TH Bingen Technical University of Applied Sciences
Department 2 - Technology, Informatics and Economics
Applied Bioinformatics (B.Sc.)

Prediction of Liquid-Liquid Phase Separation of Proteins using Neural Networks

Bachelor Thesis
Submitted on: 24.07.2025
Name: Robin Ender

Advisor: Prof. Dr. Asis Hallab
2nd Advisor: Eric Schumbera

Acknowledgements

I would like to express my heartfelt gratitude to all those who supported me throughout the course of this thesis.

My sincere thanks go to Dr. Asis Hallab for generously sharing his expertise and providing valuable support during my research.

I am also deeply grateful to Eric Schumbera for his guidance and for granting me the freedom to explore and make independent decisions throughout this project.

I would also like to thank my partner, Pia Bergknecht, for always having my back — not just during this work, but in all aspects of life.

Mainz, 24.07.2025 Robin Ender

Abstract

Liquid-Liquid Phase Separation of proteins is a fundamental cellular mechanism responsible for compartmentalization without membrane boundaries. It also plays a role in the development of certain diseases. Computational prediction of Liquid-Liquid Phase Separation propensity helps in understanding protein interactions in cellular contexts where experimental data is lacking.

This work presents a novel sequence-based predictor for Liquid-Liquid Phase Separation proteins using Neural Networks. In addition to raw amino acid sequences, a block decomposition of these sequences was explored as main input. Various Neural Network architectures were evaluated, including Convolutional Neural Networks, Bidirectional Long-Short Term Memory models, and Transformers. From these models the best performing was chosen and further enhanced with additional features such as Relative Solvent Accessibility and Post Translational Modifications. Unlike existing Liquid-Liquid Phase Separation predictors that rely on conventional Machine Learning models and therefore use tabular inputs, this approach enables the direct use of sequence data.

The results demonstrate that even relatively simple Neural Network architectures can match the performance of current state-of-the-art predictors. The usage of the raw sequences proved to be better than using the block decomposition. While Liquid-Liquid Phase Separation prediction remains challenging due to limited experimental data and complex biological mechanisms, this study suggests that sequence-based Neural Network models hold strong potential. As more data becomes available, such models may define the next generation of Liquid-Liquid Phase Separation predictors.

Table of Contents

List of Figures

List of Tables

Glossary

AF. Activation Function 6, 7

AI. Artificial Intelligence 4, 5, 9, 10

AMPL. Adaptive Max Pooling Layer 17, 18, 19, 21

AUC. Area Under the Curve 16, 17, 21, 24, 25, 26, 27, 28

BN. Batch Normalization 17, 20, 21, 25, 31, 32, 34

BiLSTM. Bidirectional Long Short-Term Memory 8, 17, 19, 24, 31

CL. Convolutional Layer 17, 18, 19, 21

CNN. Convolutional Neural Network 7, 8, 14, 16, 17, 18, 19, 20, 24, 31, 32, 34

DL. Deep Learning 4, 5

DO. Dropout 17, 18, 19, 20, 21

ED. Embedding Dimension 17, 18, 19, 20, 21

FCL. Fully Connected Layer 5, 17, 18, 19, 21

FN. False Negative 15

FP. False Positive 15

HMM. Hidden Markov Model 2, 3, 4

IDP. Intrinsically Disordered Protein 2, 13, 33

IDR. Intrinsically Disordered Region 3, 4, 13, 14, 17, 20, 21, 22, 24, 25, 26, 27, 28, 29, 31, 32, 33, 34

KS. Kernel Size 17, 18, 19, 21

LLPS. Liquid-Liquid Phase Separation 1, 2, 3, 4, 11, 12, 13, 16, 17, 20, 21, 22, 28, 29, 30, 31, 32, 33, 34

LSTM. Long Short-Term Memory 8, 19

ML. Machine Learning 1, 2, 4, 5, 10, 16, 31, 32, 34

MLO. Membraneless Organell 1, 12, 13, 21, 22, 26, 27, 32

MPL. Max Pooling Layer 17, 18, 19, 21

MSA. Multiple Sequence Alignment 5

NN. Neural Network 1, 5, 6, 7, 15, 16, 17, 19, 31, 34

OC. Output Channels 17, 18, 19, 21

PD. Padding 17, 18, 19, 21

PRC. Precision Recall Curve 16, 17, 24, 25, 26, 27, 28

PTM. Post-translational modification 2, 12, 14, 17, 20, 21, 23, 25, 32, 34

RNN. Recurrent Neural Network 8

ROC. Receiver Operator Curve 16, 17, 24, 25, 26, 27, 28

RSA. Relative Solvent Accesibility 4, 14, 20, 21, 23, 25, 31, 32, 34

ReLU. Rectified Linear Unit 7, 17, 18, 19, 21

SSUP. Structured Superficial Regions 4

ST. Stride 17, 18, 19, 21

TN. True Negative 15

TP. True Positive 15

1 Introduction

Liquid-Liquid Phase Separation (LLPS) is a biologically important process that plays key roles in cells. It is a major contributor to the functional compartmentalization but is also associated with some diseases [1]. Although it is possible to experimentally determine whether a protein undergoes LLPS under specific conditions, such experiments are expensive and consume resources. As a result, computational prediction of LLPS poses an attractive alternative.

In recent years, a variety of computational tools for LLPS prediction have been developed. These tools have improved over time, incorporating new features and knowledge as well Machine Learning (ML) techniques [2]. However, their predictive power still leaves room for improvement.

Using Neural Networks (NNs) could simplify the development of LLPS predictors, as they require less feature engineering and offer potentially greater predictive power especially as more curated data becomes available [3]. Their ability to use the whole sequence as input should enable them to capture patterns and relationships between amino acids that relate to LLPS. This is something more traditional ML models can not do. This work will therefore test if NNs are suitable for LLPS prediction and how they compare to current state-of-the-art predictors. For the main input two different approaches will be explored. One will use the raw amino acid sequence, while the other will use multiple block decompositions of the amino acid sequence that each emphasize different biochemical properties.

1.1 Liquid-Liquid Phase Separation

LLPS is a reversible process where a homogeneous mixture spontaneously separates into two liquid phases, one with a depleted and one with an increased concentration of components, see Figure 1 [1]. LLPS can form colloids like milk or layers like a mixture of oil and water. It is a widespread phenomena that is dependent on many factors like temperature, pressure, differences in polarity and hydrophobicity. Studying LLPS in polymer systems started in the mid of the 20th century. In the late 20th century the phenomena was recognized as an important process in organisms and studies began to understand the implications of LLPS in an biological context. [1]

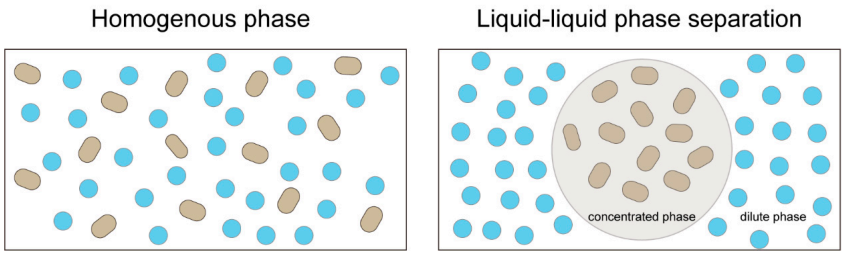


Figure 1: Visualization of LLPS.

