séparer.

— **Outer Membrane** ($F1 = 0.806$) et **Extracellular** ($F1 = 0.758$) : des résultats corrects mais perfectibles. Ces deux compartiments sont reliés par les voies de sécrétion, il est donc logique que certaines protéines soient confondues entre les deux.

5.3 Limites

Nous tenons à être transparents sur ce que ce travail ne couvre *pas* :

— Tronquer les séquences à 1 000 acides aminés signifie qu'environ 3 % des protéines sont coupées. Pour les plus longues (jusqu'à 5 627 aa), on perd potentiellement de l'information importante.

— Travailler sur CPU était une nécessité, pas un choix. Cela a limité à la fois la taille du modèle et les expériences que nous avons pu tenter.

— Tout repose sur un unique découpage train/test. Une vraie validation croisée — idéalement avec un partitionnement respectant l'homologie — donnerait des chiffres plus fiables.

6 Reproductibilité

Pour reproduire l'intégralité du pipeline depuis zéro, il suffit d'exécuter les commandes suivantes dans l'ordre :

```
python scripts/prepare_metadata.py \
    --fasta data/raw/graphpart_set.fasta \
    --output data/processed/metadata.csv

python scripts/prepare_splits.py \
    --metadata data/processed/metadata.csv \
    --output data/processed/splits.csv

python -m src.embeddings.fetch_embeddings \
    --embed2_backend prostt5 --max_len 1000 \
    --prost_batch 1 --prost_offload_dir data/interim/offload \
    --prost_max_memory 6GB

python scripts/train.py --config configs/default.yaml

python scripts/evaluate.py \
    --checkpoint results/checkpoints/best_model.pt \
    --config configs/default.yaml

make eda
make results-figures
```

7 Conclusion et perspectives

Dans ce projet, nous avons construit un pipeline de bout en bout qui prend des séquences protéiques brutes et prédit leur localisation subcellulaire parmi six classes. L'idée centrale est de combiner deux visions complémentaires de chaque protéine — les patterns évolutifs d'ESM-C et les caractéristiques structurelles de ProstT5 3Di — au travers d'une architecture de fusion Transformer–BiLSTM. Malgré une exécution entièrement sur CPU, le modèle atteint 87 % de précision et 80 % de F1 macro, la combinaison perte focale + échantillonnage équilibré s'étant montrée efficace même pour les classes sévèrement sous-représentées.

Cela dit, il reste de la marge. Avec un GPU, on pourrait supprimer le plafond de longueur et expérimenter avec des modèles d'embeddings plus grands ou plus récents. Des études d'ablation nous aideraient à comprendre ce que chaque composant de l'architecture apporte réellement. Côté évaluation, une vraie validation croisée avec clustering par homologie (via MMseqs2 à 30 % d'identité) donnerait des estimations de performance plus serrées et plus fiables. Et bien sûr, explorer des Transformers plus profonds ou des stratégies de pooling différentes pourrait encore améliorer les chiffres — surtout pour la classe Cytoplasmic Membrane, qui reste notre point faible.