Telecom Nancy

Protein Subcellular Localization

Project Report - MOM2 Biomed

Moh-Ouali AMROUNI
Rayane BELKALEM

February 13, 2026

Contents

Abstract

This report describes a machine-learning pipeline we built for predicting where prokaryotic proteins end up inside (or outside) the cell. Starting from raw amino-acid sequences, we extract two complementary representations—one from ESM-C, which captures evolutionary information, and one from ProstT5 3Di, which encodes structural features—then fuse them through a custom Transformer–BiLSTM architecture before classifying into six localization classes. Despite running everything on a laptop CPU, the model reaches 87.41 % accuracy, 80.05 % macro-F1, and a Matthews Correlation Coefficient of 0.8227 on 2 382 held-out proteins.

1 Introduction

1.1 Context and motivation

Where a protein sits inside (or outside) a cell largely determines what it does. In prokaryotes, there are six main compartments a protein can be associated with: the cytoplasm, the cytoplasmic membrane, the periplasm, the outer membrane, the extracellular space, and the cell wall. Each of these compartments is tied to specific biological functions, so knowing a protein's localization can be valuable—for instance when looking for new drug targets or trying to understand how a pathogen operates.

Figuring out localization experimentally is slow and costly. That is why computational approaches that predict it directly from the amino-acid sequence have gained a lot of traction. In particular, protein language models (pLMs) trained on massive sequence databases have been shown to produce embeddings that are rich enough to support accurate classification, without requiring any hand-crafted features.

1.2 Problem statement

Our goal is straightforward: given a prokaryotic protein sequence in FASTA format, predict which of the six compartments it belongs to. The project comes with three practical constraints:

1. we must use at least two pre-trained embedding models (ESM-C and ProstT5);
2. the model has to go beyond a plain MLP—we need a more interesting architecture;
3. everything must run on CPU-only hardware (an ASUS TUF 15 laptop).

1.3 Report outline

The rest of this report is organized as follows. Section 2 introduces the dataset and what we learned from exploring it. Section 3 walks through the full methodology, from embedding extraction to the training procedure. Section 4 presents the evaluation results, and Section 5 discusses what the numbers actually tell us. Finally, Section 6 gives the exact commands needed to reproduce our work, and Section 7 wraps up with conclusions and ideas for future improvements.

2 Data

2.1 Source and format

The data come from the DeepLocPro benchmark and are distributed as a FASTA file. Each header encodes four fields separated by pipes:

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

The main file we work with (`graphpart_set.fasta`) contains **11 906** protein sequences spanning six localization classes. We parse these headers into a `metadata.csv` file with columns `protein_id`, `sequence`, `label`, `gram_type`, and `partition`, which makes downstream processing much easier.

2.2 Exploratory data analysis

Before building any model, we explored the dataset to understand what we were working with. Three aspects stand out: the class imbalance, the wide range of sequence lengths, and the gram-type composition.

Class distribution. The first thing that jumps out is the heavy class imbalance. Nearly 58 % of the proteins are cytoplasmic, and together with cytoplasmic membrane proteins they already account for almost 80 % of the dataset. On the other end, Cell Wall has only 87 samples—less than 1 %. This kind of imbalance needs to be handled carefully during training, otherwise the model will just learn to predict the majority class.

Class	Count	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: Class distribution across the full dataset.

Sequence length. Protein lengths vary enormously—from as short as 8 amino acids to over 5 600. The bulk of the data sits between roughly 150 and 800 residues, with a median around 399 and a mean of 438. The distribution has a long right tail, which matters from an engineering standpoint: longer sequences eat up more memory and compute during embedding extraction. We chose `max_len = 1000` as a practical cutoff; looking at the percentiles (P95 = 878, P99 = 1 296), this keeps the vast majority of sequences intact while staying feasible on CPU.

