Performing Dynamic Protein Localization with TCNN and LSTM Models

Bocchieri Leonardo, Farina Emilia, Rota Riccardo

***Abstract*—In this report we go through the methods used in our project about Dynamic Protein Localization, in which we try to predict the time-dependent position of budding yeast proteins during the eukaryotic cell-cycle. We then discuss the results obtained and give some interpretation to them, highlighting which features are most relevant for the prediction.**

I. Introduction

Protein localization is a well known problem in biological machine learning. Different studies have reached a good accuracy in predicting the static position of proteins inside the cell basing on their amino acid sequences. During cell cycle, proteins appear to change position in the different phases of the process, so in this case the problem can be seen as time-dependent. In this project we try to extend the task of protein localization to a dynamic framework. We use as ground truth the data on location and concentration of budding yeast proteins during the eukaryotic cell cycle collected by [1]. The goal is to develop models able to predict the position of proteins during 5 phases of the cell division given the following features:

- sequence of the target protein
- concentration of the target protein in the different phases
- positions and concentrations of the proteins biologically interacting with the target protein.

This problem presents different challenges. The main obstacles are the small size of the dataset (due to the limited number of budding yeast proteins), the varying lengths of the sequences, and the time-dependent nature of the labels to predict. For this reason very simple architectures that do not use pre-trained models are not enough to solve it.

II. Data description and preprocessing

A. Extraction of static and dynamic localizations

We extract the locations of the proteins in the cells for both the static and the dynamic models, from a file named `S1_protein_location.xlsx` [1], which contains three replicates of the same measurements conducted in a biological laboratory. A single replicate is represented by a table that associates each protein identifier (yORF) with a DeepLoc activation function for every subcellular compartment of the yeast cell, as well as for each stage of the cell's life cycle (namely G1 Pre-Start, G1 Post-Start, S/G2, Metaphase, Anaphase and Telophase). The values generated by these functions represent the probabilities that a protein is located in a specific compartment. We began by creating a single table containing 3902 samples (i.e. proteins) and 91 columns (i.e. yORFs plus number of compartments $\times$ number of phases). This was achieved by averaging the three measurement replicates and reducing the cellular compartments to 15 groups based on closely related locations. One reason for grouping subcellular compartments is to simplify the analysis by reducing the dataset complexity without losing relevant information. This approach improves the interpretability of the results while minimizing redundancy in the data. More in particular, the 15 classes obtained are: 'Bud', 'ER', 'Eisosomes', 'Nucleus', 'Punctate Nuclear', 'Vacuole', 'cell periphery', 'cytoplasm', 'cytoskeleton', 'endosome', 'golgi', 'lipid particle', 'mitochondrion, peroxisome' and 'None'. The last entry indicates that the protein cannot be localized to any of the other compartments. To create labels for the static problem starting from this dataset, we compute the mean of each compartment across the time steps and for each sample we keep as subcellular localization the one with the maximum value.

B. Sequences processing

For the protein sequences we make reference to the file named `orf_trans_all.fasta` [2], which associates the proteins with their corresponding amino acid sequences. In order to generate the embeddings for all sequences utilizing the pre-trained ESM models [3], we merged this table with the yORFs of the data described earlier, resulting in a dataset that includes only the yORFs and the corresponding sequences. During this process, we excluded 12 proteins for which sequence data was unavailable. Moreover, due to its attention mechanism, the ESM models support sequences with a maximum length of 1024 amino acids. Since we noticed that these "long sequences" affected only 347 proteins and their distribution is balanced among classes, as shown in Figure 1, we can exclude proteins with sequences exceeding this threshold without any consequence. The result consists of a dataset with 3544 samples and 16 columns.

C. Extraction of dynamic features

To create the dataset for the dynamic models, we proceeded similarly to the static case, extracting the subcellular compartment corresponding to the maximum DeepLoc value for each phase, while removing all irrelevant columns. In this way we constructed a dataset containing the yORFs and the localizations for all stages. Using this dataset, we analyzed the frequency of dynamic and static proteins, finding out that approximately 400 proteins change their position at least once during the cell's life cycle, as shown in 2. The remaining

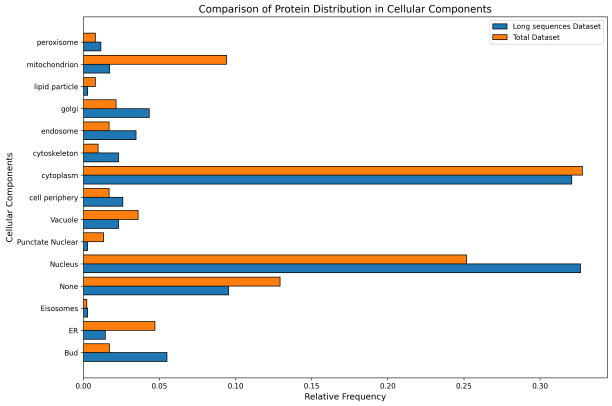

Fig. 1. Distribution of long sequences

proteins are stationary and the most are localized to the nucleus or cytoplasm.