Cells are capable of conducting various different tasks that involve many biochemical reactions. As these reactions often need different educts, enzymes and conditions a spatial organization helps the cell to carry out these reactions effectively. Organelles are the compartments of the cell. They need a boundary that separates them from the rest of the cell and the components inside the compartment have to be able to move freely. Compartments that are confined by a membrane, like the nucleus or mitochondria, have been well known for a long time. A more recent discovery was the existence of Membraneless Organells (MLOs). They often form via LLPS. A well known example of one such MLO is the nucleoli. [1]

The macro molecules responsible for LLPS in cells are proteins and nucleic acids. A combination of simultaneous weak and strong interactions between proteins and proteins or proteins and nucleic acids seems to be the driving force. The presence of many different binding sites, multi valency, is a crucial parameter. The solubility also effects the propensity to undergo LLPS, as many proteins that

undergo LLPS have a poor solubility in water. Other than proteins with multiple domains, Intrinsically Disordered Proteins (IDPs) are often involved in LLPS. They differ from globular molecules in two ways. They do not posses a fixed conformation and they only use a small subset of the available amino acids. Their function is generally less dependent on their exact sequence than on more general characteristics like charge patterns. [4]

As proteins that take part in LLPS can serve different roles, they were divided into two groups. One group that is able to undergo LLPS on its own and one group that can only join existing condensates. The proteins of the first group are called drivers, scaffolds or self-assembling proteins, while the proteins in the second group are called clients or partner-dependent proteins. [5], [6]

As LLPS is extremely sensitive to changes in physico-chemical conditions, it is possible that it also plays an important role in stress adaptation [4]. However, LLPS is not always wanted. In some cases proteins undergo LLPS that normally would not. This can be due to several reasons like mutations or Post-translational modifications (PTMs). These unwanted aggregates are suspected to lead to diseases like cancer or neurodegenerative diseases. [1]

1.2 Phase Separation Predictors

1.2.1 Overview

With the rise of interest on the topic of LLPS, tools were developed to predict LLPS propensity for proteins. The first generation of tools were not specifically developed for LLPS prediction. Their goals centered around finding prion-like domains (PLAAC [7]), finding proteins that are associated with RNA granules (catGranule [8]) or detecting proteins with high propensities for $\pi - \pi$ interactions (PScore [9]). While these tools were not intended for LLPS prediction, their predictions all correlated with the prediction of LLPS. These tools used Hidden Markov Models (HMMs) or scoring formulas. PSPer was one of the first tools specifically designed for LLPS prediction, it still used HMMs [10]. The tools that followed started to use ML models as they were better suited for the complex task of LLPS prediction [11]. With better understanding of the driving forces of LLPS and the incorporation of according features the predictors steadily improved. A short summary of some of the current LLPS predictors including a short description is given in Table 1. Two predictors, PdPS / SaPs (Section 1.2.2) and PSPire (Section 1.2.3) will be covered in more detail due to their relevance and performance.

Table 1: Summary of recent Liquid-liquid phase separation predictors.

Name	Information used for prediction	Release Article
PLAAC [7]	HMM to find prion like domains	2014
catGRANULE [8]	Scoring Formula that uses RNA bindings, structural disorder propensity, amino acid patterns and polypeptide length	2016
PScore [9]	Scoring Formula that uses π-π interaction potential	2018
PSPer [10]	Modified HMM to find prion like domains that undergo LLPS	2019
FuzDrop [11]	Logistic Regression model using disordered binding modes and the degree of protein disorder	2020
MaGS [12]	Generalized Linear Model that uses protein abundance, percentage of protein intrinsic disorder, number of annotated phosphorylation cites, PScore, Camsol score, RNA interaction, and percentage of composition of leucine and glycine	2020
PSAP [13]	Random Forest Classifier that uses amino acid composition, sequence length, isoelectric point, molecular weight, GRAVY, aromaticity, secondary structure content, intrinsic disorder scores, hydrophobicity profiles	2021
PSPredictor [14]	Gradient Boosting Decision Tree that uses Word2Vec embeddings of sequence k-mers	2022
PdPS / SaPS [5]	see Section 1.2.2	2022
PSPire [2]	see Section 1.2.3	2024
PSPHunter [15]	Random Forest Model that uses amino acid composition, evolutionary conservation, predicted functional site annotations, word embedding vectors, protein annotation information and protein–protein interaction network features	2024
PICNIC [16]	Catboost classifier that uses disorder scores, sequence complexity, sequence distance-based features, secondary structure based features and Gene Ontology Features (for their GO-model)	2024
catCRANULE 2.0 [17]	Multilayer Perceptron that uses a combination of 82 physico-chemical and 46 structural and RNA-binding features	2025

1.2.2 PhaSePred - PdPS / SaPS

PhaSePred is a meta-predictor that integrates multiple existing tools to predict LLPS propensity. It combines outputs from several established LLPS predictors listed in Table 1 (including CatGranule, PLAAC, and PScore), an Intrinsically Disordered Region (IDR) predictor (ESpritz [18]), a low-complexity region prediction [19], hydropathy predictions from CIDER [20], the coiled-coil domain predictor DeepCoil [21], the Immunofluorescence-image-based droplet-forming propensity predictor DeepPhase [22], and phosphorylation data from PhosphoSitePlus [23].

PhaSePred has two specialized submodels: PdPS and SaPS, tailored for different LLPS mechanisms. PdPS is trained and evaluated on partner-dependent LLPS proteins, while SaPS is trained and tested on self-assembling proteins. Each of these has two versions: one for human proteins, which utilizes the full feature set including phosphorylation frequency and Immunofluorescence-image-based droplet propensity and one for non-human proteins, which omits those two features due to limited data availability. By distinguishing between partner-dependent and self-assembling proteins, the PdPS and SaPS models achieve significantly improved predictive performance, particularly for partner-dependent LLPS proteins. They chose XGBoost [24] as their ML model. [5].