Gram type. Most proteins come from Gram-negative bacteria (8 417), followed by Gram-positive bacteria (3 206), with a small fraction from archaea (283). This is worth noting because outer membrane and periplasmic compartments are specific to Gram-negative organisms, so the gram-type distribution directly shapes the class distribution we see.

Interpretation of the overview figure. Figure 1 brings together several views of the dataset on a single page. The label bar chart makes the imbalance immediately visible—the Cytoplasmic bar dwarfs everything else. The sequence-length subplot shows the long-tailed shape we mentioned: most proteins cluster between 100 and 700 aa, but a handful stretch well past 2 000. The split bar chart confirms that our stratified split keeps the class proportions consistent across train, validation, and test sets.

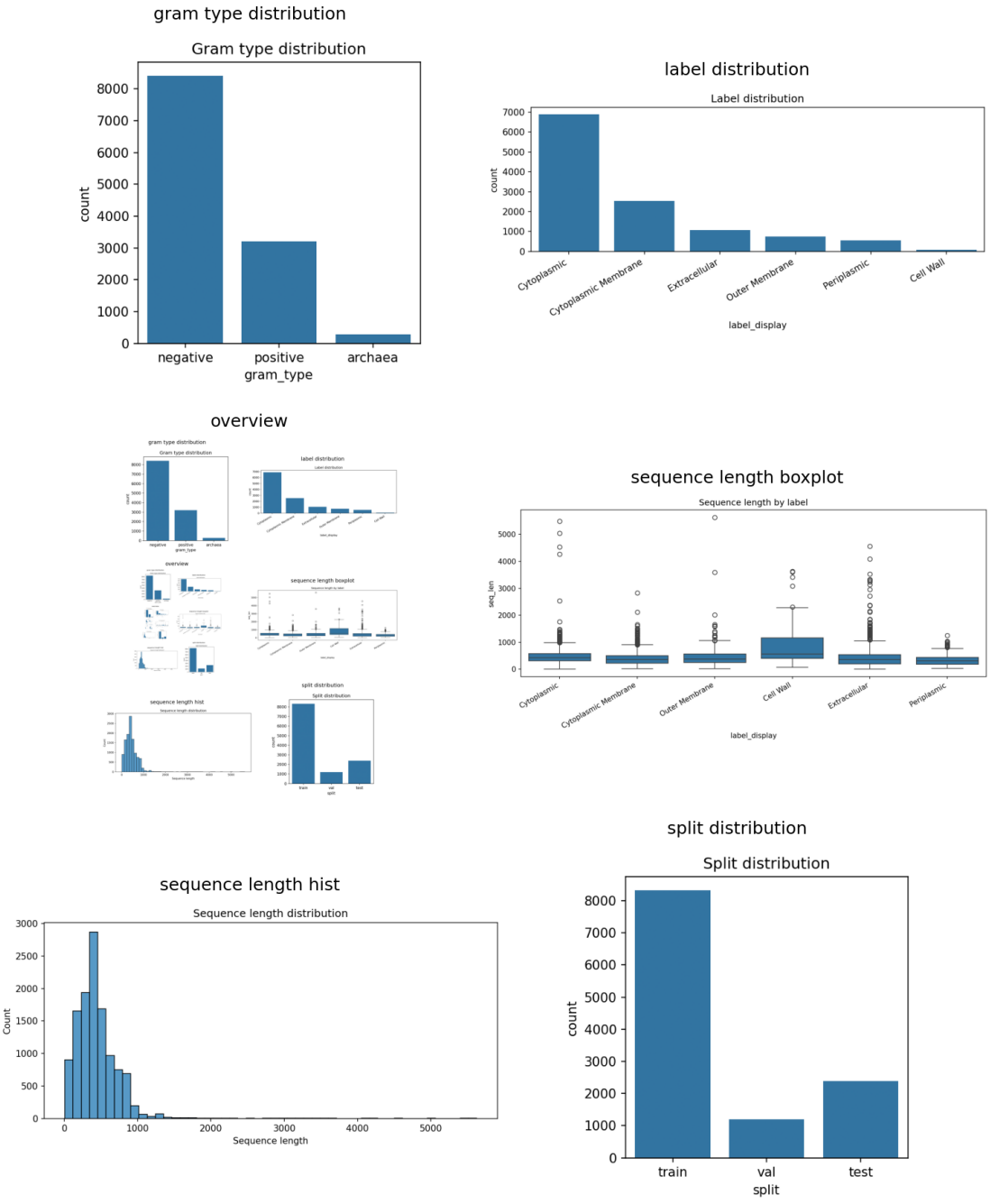

Figure 1: Overview of the dataset: label distribution, sequence lengths, and split sizes at a glance.

Interpretation of the length histogram. Figure 2 zooms in on the sequence-length distribution. The histogram confirms a right-skewed profile with a peak around 300–