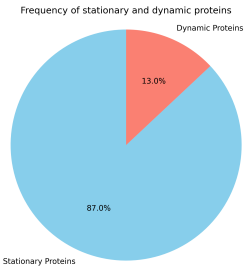

Fig. 2. Dynamic proteins

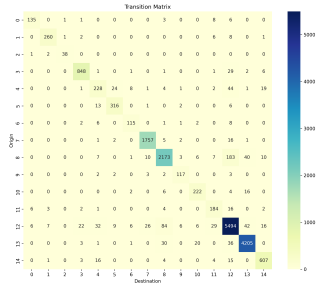

Fig. 3. Transition matrix.

To obtain the final dynamic dataset, we also processed two additional files. The first, named `S3_protein_level.xlsx` [1], provides the protein concentrations in the last five phases, including TE (Translational Efficiency) and TL (Transcript Level) values. As with protein localization, this file also includes two replicates of the data for both TE and TL concentrations, from which we generated a single table by averaging the values. The second, `The_Yeast_Interactome_Edges.xlsx` [4], consists of a table that shows the level of interaction between each pair of proteins. From this document, we constructed a symmetric matrix to represent protein interactions, assigning a value of 0 to pairs not included in the file. Then this is processed as follows: at each phase, all the interaction scores are multiplied for the protein concentrations. Then, for each protein, we select the 10 proteins with the highest score. We assign a weight of 0.1 to each of their localizations, so that we obtain a distribution probability over the 15 classes that will be passed to the model as a feature representing the interaction information. This process is made in 3 ways: considering both the TE and TL concentration levels and without considering the concentration. The final dynamic dataset, composed of 3544 rows and 734 columns, was then created by merging different information: the yORFs, the embeddings for the full sequences, the raw sequences for the first and last 20 amino acids, the static localization, the dynamic localizations, the protein concentrations, the variations of protein concentrations and the interaction matrix.

III. Static Model Development

Before addressing the time-dependent protein problem we tried to develop a model able to predict static positions for the proteins starting from their sequences. This task was successfully performed on other types of proteins by SOTA models based on embeddings [5]. Some studies also suggest that important information about localization could be found in the two ends of the sequence, i.e. in the first and last 20 amino acids [1]. Due to the small dimension of the dataset, it was really important to get information from some pre-trained models. For this reasons we developed architectures taking as inputs embeddings obtained from the **esm2_t30_150M_UR50D** model [3] that was trained on a large set of proteins of other species. We tested the accuracy of three different architectures on this task. All models were trained with the Cross Entropy Loss function, which applies a softmax activation to the output logits.

A. Static Model 1: MLP

The first architecture tried is a simple MultiLayer Perceptron taking as inputs only the embeddings of the sequences. It consists of two fully connected hidden layers: the first layer reduces the size to 256 units, and the second reduces it further to 64 units. The output layer produces the logits for each of the 15 classes. After each of them we use a normalization layer and the ReLU activation function, together with dropout. This model reaches an accuracy of 56.56%, proving that the embeddings provide significative information about the sequences features. However,it is strongly overfitting despite all the adopted regularization techniques.

B. Static Model 2: XGBoost

For the second model, we tried to use the implementation provided by the library XGBoost [6] in Python, which performs the task of classification with an architecture based on binary decision trees. We passed in input the ESM embeddings of both the global sequences and the extremities sequences (so that the input dimension is three times the dimension of the embedding) and set as parameters the following: learning rate = 0.02, max depth = 8, lambda = 9, alpha = 2, gamma = 0.5, objective = multi:softmax, eval metric = 'mlogloss'. As showed in Table I, we found out that this model does not provide good accuracy on our problem, so we do not develop it further.