1.2.3 PSPire

PSPire was developed due to the ongoing challenge of accurately predicting LLPS in proteins that do not contain IDRs. The key innovation in their approach was to separate protein features into two categories: IDR related and Structured Superficial Regions (SSUP) related features. Amino acids in SSUPs of the protein were only taken into account if their Relative Solvent Accesibility (RSA) value was above 25%, as amino acids that are on the surface of a protein are more likely to contribute to the formation of LLPS. [2]

The structural information was taken from AlphaFold and the calculation of the RSA values was conducted using PSAIA [25]. Overall 44 features were engineered from which 39 were calculated separately for IDRs and SSUPs. These include fractions of amino acids, fractions of groups of amino acids, the averaged scores of the: isoelectric point, molecular weight, hydropathy and polarity as well as IDR sequence length, fraction of IDRs, stickers and sticker pairs in SSUPs and number of phosphorylation sites. They also used XGBoost as their ML model and were able to significantly outperform other LLPS predictors when it comes to the prediction of non-IDR proteins. [2]

1.2.4 Problems of Liquid-Liquid Phase Separation Prediction

While the prediction of proteins undergoing LLPS has steadily improved, significant challenges remain. One of the primary difficulties lies in the complexity of LLPS itself. It is driven by multiple factors, including both strong and weak multivalent interactions [1], $\pi - \pi$ interactions [9], and hydrophobic effects [1]. As a result, capturing the full spectrum of mechanisms that contribute to LLPS is difficult using tabular data like the current predictors do.

Computational approaches, such as HMMs, have proven useful in many areas of bioinformatics, particularly for detecting conserved sequence motifs or structural domains [26]. However, their applicability to predict LLPS is limited. This is because LLPS is not solely determined by specific sequence motifs or local features [1].

Another major challenge is the limited availability of experimental data, particularly for negative samples [6]. This scarcity restricts the ability of ML models to generalize and hinders the development of balanced training datasets. Since ML models require a lot of data, especially deep learning approaches, the lack of it slows down progress.

Furthermore, many existing tools perform poorly in predicting partner-dependent and non-IDR proteins [2], [5]. As partner-dependent proteins usually have smaller IDR contents [27] these problems correlate. Predictors like PhasePred and PSPire have tried to address this issue, yet their performance on said LLPS proteins is still lacking.

1.3 Artificial Intelligence in Bioinformatics

Artificial Intelligence (AI) has brought new approaches to biological questions. Applications like ML, Deep Learning (DL), or natural language processing are already widely used and have helped to develop powerful bioinformatic tools. These tools are used in a variety of bioinformatic domains such as genome sequencing, protein structure prediction, drug discovery, systems biology, personalized

medicine, imaging, signal processing and text mining. Support Vector machines, random forests and NNs are just a few of the AI models employed in bioinformatics. [28]

As the techniques of AI models differ they are often categorized into ML, DL and NN as shown in Figure 2. Although NN and DL are technically subfields of ML, we distinguish between them in this work for clarity. Therefore, when we refer to ML in this text, it explicitly excludes NNs. DL models are a subcategory of NNs that consist of three or more layers. [29]

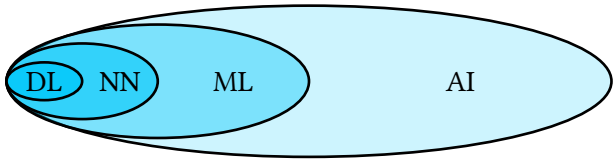


Figure 2: Categorization of AI models

Comparing ML models to NNs, there are some important differences. ML models usually perform better on small data sets (< 10000). NNs tend to overfit if the sample size is not large enough. ML models also need less computational resources and can therefore be trained faster and on lower end devices. They are also easier to interpret. NNs on the other hand are able to learn more complex interactions than ML models and are therefore able to outperform ML models in complex scenarios were there is enough training data. [30]
The input data for these models also differs. While ML models usually require tabular unstructured data, NNs are able to handle structured data.

One well known tool in bioinformatics that was created using AI was AlphaFold. It is a DL model developed by google that is able to predict the structure of proteins with almost experimental precision. It uses Multiple Sequence Alignments (MSAs) and is based on Bidirectional Encoder Representations from Transformers. [31]

In the following sections AI models used during this work will be covered briefly.

1.3.1 Neural Networks

NNs are inspired by nature. They imitate the complex connected networks of neurons in a living organism. Both biological and artificial neurons receive signals from multiple neurons and transmit signals to multiple neurons. An artificial neuron computes a weighted sum of its inputs, adds a bias term and then applies an activation function to produce its output, see Figure 3 [32]. A NN consists of an input layer, an arbitrary number of hidden layers and an output layer. In a classical NN every neuron will have all neurons from the previous layer as input. Such a layer is called Fully Connected Layer (FCL). [32]

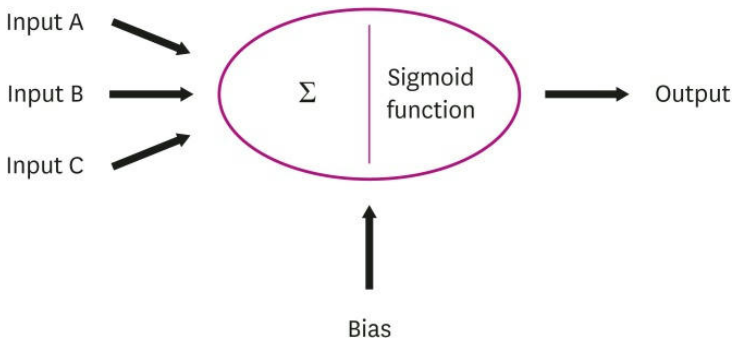


Figure 3: Visualization of an artificial neuron.

The output of a neuron i with n inputs x, the weights w and an activation function could be described using Equation 1.

$$\text{output}_i = \text{activation function}\left(\sum_j^n w_{ij}x_j + b_i\right) \tag{1}$$

The weights and the biases are the learnable parameters of a NN. They are initialised with random values and are updated each training epoch, see Figure 4 [32].

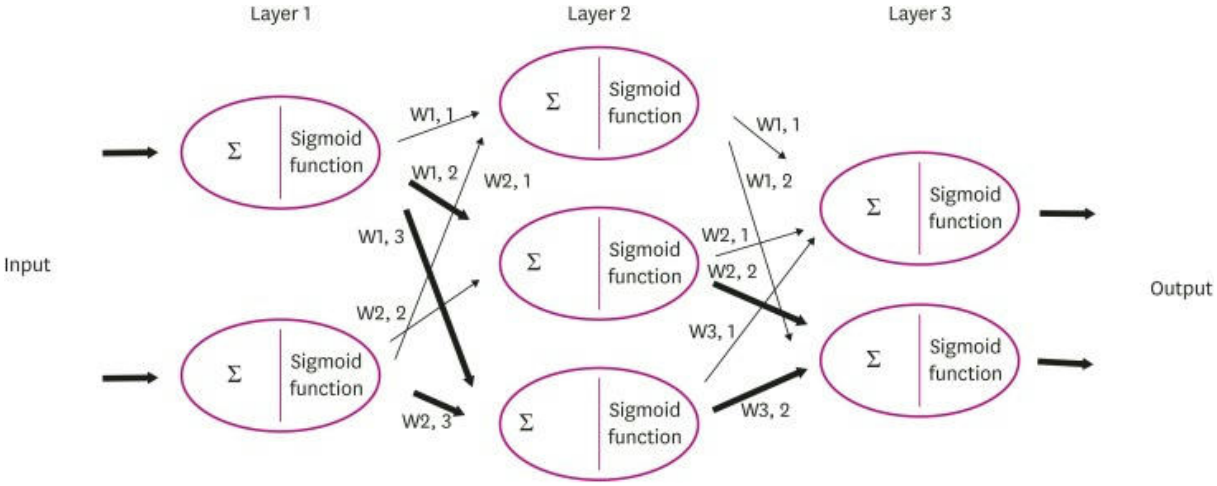


Figure 4: Visualization of the weights in a Neural Network.