500 aa. There is a rapid drop-off after roughly 800 aa, and only a thin tail extends past 1 000. This reassured us that capping at `max_len = 1000` does not discard a significant portion of the data—we lose roughly 3 % of the sequences, which is an acceptable trade-off given the memory savings on CPU.

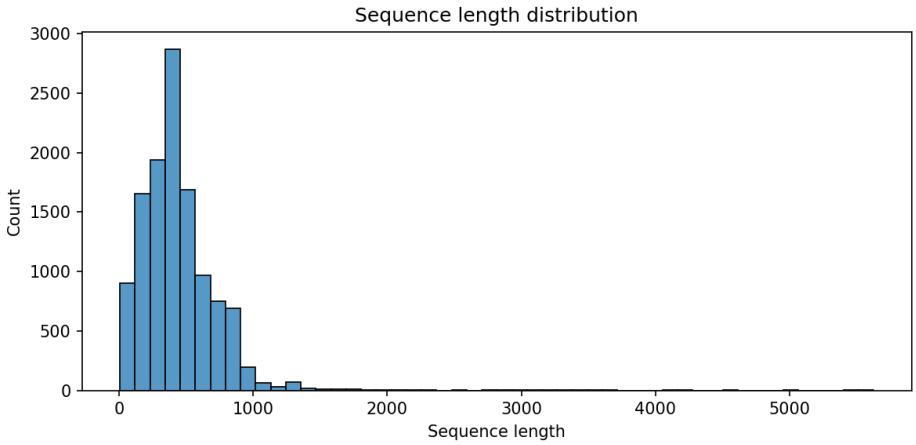

Figure 2: Histogram of protein sequence lengths (amino acids).

2.3 Data splitting strategy

We split the dataset into **train** (8 333 sequences, 70 %), **validation** (1 191, 10 %), and **test** (2 382, 20 %), using stratification on the class label so that each subset roughly mirrors the overall class proportions.

One subtlety worth mentioning: proteins with very similar sequences can end up on both sides of a split, which inflates performance. To guard against this, we implemented an optional clustering step using MMseqs2 at a 30 % sequence-identity threshold. When enabled, entire clusters stay together in the same split, preventing near-duplicate proteins from leaking across partitions.

Interpretation of the split figure. Figure 3 shows the resulting split sizes. The bars confirm the 70/10/20 ratio, and thanks to stratification the class balance is preserved within each subset. The test set (2 382 samples) is large enough to give us meaningful per-class metrics, even for the smaller classes.