C. Static Model 3: Multibranch model

For the third model we tried a more original architecture. To contrast overfitting, we used only a dense layer on embeddings. We also added the information about the sequence extremities, passed as numeric vectors obtained by

mapping letters to numbers with a vocabulary, and applied small learnable embeddings to them. The designed model 4 consists of three branches:

- A single dense layer with a ReLU activation function and a dropout rate of 0.3 that takes as input the protein embedding and outputs 32 units.
- Two identic branches composed by learnable 16-dimensional embeddings of each one of the 20 amino acids, followed by transformers and a last linear dense layer (with 0.3 dropout rate and ReLU activation) that outputs 32 units. One of them processes the beginning part of the sequence, the other one processes the end.

The three 32-dimensional outputs were given as input to a final classificator, a single layer Perceptron that outputs the logits of the 15 classes. As showed in Table I, this model reaches the best performance among the ones tried, so we selected it to be part of the dynamic models. We also performed cross-validation on different parameters set in order to find the ones to be used when extending the problem to the dynamic case. Results are reported in B.

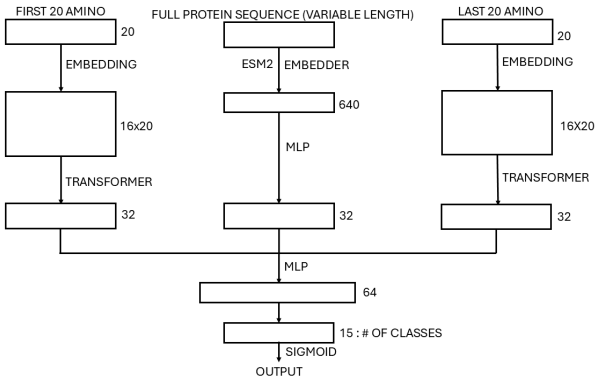

Fig. 4. Multibranch architecture

IV. Dynamic Model development

After developing the static model, we can pass to the dynamic problem. We create two different models, both extending the existing static Model 3 developed in I, with networks trained on dynamic data. The accuracy obtained by each model is reported in Table II. The dynamic data at each of the 5 timesteps are:

- the distribution on the 15 classes obtained from the proteins interacting with the target protein as described in II (15-dimensional)
- the TE and TL concentration of the target protein (2-dimensional)
- the variation of the TE and TL concentrations at each timestep (2-dimensional).

The dynamic data for each batch will then be a 3-dimensional tensor, with dimensions: $[n_{batch}, 5, n_{input}]$, where n_{input} is 19.

A. Dynamic Model 1: TCNN

The first model treats the dynamic data in a block composed by two layers of 1D Temporal Convolutional Neural Network. The 19 dynamic data are treated as 19 different channels, so for each channel we have 5 timesteps. As shown in 5 the first layer has a kernel of 3 units with a dilation of 1 and a padding of 2 on the left and outputs 15 channels. The second kernel also has a kernel of 3 units with a dilation of 2, so we need a padding of 4 on the left. The final output is also composed by 15 units per timestep. This double layer structure guarantees that each timestep of the input takes information from all the previous timesteps, without anticipating information from the following ones. After each layer a normalization and a dropout with 0.3 rate are performed.

The static data are passed as input to a static block, which is the static Model 3 developed in I, pre-trained with the static locations of the proteins in the training set. The output is then identically replicated on the 5 phases, so that its shape matches the one of the dynamic block. We tried both to freeze the weights of this part and to make them learnable by training them with a small learning rate.

Finally the outputs of the two blocks (both 15-dimensional for each timestep) are combined by a dense layer, which computes the logits of the 15 classes.

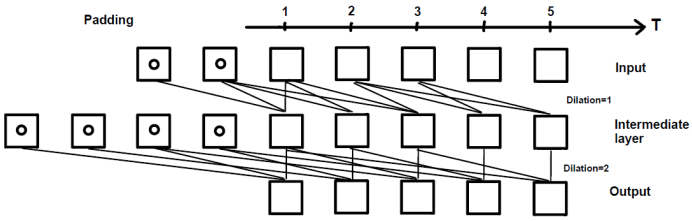

Fig. 5. TCNN architecture

B. Dynamic Model 2: LSTM

The second model integrated dynamic data with the output of the static Model 3 developed in I, into a combined dataset, which was then used as input for a Long Short-Term Memory architecture. An LSTM model is a type of recurrent neural network (RNN) architecture specifically designed to learn and remember long-term dependencies in sequential data. LSTMs address the vanishing gradient problem commonly encountered in traditional RNNs, enabling them to capture patterns across longer sequences. The architecture achieves this through a structure of interconnected memory cells, each with three key gates:

- forget gate: decides which information from the past should be discarded.
- input gate: determines which new information should be stored in the cell state.
- output gate: controls the output of the current state, determining the information to pass to the next timestep.