Updating these values uses an algorithm called backpropagation, see Figure 5 [32]. After the model makes predictions, the difference between the predicted output and the label is calculated. This value is calculated by using a loss function and is called loss. Backpropagation then computes how much each weight and bias contributed to that loss by calculating the gradients. Rather than calculating an exact solution for all parameters, which is computationally difficult in deep networks, neural networks use an optimization technique called gradient descent. This method uses the computed gradients to update each parameter in the direction that reduces the loss the most. Over many training epochs, the model gradually learns better weights and biases that minimize the prediction error. Gradient descent is not guaranteed to find the global minima of the loss, as it is able to get stuck in a local minima. [32]

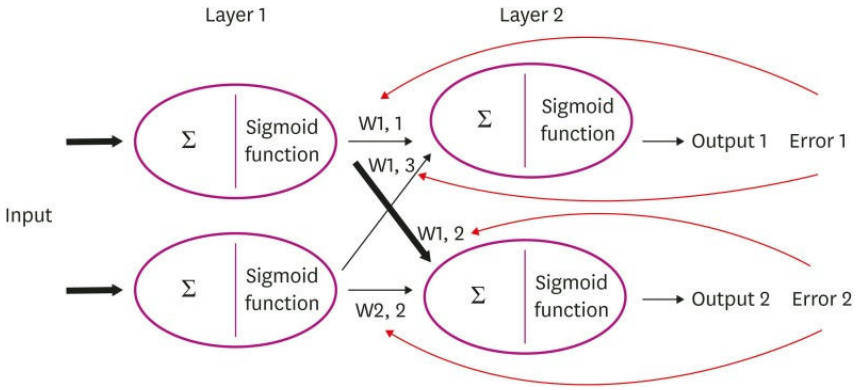


Figure 5: Visualization of the backpropagation mechanism.

Activation Functions (AFs) are an important part of a NN. Their main responsibility is adding non-linearity to the network. Without these a NN would just produce the outputs as a linear function of the inputs. An activation function should also have some more properties. It should be easy to calculate, it should be differentiable and it should retain the distribution of data. The first common AFs were the sigmoid function, see Equation 2, and the tanh function, see Equation 3.

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{2}$$

$$f(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{3}$$

Today the Rectified Linear Unit (ReLU) function is commonly used, as it is computational less complex, see Equation 4.

$$\mathrm{ReLU}(x) = \begin{cases} x \ , \text{ if } x \geq 0 \\ 0 \ , \text{ otherwise} \end{cases} \tag{4}$$

There are also other more advanced AFs, that do have some advantages over the here mentioned ones, but they are not in focus of this work. [33]

There are several types of NNs, each designed to handle different tasks. Some of these used during this work will be introduced in the following sections.

1.3.1.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a popular class of NNs that are often used for image classification, speech recognition and many more tasks. They typically consist of four components. A convolutional layer, a pooling layer, an activation function and a fully connected layer. In a convolutional layer neurons of one layer are only connected to some neurons of the previous layer. These neurons of the previous layer are called the receptive field of the corresponding neuron in the next layer. The output of the receptive field is calculated using a weight vector that is called filter or kernel. This filter is slid over the whole input, see Figure 6 [34]. The weights of the filter are the same for the whole layer. [35]

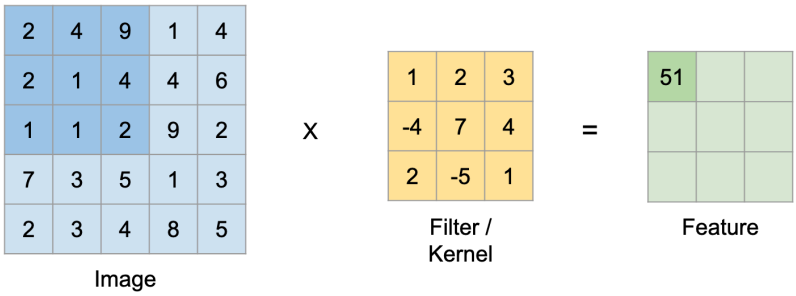


Figure 6: Visualization of a convolutional layer with a two dimensional input.

The pooling layer typically shrinks its input further. There are different pooling techniques. Most common are average pooling and max pooling. Similar to the convolutional layer a window is slid over the input and the pooling function is applied. What differs is that the window usually does not overlap with the next window, leading to a considerable reduction in size, see Figure 7 [36]. [35]

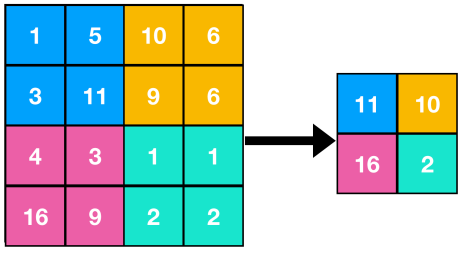


Figure 7: Visualization of max pooling.

Both activation function and fully connected layer were already described in Section 1.3.1. The fully connected layer is usually only used for the final prediction. CNNs are widely used because their shared-weight architecture significantly reduces the number of trainable parameters, lowering computational costs and enhancing generalization. Additionally, their convolutional layers enable them to automatically learn hierarchical feature representations from the input data. [35]

1.3.1.2 Long short-term memory

Long Short-Term Memory (LSTM) networks are part of the Recurrent Neural Network (RNN) family. Unlike standard neural networks, RNNs allow connections across time via feedback loops. This architecture gives the network a form of "memory". LSTMs solve key problems faced by standard RNNs, such as the vanishing and exploding gradient issues during backpropagation through time. They achieve this by enforcing a more stable error flow through their cell state, which enables the model to retain relevant information over longer sequences. [37]

A LSTM unit consists of three sections called gates, the forget gate, the input gate and the output gate. It also carries two memories, the short-term memory and the long term memory. The forget gate determines what percentage of the long-term memory will be remembered. The input gate updates the long-term memory and the output gate updates the short-term memory. The short-term memory of the last unit is the output of the LSTM. Each step takes the short-term memory, the long-term memory and the current input into consideration. Two activation functions are used. The sigmoid and the tanh function, see LSTM [38]. [38]

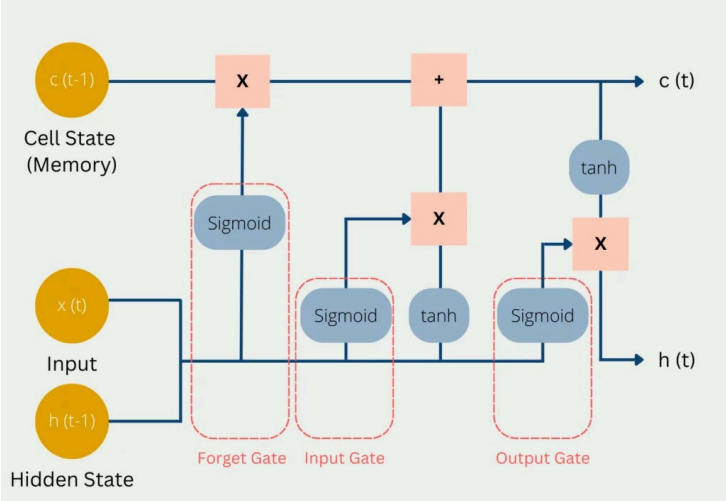


Figure 8: Visualization of a Long Short Term Memory unit.