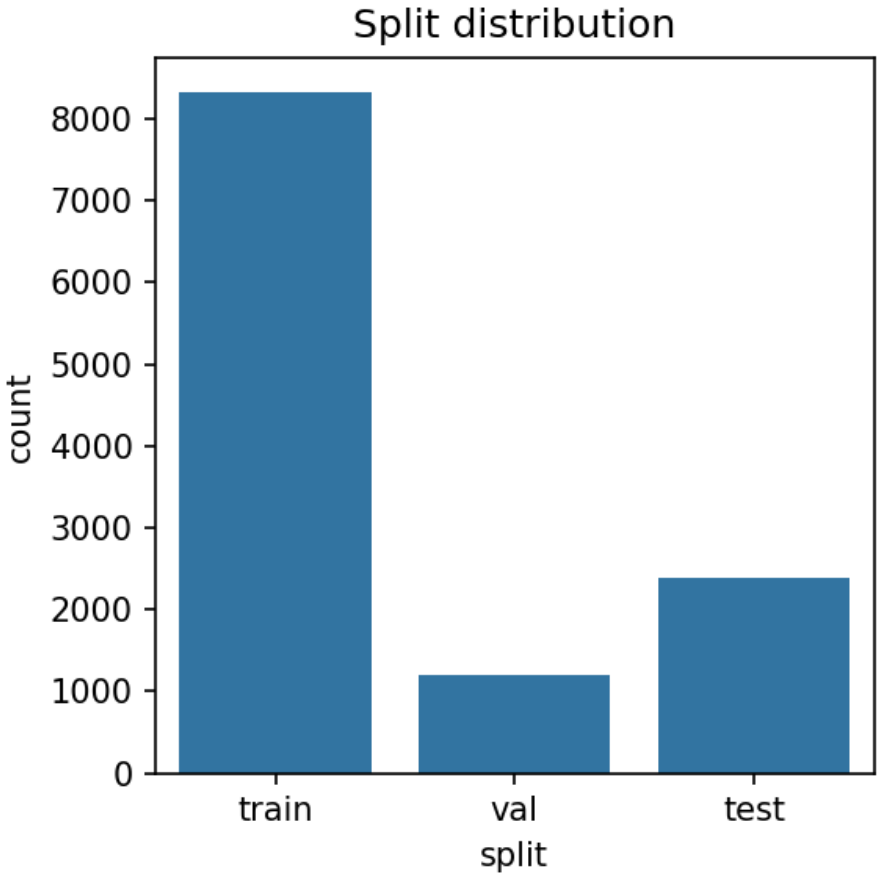

Figure 3: Train / Validation / Test split sizes.

3 Methodology

3.1 Pipeline overview

At a high level, the pipeline goes through five stages:

1. **Metadata preparation** – we parse the FASTA headers into a clean CSV.
2. **Split creation** – stratified and (optionally) leakage-safe partitioning.
3. **Embedding extraction** – we run ESM-C and ProstT5 on every sequence.
4. **Training** – we fit our Transformer–BiLSTM fusion model.
5. **Evaluation** – we measure accuracy, macro-F1, and MCC on the test set.

Each of these stages is detailed below.

3.2 Embedding extraction

We use two pre-trained protein language models, each capturing a different kind of information about the protein.

3.2.1 ESM-C – evolutionary embeddings

ESM-C 300 M is a model from EvolutionaryScale that learns representations from large collections of aligned protein sequences. It is good at picking up co-evolutionary signals—

residues that tend to mutate together, which often reflects structural or functional constraints.

In practice, we tokenize each protein, pass it through the model, and then aggregate the per-residue hidden states with attention-weighted pooling. This gives us a single **960-dimensional** vector per protein. On CPU we process sequences in batches of 16, which keeps memory usage reasonable.

3.2.2 ProstT5 3Di – structural embeddings

ProstT5 comes from Rostlab and is based on the T5 encoder. Its originality lies in the 3Di structural alphabet: rather than predicting the next amino acid, it learns to translate sequences into structural states.

We prefix each sequence with the `<AA2fold>` token (amino acids are space-separated), run it through the encoder, and then mean-pool the hidden states of the amino-acid positions—excluding the prefix token and any padding. The result is a **1024-dimensional** vector per protein.

3.2.3 Compute constraints

A key practical challenge was that we had no GPU. Everything ran on an ASUS TUF 15 laptop, CPU-only.