In our model we at first applied a Softmax function to the static logits obtained as outputs of the Multibranch model. Then, to ensure consistency with the dynamic data, we expanded the dimension of these features from $[n_{batch}, n_{classes}]$ with $n_{classes} = 15$ to $[n_{batch}, 5, n_{classes}]$, where 5 represents the number of timesteps considered. We then concatenated the static and dynamic tables, obtaining a final combined dataset of dimensions $[n_{batch}, 5, n_{static} + n_{dynamic}]$, with $n_{static} + n_{dynamic} = 34$. We passed this last dataset as input to an LSTM layer setting $hiddensize = 64$, $numlayer = 2$ and $dropout = 0.2$. We decided also to set the parameter $bidirectional = True$, which extends the capability of a traditional LSTM by processing data in both forward and backward directions. This allows the model to capture information from past and future contexts simultaneously. Successively, we used a Multi-Head Attention layer in which all the parameters, i.e. query, key and value, coincided with the output of the LSTM layer. This was done to implement a self-attention mechanism, in order to identify global relationships within the sequence. Finally, the predictions are obtained applying a linear layer to the outputs.

C. Dynamic Model 3: MLP

The last model we tried involves one dense layer, which takes as input 5 replicates of the sequence embedding (one for each timestep) and outputs the dynamic predictions on the 15 classes. As shown in V, this simple architecture performs worse than the other two, because it is not designed to handle temporal dependencies. However, the results obtained are still good and this can be given to the large amount of information contained in the embeddings.

V. Results

In Table I and Table II we report the test losses and the test accuracies reached by our static and dynamic models:

Model	Loss	Accuracy
MLP	1.258	58.11%
XGBoost	2.123	25.00%
MultiBranch	1.296	57.55%

TABLE I
ACCURACY OF THE STATIC MODELS

Model	Loss	Accuracy	Accuracy only dynamic
TCNN	1.288	57.94%	37.96%
LSTM	1.342	58.50	36.53%
MLP	1.375	53.82%	31.22%

TABLE II
ACCURACY OF THE DYNAMIC MODELS

For an interpretability purpose, we tried to run the LSTM model giving to it only a part of the inputs. The results of this experiment are reported in the Table III, and show that the sequences contain the most relevant information, adding the interaction helps in increasing the accuracy, while the concentration seems not to be a relevant parameter for our task.

Concentration	Interaction	Embeddings	Test Loss	Test Accuracy
YES	YES	YES	1.2980	57%
YES	YES	NO	1.6937	44%
YES	NO	YES	1.3773	54%
NO	YES	YES	1.2990	58%
YES	NO	NO	1.8650	36%
NO	YES	NO	1.6382	49%
NO	NO	YES	1.3658	54%

TABLE III
ACCURACY VARYING THE INPUT FEATURES

The graphs in Figure 6 and 8 show the trends of losses and accuracies on the train and test sets during the training of the two best dynamic models:

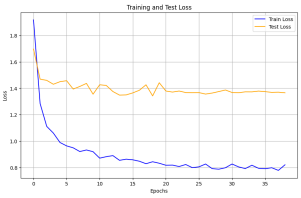

Fig. 6. Loss LSTM

Fig. 7. Accuracy LSTM

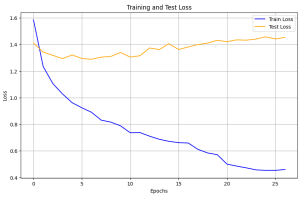

Fig. 8. Loss TCNN

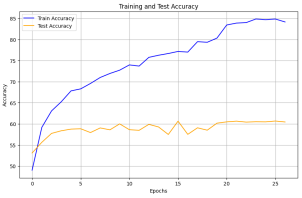

Fig. 9. Accuracy TCNN

VI. Conclusion

Basing on the results we obtained we can draw some conclusions. The static model we develop (Static Model 3) achieves 57% accuracy on the bud yeast dataset. Given that the SOTA models, when working on datasets considered "hard" to predict (such as [5]) reach accuracies around 60%, we can conclude that our static model is a good base to build dynamic models on. We see that the ending parts of the sequences contain relevant information. Anyway, having a very small dataset to train our architectures, the contribution of pretraining on the ESM models is very decisive. When passing to the dynamic prediction, we note that both models reach similar level of accuracy. However, if considering only the dynamic proteins, accuracy levels drop. This means that our model, while being good at detecting the positions, is not detecting their variations well. We explain this fact with the small size of the dataset: as explained in II, only around 400 proteins have a dynamic behavior, and this is not enough for NN models to find recurrent patterns, given that past studies have shown that it is not possible to perform data augmentation by perturbating the sequences. Future studies on this problem could improve our results by increasing the size of the dataset by considering other species of yeast or fine-tuning ESM.