Bidirectional Long Short-Term Memories (BiLSTMs) are an extension of LSTMs that process sequential data in forward and backward directions. This allows the model to capture context from past and future states simultaneously. This ability is important for tasks like natural language processing, where understanding both past and future context is crucial. [39]

1.3.1.3 Transformers

The introduction of the Transformer architecture has had a big impact that is not limited to scientific research. Large Language Models like Open AIs ChatGPT used this technology to revolutionize chat bots [40]. It could be seen as an advanced version of RNNs. Compared to these, it does enable parallel processing and has improved long-range dependence [41].

The original Transformer consisted of an encoder and a decoder and was created for the purpose of machine translation [41]. Today there are also Transformers that only use one of theses two. ChatGPT for example only uses the decoder as it only needs to generate text, while BERT uses only the encoder as

it only needs to process the input sequence [41]. Here we will cover the original transcoder architecture used for machine translation.

As for most AI models, the input needs to be converted into a numerical representation. Every token, which corresponds to a word or subword, is mapped to a dense vector using a learned embedding matrix. To help the model understand the position of each token in the sequence, positional encoding is added to the embeddings. This encoding uses sine and cosine functions at different frequencies to generate a unique pattern for each position. The result is added to the embedded input. [41]

Next comes the self-attention mechanism. For each token, three vectors are computed: a query, a key, and a value. These are obtained by multiplying the input (embedding + positional encoding) with learned weight matrices. Then, the attention score for each token is calculated by taking the dot product between the query of the current token and the keys of all tokens. These scores are passed through a SoftMax function to normalize them into probabilities, determining how much of the value of each token should contribute to the current one. This process is done multiple times in parallel with different sets of weights, a technique known as Multi-Head Attention. The result is then added back to the original input. [41]

After attention, the output passes through a feed-forward neural network, which consists of two linear layers with a non-linearity in between. This step is applied to each position separately and identically. Again, a residual connection is used, and layer normalization is applied before or after each sub-block, depending on the implementation, to stabilize training and improve performance. [41]

In the decoder, Masked Multi-Head Attention is used instead of regular self-attention. This ensures that each token can only attend to earlier positions in the sequence. Another important component is encoder-decoder attention, which allows the decoder to focus on relevant parts of the encoded input. It works similarly to self-attention: a query is computed for the current token being generated, and dot products are taken with the keys from the encoder's output. After applying SoftMax, the decoder learns which encoded tokens are most important for generating the next word. Like other attention mechanisms, this can also be stacked. [41]

Finally, the output from the decoder is passed through a linear projection layer followed by a SoftMax function, producing a probability distribution over the entire vocabulary. The most likely next token is selected based on this distribution. Layer normalization and residual connections are applied throughout the model to ensure stable learning and better generalization, see Figure 9 [42] for a detailed overview. [41]

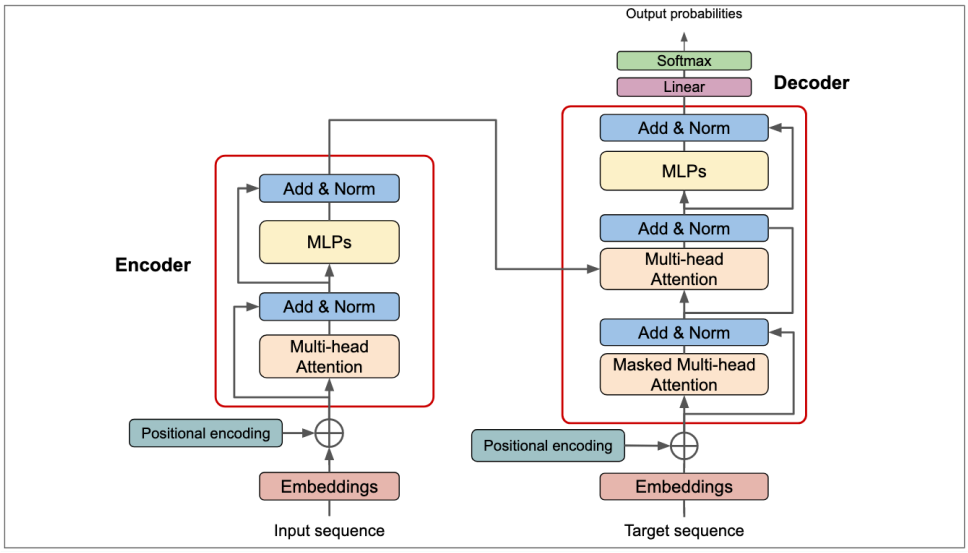


Figure 9: Visual overview of the architecture of an encoder-decoder Transformer.

1.3.2 XGBoost

XGBoost (eXtreme Gradient Boosting) is a powerful and widely used ML algorithm based on the concept of gradient boosted decision trees. Boosting models build an ensemble by sequentially adding many weak learners where each new learner is trained to correct the errors made by the previous learners. It has several features that set it apart from other boosting implementations: It incorporates regularization to prevent overfitting and improve generalization. It can automatically handle missing values and is able to work in parallel. It employs tree pruning via a post-pruning process that starts with deep trees and removes branches that do not contribute to reducing the loss, thereby preventing overfitting. A schematic of the XGBoost algorithm is shown in Figure 10 [43]. [24]

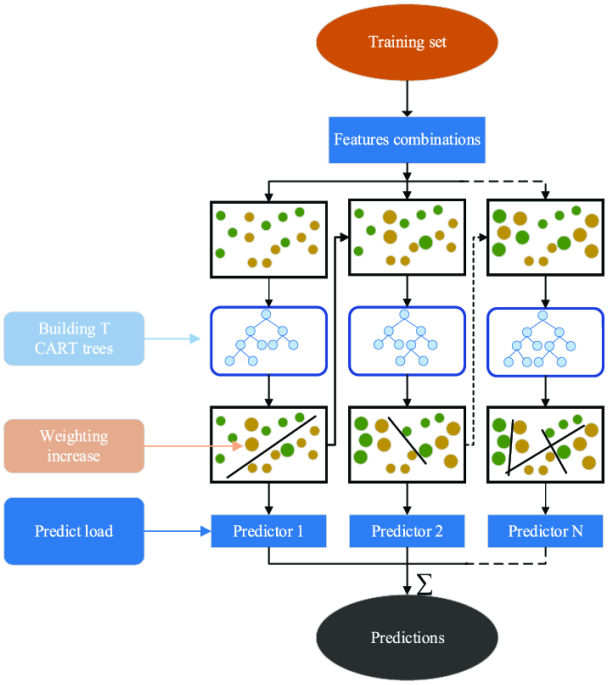


Figure 10: Schematic of the XGBoost algorithm.