ProstT5 is quite memory-hungry, so we had to activate disk offload through the `accelerate` library and limit ourselves to processing one sequence at a time (`batch = 1`). We also capped the memory budget at 6 GB to avoid swap-thrashing. ESM-C is lighter and can handle batches of 16 without issue.

We set `max_len = 1000` amino acids for both models. As the EDA showed, this covers about 97 % of the dataset, which felt like a good compromise between information retention and compute time. Hugging Face model weights are cached locally after the first download.

3.3 Model architecture

We use what we call the `TransformerLSTMFusionModel`. The idea is to take the two fixed-size embedding vectors, let a Transformer figure out how they relate to each other, then refine that with a BiLSTM before making the final classification. Below we describe each component.

The model proposes a more original architecture than a simple baseline: embeddings → Transformer → BiLSTM → MLP.

- linear projection of the two embeddings into a common dimension;
- treatment of the modalities as tokens of a Transformer encoder;
- passage through a BiLSTM to model ordered interactions between tokens;
- fusion of CLS + attention-pooling LSTM then classification MLP.

3.3.1 Projection layers

Since the two embeddings live in different spaces (960 D for ESM-C, 1024 D for ProstT5), we first project them into a common 512-dimensional space. Each projection consists of:

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

We chose GELU over ReLU here because it tends to work better in Transformer-style architectures.

3.3.2 Transformer encoder

Once both embeddings are in the same space, we treat them as tokens in a short sequence. We prepend a learnable `[CLS]` token, giving us a 3-token input: $[\texttt{CLS},\ \mathbf{e}_{\text{ESM-C}},\ \mathbf{e}_{\text{ProstT5}}]$.

To help the model tell the tokens apart, we add learnable token-type embeddings (one for each position). The sequence then goes through a 2-layer Transformer encoder with 4 attention heads and GELU activation, followed by layer normalization.

The Transformer's role here is to let the two modalities attend to each other—essentially learning how evolutionary and structural information should be combined for each protein.

3.3.3 BiLSTM and attention pooling

After the Transformer, we pass the full 3-token encoded sequence through a single-layer bidirectional LSTM (hidden size 256, so the output is 512 D because of the two directions).

We then apply a learnable attention-pooling mechanism over the LSTM outputs:

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

The intuition is that some tokens are more informative than others, and the model should learn which ones to focus on. The attention-pooled vector $\mathbf{s}$ (512 D) is then concatenated with the `[CLS]` output from the Transformer (also 512 D), producing a 1024-dimensional representation.

3.3.4 Classification head

Finally, the 1024-D fused vector goes through a three-layer MLP:

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

with dropout between each layer. The output is six raw logits—one per localization class.

Architecture summary

Table 2 collects all the key architectural choices in one place. The model is deliberately compact ($\sim$964 K parameters) to fit comfortably on CPU.

Component	Configuration
ESM-C input dimension	960
ProstT5 input dimension	1024
Hidden dimension (d)	512
Transformer heads	4
Transformer layers	2
Transformer activation	GELU
LSTM hidden dimension	256
LSTM layers	1 (bidirectional)
Attention pooling dim	512
Classification MLP	$1024 \to 512 \to 256 \to 6$
Dropout (all layers)	0.3
Approx. total parameters	$\sim$964 K

Table 2: Architecture configuration of the TransformerLSTMFusionModel.

3.4 Training procedure

3.4.1 Loss function and class-imbalance handling

Given how skewed the class distribution is (Cell Wall is less than 1 % of the data), a naive cross-entropy loss would barely see the minority classes. We tackle this from two angles:

- **Focal Loss** (Lin et al., 2017) with $\gamma = 2.0$: this variant of cross-entropy down-weights samples the model already classifies well and puts more gradient on the hard ones. Formally:

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

- **Balanced sampling**: we use a `WeightedRandomSampler` that oversamples minority classes so that each batch sees a roughly uniform class mix, regardless of the true distribution.