REFERENCES

[1] A. Litsios, B. T. Grys, O. Z. Kraus, H. Friesen, C. Ross *et al.*, "Proteome-scale movements and compartment connectivity during the eukaryotic cell cycle," *Cell*, vol. 187, no. 6, pp. 1490–1507, 2024. [Online]. Available: https://doi.org/10.1016/j.cell.2024.02.014

[2] Saccharomyces Genome Database (SGD), "Yeastgfp: Localization data for yeast proteins," 2024.

[3] A. Rives, J. Meier, T. Sercu, S. Rao, S. Lu, D. F. Heinzinger, S. Kannan, T. Bereznyak, Z. Ying, D. R. Schwab *et al.*, "Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences," 2019. [Online]. Available: https://github.com/facebookresearch/esm

[4] A.-D. B. André C. Michaelis, "The social and structural architecture of the yeast protein interactome)," *Nature*, vol. 620, p. 123–134.

[5] C. D. Hannes Stärk, "Light attention predicts protein location from the language of life," *Bioinformatics Advances*, vol. 1, no. 1, p. vbab035, 2021. [Online]. Available: https://academic.oup.com/bioinformaticsadvances/article/1/1/vbab035/6432029

[6] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," 2016, accessed: 2024-12-19. [Online]. Available: https://github.com/dmlc/xgboost

APPENDIX

A. Ethical Risk

The ethical risk associated with our project is that the proposed classifier could be used to design or engineer pathogenic proteins with precise intracellular localization, potentially enhancing their virulence or altering their function in harmful ways. The primary stakeholders impacted by this are the global population and the scientific community. The former are the potential victims of bio engineered pathogens, while for the latter the risk could be a decline in trust in biotechnological advancements due to the misuse of scientific tools. Even if the likelihood of this event can be considered very moderate, the severity of the problem is very high. If realized, this could lead to catastrophic health consequences or bioterrorism.

To evaluate the magnitude of this risk we have considered two principal metrics: the accessibility of the classifier and the complexity of the tool. Firstly, the accessibility of our classifier is very limited given the specific scope and purpose of our project. Secondly, the expertise required to exploit the tool for harmful purposes is highly specialized and advanced. Highlighting this risk in our report could encourage the development of safeguards against dual-use challenges. However, once the knowledge and the classifier are shared, fully preventing its measure could be very difficult, if not impossible. Indeed, balancing transparency in science with security concerns is very challenging and restricting access could also limit the beneficial applications of this research.

B. Cross-validation results

The results of the cross validation grid search performed to find the best hyperparameters for the static model are reported in IV.

Dropout	Learning Rate	Weight Decay	Scheduler	Loss	Accuracy
0	0.001	0	CosAnnLR	1.586	52.23
0	0.001	0	StepLR	1.601	53.36
0	0.001	0	ExpLR	1.450	53.18
0	0.001	0.0001	CosAnnLR	1.580	52.99
0	0.001	0.0001	StepLR	1.563	53.75
0	0.001	0.0001	ExpLR	1.438	54.37
0	0.001	0.01	CosAnnLR	1.595	54.63
0	0.001	0.01	StepLR	1.623	52.26
0	0.001	0.01	ExpLR	1.437	54.03
0	0.01	0	CosAnnLR	1.905	51.60
0	0.01	0	StepLR	1.789	53.50
0	0.01	0	ExpLR	1.851	53.55
0	0.01	0.0001	CosAnnLR	1.870	52.37
0	0.01	0.0001	StepLR	1.918	52.71
0	0.01	0.0001	ExpLR	1.847	52.29
0	0.01	0.01	CosAnnLR	1.729	53.41
0	0.01	0.01	StepLR	1.840	52.85
0	0.01	0.01	ExpLR	1.902	53.24
0.25	0.001	0	CosAnnLR	1.393	55.30
0.25	0.001	0	ExpLR	1.406	53.87
0.25	0.001	0.0001	CosAnnLR	1.394	54.94
0.25	0.01	0.0001	StepLR	1.374	55.36
0.25	0.01	0.0001	ExpLR	1.377	54.09
0.25	0.001	0.01	CosAnnLR	1.385	55.30

TABLE IV
CROSS VALIDATION ON DIFFERENT PARAMETERS FOR STATIC MODEL

The results of the cross validation grid search performed to find the best hyperparameters for the static model are reported in V.