1.3.3 Categorical Embeddings

As already mentioned, most AI models need numerical data as input. As inputs do not always fulfill this several embedding techniques have been developed. In this section we will focus on static word

embeddings that were used in this work. Static here means, that each word in the vocabulary only has one vector representation.

Words are, in a way, categorical data. Representing them as numerical values could be accomplished by using something like a one-hot encoding. In a one-hot encoding a vector of the size of the number of categories is created for each element. The position that corresponds to the category of this element is filled with a one, while all other positions contain a zero. However, this approach does have two drawbacks. First, these vectors can become very large and second, all words are treated the same. This means there is no way similarities between words can be expressed. [44]

A better representation for words is to use dense vectors of real numbers. Such a vector can be significantly shorter than a one-hot encoded vector and is filled with real numbers that each represent a property of the element. In a biological context, if we assume that the elements are for example amino acids, those properties could be thought of as chemical properties. A representation of valine (Equation 5) and threonine (Equation 6) could look like this: [44]

$$q_{\text{valine}} = \begin{bmatrix} \overset{\text{Acidity}}{2.4} & \overset{\text{Size}}{, 1.2, ...} \end{bmatrix} \tag{5}$$

$$q_{\text{threonine}} = \begin{bmatrix} \overset{\text{Acidity}}{6.5} & \overset{\text{Size}}{, 1.3, ...} \end{bmatrix} \tag{6}$$

This way similarities between words can be expressed and calculated, see Equation 7. Φ is the angle between the two words. That way similar words will result in a value close to one while dissimilar words will result in a value close to zero. [44]

$$\text{Similarity}(\text{valine}, \text{threonine}) = \frac{q_{\text{valine}} \cdot q_{\text{threonine}}}{\|q_{\text{valine}}\| \, \|q_{\text{threonine}}\|} = \cos(\Phi) \tag{7}$$

In an actual word embedding these vectors do not contain real world properties like in this example. Instead a model learns the values that result in the smallest loss. [44]

1.3.4 Block Decomposition of Protein Sequences

The block decomposition algorithm by Martin Girard, a researcher at the Max Planck Institute for Polymer Research, was used. This algorithm stems from an unpublished article where a surrogate model for low complexity protein sequences was created. The model itself is based on combinatorics on words, particular sturmian words and their generalizations. It was able to show that low complexity protein sequences have similar properties to homopolymers with equivalent parameters. For example the radius of gyration. Changes to the radius of gyration are strongly correlated to changes in the LLPS propensity of a protein. The algorithm was kindly provided to test it as feature in this work. [45]

The block decomposition algorithm itself uses word balance as a measure of homogeneity. It finds the longest segments of the sequence, that have a word balance below a certain threshold. As the algorithm originates from the field of polymer physics these segments are called blocks. A word w is q-balanced if, for any two equal-length substrings u and v within w, the count of each character differs by no more than q. Once the largest homogeneous block is identified, the algorithm recursively searches for the largest blocks to the left and right of it. Segments shorter than a predefined length threshold are discarded. Before applying the block decomposition algorithm, a mapping is applied to be less sensitive to mutations. [45]

2 Material

2.1 Datasets

Four datasets were used during this this work. They are listed in Table 2. The composition and reason for their inclusion is provided in the following sections.

Table 2: Datasets used during this work.

Article	Dataset Name	Description
PSPire [2]	Supplementary Data 4	Training and Testing Data
PhasePred [5]	Dataset S02	Training Data
	Dataset S03	Test Data
	Dataset S06	MLO Data
PPMC-lab [6]	datasets	Proteins classified into driver, client or negative
CatGranule 2.0 [17]	13059_2025_3497_MOESM4_ESM.xlsx	Training and Testing Data

2.2 Protein Features

The protein features, like protein sequence, structural data and PTMs were downloaded from the sources listed in Table 3. The sequences for the PPMC-lab dataset were already included and therefore not downloaded again. For the PhasePred dataset the PTMs were not downloaded.

Table 3: Sources for protein features used during this work.

Protein Feature	Source	Version / Date
Protein Sequence	UniProt [46]	PSPire: 19.05.2025 PhasePred 23.05.2025 PSPire MLO: 28.05.2025 PhasePred MLO 28.05.2025 catGranule 2.0: 24.06.2025
Structural Data	AlphaFold [31]	V2.0
PTM	UniProt [46]	PSPire: 12.06.2025 PSPire MLO: 12.06.2025 PPMC-lab: 16.06.2025 catGranule 2.0: 25.06.2025

2.3 PhaSePred Data Set

The dataset to develop the PhaSePred model contains proteins of 49 organisms. They were taken mainly from PhaSepDB [47]. Some were also taken from LLPSDB [48] and PhaSePro [49]. To reduce redundancy the CD-HIT [50] algorithm was used. The positive LLPS proteins consist of 201 self-assembling and 327 partner-dependent proteins. The positive training set consisted of 128 self-assembling and 214 partner-dependent proteins. The remaining positives were used for independent testing. [5]

The negative dataset consisted of 10 proteomes (*Homo sapiens*, *Saccharomyces cerevisiae*, *Mus musculus*, *Drosophila melanogaster*, *Caenorhabditis elegans*, *Rattus norvegicus*, *Xenopus laevis*, *Arabidosis thaliana*, *Escherichia coli*, *Schizosaccharomyces pombe*). The CD-HIT algorithm was again used to reduce redundancy. 60,251 proteins remained. 20 % were used in the testing set, while the other 80 % were used in the training set. [5]

This data set was planned to be used for testing the influence of grouping LLPS proteins into the two categories of self-assembling and partner-dependent LLPS proteins.

2.4 PSPire Data Set

The PSPire dataset is based on the dataset of the PhaSePred models. It also incorporates proteins from LLPSDB [48], PhaSePro [49], PhaSepDB [47], and DRLLPS [51]. The PSPire training data set contains 259 positives with 195 of these labeled as IDPs and 64 labeled as non-IDPs as well as 8323 negative entries. The testing data set contains 258 positive entries with 194 labeled as IDPs and 64 labeled as non-IDPs as well as 1961 negative entries.

For evaluation, five human MLO datasets were collected: G3BP1 proximity labeling set [52], DACT1-particulate proteome set [53], RNAgranuleDB Tier1 set [54], the PhaSepDB low and high throughput MLO [47] set as well as the DRLLPS MLO set [51].