The class weights α_t are set to the inverse of each class's frequency in the training set.

3.4.2 Optimizer and learning-rate schedule

We use AdamW with an initial learning rate of 10^{-4} and a weight decay of 0.01. The learning rate follows a cosine annealing schedule, gradually decaying to 10^{-6} over the course of training, with a 5-epoch warmup at the start to let the model stabilize before the decay kicks in.

3.4.3 Regularization

We apply dropout at a rate of 0.3 pretty much everywhere: in the projection layers, inside the Transformer encoder, and in the classification head. Gradient norms are clipped to 1.0 to prevent instabilities. Training runs for up to 100 epochs, but in practice early stopping (patience of 15 epochs, monitored on validation macro-F1) usually kicks in well before that.

Training hyperparameters

Hyperparameter	Value
Optimizer	AdamW
Learning rate	10^{-4}
Weight decay	0.01
Betas	(0.9, 0.999)
Scheduler	Cosine annealing
Warmup epochs	5
Minimum learning rate	10^{-6}
Max epochs	100
Batch size	32
Loss function	Focal Loss ($\gamma = 2.0$)
Class weights	Inverse frequency
Balanced sampler	Yes
Gradient clipping	1.0 (max norm)
Early stopping	Patience 15 (macro-F1)
Random seed	42

Table 3: Training hyperparameters.

4 Results

Training stopped via early stopping, and we evaluated the best checkpoint on the held-out test set (2 382 proteins).

4.1 Global metrics

	Accuracy	Macro-F1	MCC
Test set	0.8741	0.8005	0.8227

Table 4: Global evaluation metrics on the test split.

Overall, the numbers are encouraging. 87 % accuracy is solid, and the MCC of 0.82 tells us that the model is not just riding the majority class. The macro-F1 is somewhat lower at 80 %, which is expected since it gives equal weight to every class, including the hard minority ones.

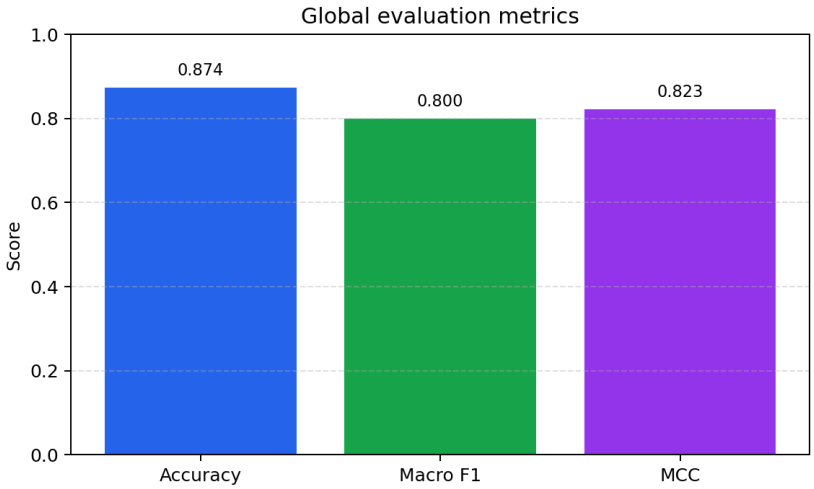

Figure 4: Global metrics bar chart. All three metrics sit above 0.80, with accuracy slightly ahead due to the majority-class effect.

4.2 Per-class analysis

Class	Precision	Recall	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: Per-class precision, recall, and F1-score on the test set.

Looking at the per-class figures, the standout is Periplasmic with an F1 of 0.93—this suggests that periplasmic proteins have quite distinct signatures that the dual-embedding approach captures well. Cell Wall is also a pleasant surprise at 0.84, considering there are only 87 training examples. On the other hand, Cytoplasmic Membrane lags behind at 0.60. We suspect this is because transmembrane proteins share features with both cytoplasmic and membrane classes, making them harder to disentangle.