Learning Rate	Lambda Penalty	Static Learnable	Hidden Size	Loss	Accuracy
0.001	0	True	32	1.429	54.96
0.001	0	True	64	1.437	53.31
0.001	0	False	32	1.432	54.61
0.001	0	False	64	1.422	55.06
0.001	1e-05	True	32	1.430	53.44
0.001	1e-05	True	64	1.418	54.85
0.001	1e-05	False	32	1.423	54.63
0.001	1e-05	False	64	1.416	55.51
0.001	0.01	True	32	1.390	55.50
0.001	0.01	True	64	1.443	54.40
0.001	0.01	False	32	1.417	54.19
0.001	0.01	False	64	1.419	53.43
0.01	0	True	32	1.504	52.92
0.01	0	True	64	1.528	54.19
0.01	0	False	32	1.476	54.36
0.01	0	False	64	1.487	53.50
0.01	1e-05	True	32	1.476	53.42
0.01	1e-05	True	64	1.494	52.69
0.01	1e-05	False	32	1.473	52.26
0.01	1e-05	False	64	1.512	53.28
0.01	0.01	True	32	1.484	54.28
0.01	0.01	True	64	1.459	54.10
0.01	0.01	False	32	1.455	54.44
0.01	0.01	False	64	1.456	53.04

TABLE V
CROSS VALIDATION ON DIFFERENT PARAMETERS FOR TCNN DYNAMIC MODEL

C. Baseline model for comparison

We developed a simple baseline model to evaluate whether our dynamic models outperform a naive approach. In this baseline, each sequence is padded or truncated to 512 amino acids, ensuring uniform input length. The sequences are then mapped to numerical values using the same vocabulary described in I. The processed sequences are combined with the protein dynamic data obtained in II. Subsequently, we apply a Multi-Layer Perceptron (MLP) with a hidden layer of size 64, training the model for 50 epochs. This approach achieves an accuracy of 33.57% on the test set and exhibits significant overfitting, highlighting its limitations.

Fig. 10. Loss

Fig. 11. Accuracy

D. Supplementary Figures

To complement the graphs illustrating the frequency of dynamic proteins in our datasets, we also created pie charts to depict the frequency distribution across each phase transition 12. Additionally, we computed the transition matrices for each phase (Figure 13).

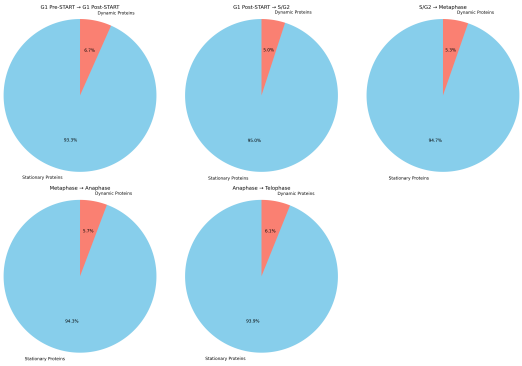

Fig. 12. Dynamic proteins per phase.

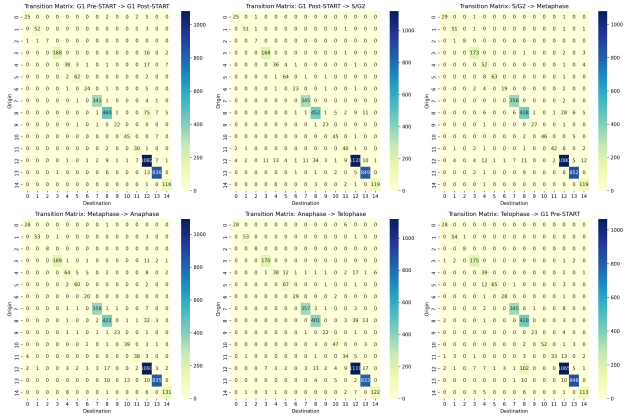

Fig. 13. Transition matrices for dynamic proteins per phase.