The PSPire dataset was used due to its focus on LLPS proteins that contain no IDR and because it enables a broad comparison between many LLPS predictors. It also features the MLO datasets to further evaluate and compare model performance.

2.5 PPMC-lab Data Set

The PPMC-lab data set was specifically designed to help developing better Phase Separation Predictors. It provides a curated negative dataset as well as categorizations for LLPS proteins into driver and clients. It contains 784 positives, where 367 are clients, 358 are drivers and 59 are both. The negative set contains 2121 proteins where 1120 are structured proteins and 1001 are disordered proteins. The proteins are collected from DRLLPS [51], LLPSDB [48], PhaSepDB [47], PhaSePro [49], CD-Code [55], DisProt [56] and PDB [57].

The PPMC-lab data set was chosen because it provides a curated negative data set and to test it against the PSPire dataset.

2.6 CatGranule 2.0 Data set

The CatGranule 2.0 data set consists of 3333 positive and 3252 negative proteins in the training data set and 1422 positives and 1376 negatives in the testing data set. The sources of the proteins are similar to the other data sets and include PhaSepDB [47], PhaSePro [49], LLPSDB [48], CD-Code [55], DRLLPS [51] as well as data from the PRALINE database [58], proteins from previous LLPS predictors [59] and from an article about properties of stress granule and P-body proteomes [54]. Like the previous studies CD-Hit was used to filter out too similar proteins. The negative data set is created by using the human proteome and excluding the proteins of the positive data set and their first interactors. [17]

This data set was used to be able to compare the tool created in this work with newer LLPS predictors that claim to be the top performing state-of-the art methods. This includes CatGranule 2.0, MaGS, PICNIC, and PICNIC-GO. [17]

3 Methods

3.1 Use of Artificial Intelligence Tools

During the course of this work, AI assistance was employed to support various aspects of the writing and development process. Specifically, ChatGPT was used for translating text passages, debugging code, and generating suggestions for rewriting or rephrasing sentences to improve clarity and structure. Additionally, it was consulted to retrieve standard parameter values and practices commonly used in machine learning models. All outputs were carefully reviewed and adapted to ensure accuracy, relevance, and alignment with the goals of this thesis.

3.2 Programs

A variety of programs and packages were used during this work. They are listed in Table 4. All computations were conducted on an Apple MacBook Pro with an ARM-based, unbinned M1 Pro processor and 16 GB of RAM.

Table 4: Programs used in this work.

Program	Package	Version
Python	-	3.13.3
	numpy [60]	2.2.6
	bio [61]	1.8.0
	pandas [62]	2.2.3
	matplotlib [63]	3.10.3
	requests	2.32.3
	scikit-learn [64]	1.6.1
	seaborn [65]	0.13.2
	torch [66]	2.7.0
	xgboost [24]	3.0.2
	captum [67]	0.8.0
DSSP [68]	-	4.5.3

3.3 Data Preparation

Protein sequences, protein structures and PTMs were all downloaded using the Python requests module.

As the datasets that were used already undergone data preparation, only three additional preparation steps were conducted. The first was to filter out all entries that had a letter in the amino acid sequence that was not one of the twenty amino acids that were mapped. The second step filtered out all sequences that were longer than 2700 residues. The third step was to pad all sequences to a length of 2700, as CNNs require input vectors of the same length. This length was chosen due to comparability as PSPire uses the same maximum length.

The split used for the PPMC-lab dataset was 80% of the data for the training set and 20% of the data for the testing set. For all models that incorporate RSA values the datasets were cleansed of entries for which no structure data was available on AlphaFold. The length distribution of the filtered datasets was visualized to assess the impact of padding. The IDR labels were inherited for the PSPire dataset from their source. For the PPMC-lab dataset all proteins that had a disorder fraction of zero were labeled as non-IDR proteins.

3.3.1 Block Decomposition

The block decomposition algorithm by Martin Girard [45] was used to factorize the protein sequences into blocks of a certain uniformity. To be able to use the block decomposition as a feature for NNs the output was transformed from a list of tuples containing the start and end of a block to a sequence, where each position represents the block that it is in. As the original algorithm did not provide information about the content of a block, a label for each block was generated, that represented the most common mapping or mappings in the block.

The label was derived as follows. After creating the block decomposition every block was checked if it consisted to more than 66% of the same group. If it did the groups mapping number was assigned as label to this block. If it did not, it was tested if more than 66% of the block were from two groups. In that case the two group numbers were concatenated, with the first one corresponding to the larger group and used as the label for the block. If neither was the case, the block was given a number that represented no group as a label.

The mappings that were used as well as the source and a short description are shown in Table 5.

Table 5: Mappings used by the block decomposition algorithm.

Name	Description	Source	Mapping
APNA	Categorization into aliphatic, positive, negative and aromatic	General	0: A, C, G, H, I, L, M, N, P, Q, S, T, V 1: D, E 2: K, R 3: F, W, Y
RG	RG-Motif	[69]	0: Other 1: R, G
RG2	Significantly more abundant in phase separating proteins that contain an RG-Motif	Internal	0: Other 1: D, E, F, G, I, K, M, N, R, Y
IDR	Amino acids that are more common in IDRs	[70]	0: F, I, L, M, V, W, Y 1: A, C, N, T 2: D, E, G, H, K, P, Q, R, S
MM5	Most Meaningful grouping with five groups	[71]	0: A, G, P, S, T 1: C 2: D, E, H, K, N, Q, R 3: F, L, M, V, Y 4: W
PiPi G	Categorization after PiPi	[9]	0: A, C, G, I, K, L, M, P, S, T, V 1: D, E, N, Q, R 2: F, H, W, Y
PiPi F	Pi Pi	[9]	0: C, I, K, L, M, P, T, V 1: A, D, E, H, N, Q, S, W 2: F, G, R, Y

3.4 Evaluation Metrics

In statistics there are several evaluation metrics for binary classification. The most common will be briefly covered in this section. Most of them use the four terms True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN) for their calculation. The TP and TN values are the values the predictor got right in each category. The FP and FN values describe the values that are predicted to be in the wrong category. [72]

The Accuracy describes the fraction of all correctly predicted samples, see Equation 8.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{8}$$

The Recall describes the fraction of data with the positive label that was identified as such, see Equation 9.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{9}$$

The Specificity describes the fraction of data with the negative label that was identified as such, see Equation 10.

$$\text{Specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}} \tag{10}$$

The Precision describes the fraction of the data that has a correctly predicted positive label in all predicted positive samples, see Equation 11.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{11}$$

While these values are all dependent on the threshold, a value that decides the probability level above a value gets the positive label, there are also two important metrics that do not depend on it. They are both curves that can be evaluated by visual inspection as well as the Area Under the Curve (AUC) value. The Receiver Operator Curve (ROC) plots the recall against the false positive rate, which is equal to one minus specificity. The Precision Recall Curve (PRC) plots the precision against the recall. AUC values lie between zero and one. A higher AUC value is better. While a AUC of 0.5 for the ROC means that the predictions of the model are not better than random guessing, the interpretation of the PRCAUC depends on the positive class frequency. A PRCAUC value of the positive class frequency would equal random guessing. The usage of PRCAUC values is especially important if imbalanced datasets are used, where there are more negative samples than positives. Both metrics combined are a sufficient and threshold independent way to compare model performance. [73]

3.5 Model Selection and Optimization

The process of selecting and testing different models during this work can be divided into three phases. The first phase was about testing if NNs, especially CNNs are capable of predicting LLPS and comparing the block decomposition to the raw sequence as input. The second phase revolved around testing other NN architectures that are more complex than the models from the previous phase and testing the block decomposition as input for a ML model. The third phase focused on enhancing the best model with additional features and optimizations.