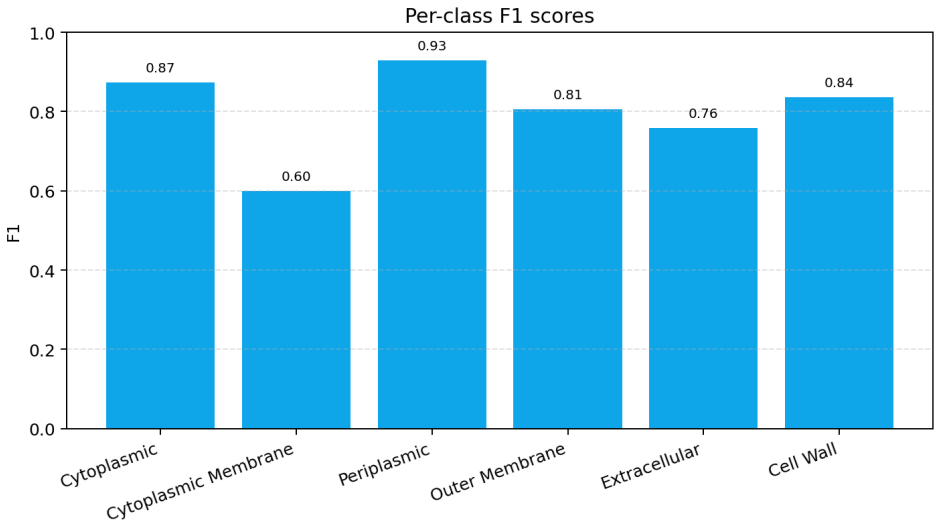

Figure 5: F1-score per localization class. Periplasmic leads while Cytoplasmic Membrane clearly stands out as the most challenging class.

4.3 Confusion matrix

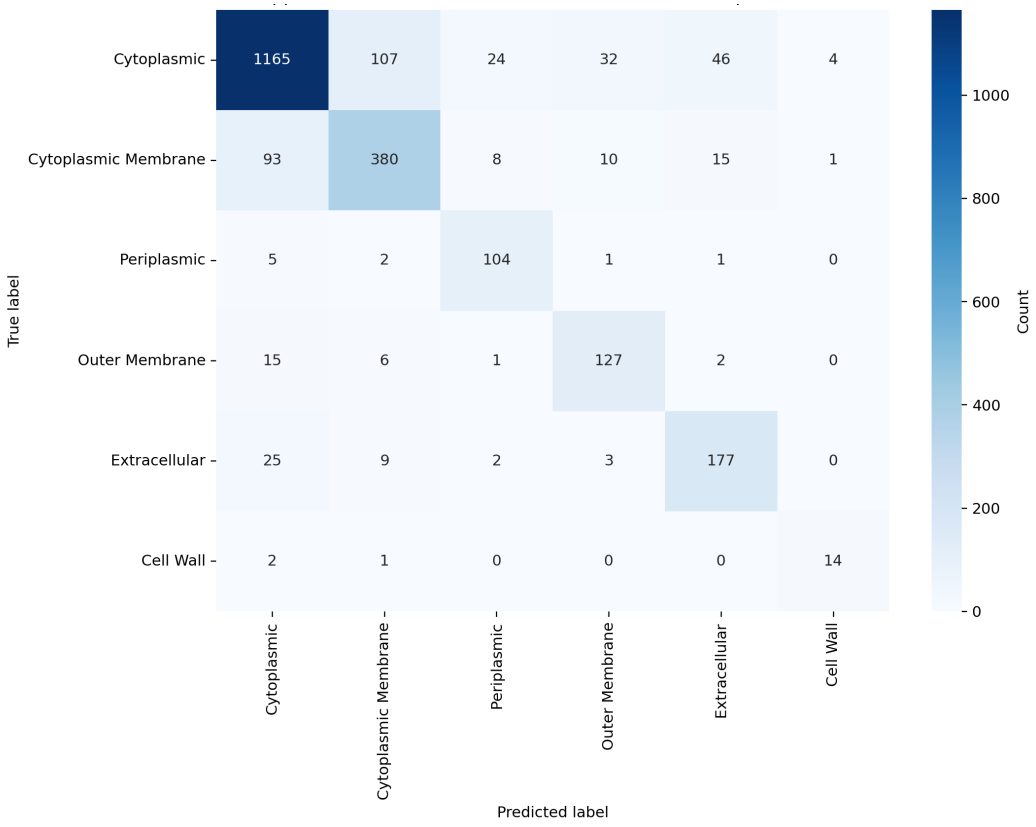

Figure 6: Confusion matrix

5 Discussion

5.1 What the numbers tell us

At first glance, 87 % accuracy might seem good enough on its own, but it is worth looking more carefully. Since Cytoplasmic proteins make up nearly 58 % of the test set, a model that always predicts "Cytoplasmic" would already score around 58 % accuracy without learning anything useful. The fact that our macro-F1 reaches 80 % and the MCC sits at 0.82 tells a more complete story: the model is genuinely discriminating across all six classes, not just latching onto the majority.

The gap between accuracy (87 %) and macro-F1 (80 %) is itself informative—it comes from the weaker classes dragging the macro average down, which is exactly what we would expect given the imbalance.

5.2 Per-class insights

A few classes deserve a closer look:

- **Periplasmic** (F1 = 0.929): the best result by far. Periplasmic proteins in Gram-negative bacteria tend to have signal peptides and specific structural motifs that apparently leave a clear footprint in both the evolutionary and structural embeddings.
- **Cell Wall** (F1 = 0.837): this is a reassuring result. It shows that a well-chosen training strategy can compensate for very small class sizes.
- **Cytoplasmic Membrane** (F1 = 0.600): the clear weak spot. This is the second-largest class (2 535 samples), so the issue is not data scarcity. The likely explanation is biological: transmembrane proteins straddle the membrane, meaning they have substantial cytoplasmic domains. At the embedding level, this blurs the line between "cytoplasmic" and "membrane," making the two hard to separate.
- **Outer Membrane** (F1 = 0.806) and **Extracellular** (F1 = 0.758): both are decent but not great. These two compartments are linked through secretion pathways, so it makes sense that some proteins would be confused between them.

5.3 Limitations

We want to be upfront about what this work does *not* cover:

- Capping sequences at 1 000 amino acids means about 3 % of proteins are truncated. For the longest ones (up to 5 627 aa), we are potentially losing important information.
- Running on CPU was a necessity, not a choice. It restricted both the model size and the experiments we could try.
- Everything relies on a single train/test split. Doing proper cross-validation—ideally with homology-aware partitioning—would give more reliable numbers.

6 Reproducibility

To reproduce the full pipeline from scratch, run the following commands in order:

```
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 and future work

In this project we built an end-to-end pipeline that takes raw protein sequences and predicts their subcellular localization across six classes. The core idea is to combine two complementary views of each protein—evolutionary patterns from ESM-C and structural features from ProstT5 3Di—through a Transformer–BiLSTM fusion architecture. Even though everything ran on CPU, the model achieves 87 % accuracy and 80 % macro-F1, with the combination of focal loss and balanced sampling proving effective for the severely underrepresented classes.

That said, there is room for improvement. With GPU access, we could drop the sequence-length cap and experiment with larger or more recent embedding models. Running ablation studies would help us understand exactly what each component of the architecture brings to the table. On the evaluation side, proper cross-validation with homology-based clustering (via MMseqs2 at 30 % identity) would give tighter, more trustworthy performance estimates. And of course, exploring deeper Transformers or different pooling strategies could push the numbers further—especially for the troublesome Cytoplasmic Membrane class.