As the PSPire dataset proved to be challenging and provided comparability towards many other LLPS predictors it was chosen as the main dataset for deciding which model or optimization to keep. Therefore, not all models were run on all datasets. While the PhasePred dataset was tested in the beginning of this work, it was abandoned due to the similarity to the PSPire dataset and lack of time. Therefore, no results of this dataset are part of this work.

During all runs a random seed of 13 was set to achieve reproducibility. The training data loaders were set to shuffle, while the validation data loaders were not. The batch size was set depending on the complexity of the model and the size of the data set, these values can be found in the raw data, see appendix. All models, excluding the XGBoost model, used cross entropy loss for the loss function and considered the distribution of positive and negative values using class weights. The ADAM optimizer [74] was used in all models except the XGBoost, as it requires less manual optimization than stochastic

gradient descent. The learning rate and decay was adjusted per model and can be found in the raw data as well, see appendix. The models will be evaluated with the AUC values from the ROC and PRC. The evaluation will mostly be split into proteins that contain IDRs and proteins that do not contain IDRs for comparability with the results of the PSPire. Testing other splits like the driver and client split from the PPMC-lab dataset as well as the split into partner-dependent and self-assembling proteins was only touched briefly and was not included in this work.

During this development Batch Normalization (BN) was used [75] for smoother training as well as Dropout (DO) for reducing overfitting. An overview of the tested models and optimizations is shown in Figure 11. The following sections will cover each phase and go into detail about what models were used, the inputs as well as the parameters.

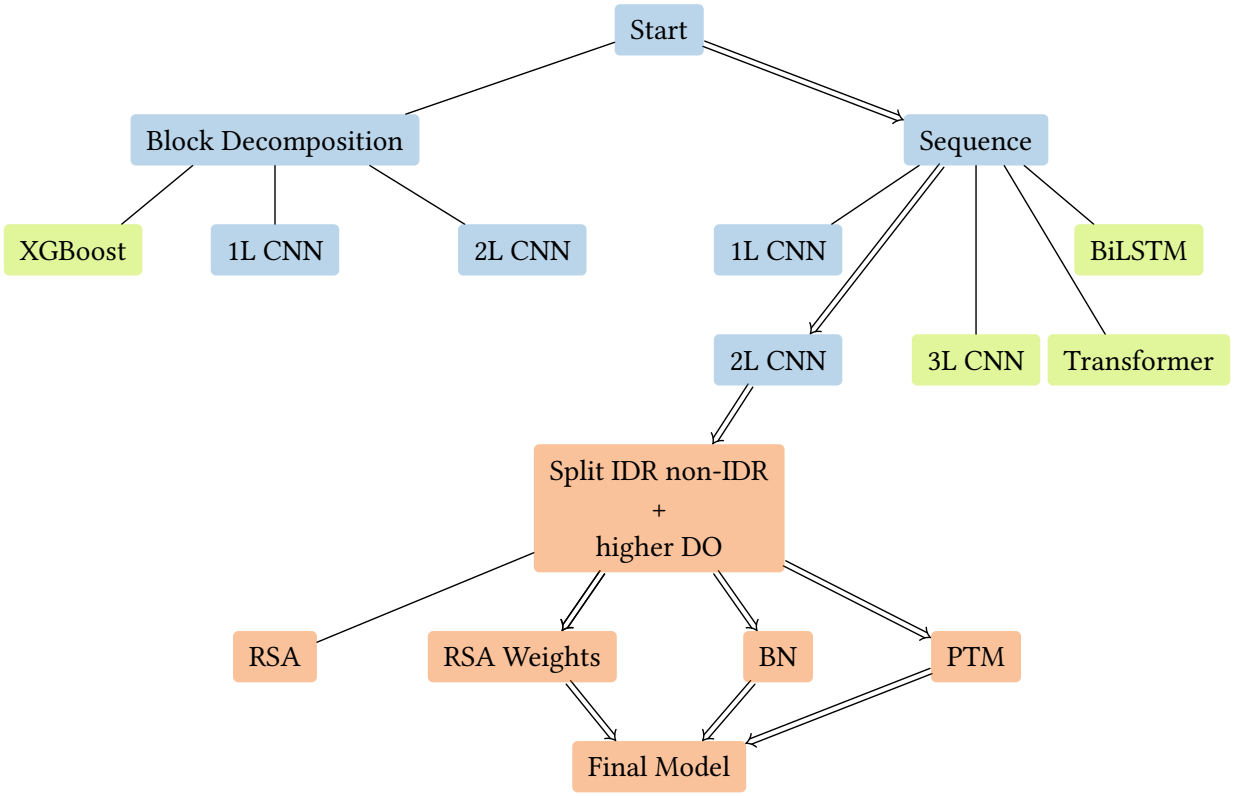


Figure 11: Overview of the models tested during this work. The path to the final model is shown via the double arrows. The phases are represented by the fill of the nodes. Phase one is colored blue, phase two is colored green and phase three is colored orange.

3.5.1 Phase One

As there have been no previous LLPS predictors that relied on NNs, the first step was to create simple models to see if CNNs are capable of this task. Throughout this work the 1-dimensional versions of Convolutional Layers (CLs), Max Pooling Layers (MPLs) and Adaptive Max Pooling Layers (AMPLs) were used. Another goal of these simple models was to test if the block decomposition proves to be beneficial for the prediction over using only the sequence as input. Two models were created for each input. A one Layer CNN and a two layer CNN. The basic models only consisted of an embedding layer, CLs followed by the ReLU activation functions and MPLs and ended in an AMPL, DO and a FCL.

3.5.1.1 1 Layer Convolutional Neural Network

The architecture of the one layer CNN for the sequence based approach is shown in Figure 12. The values for the Output Channels (OC), Embedding Dimension (ED), Kernel Size (KS), Stride (ST) and Padding (PD) are given in Table 6.

Table 6: Parameters for 1 Layer CNNs.

Input	ED	CL				MPL		DO
		OC	KS	PD	ST	KS	ST	
Sequence	10	70	10	2	2	2	2	0.3
Block Decomposition	3	70	10	2	2	2	2	0.3

The model created for the Block Decomposition input already had 14 channels, two for every mapping. Each channel got its own embedding dimension. The architecture of the one layer CNN is visualized in Figure 12.

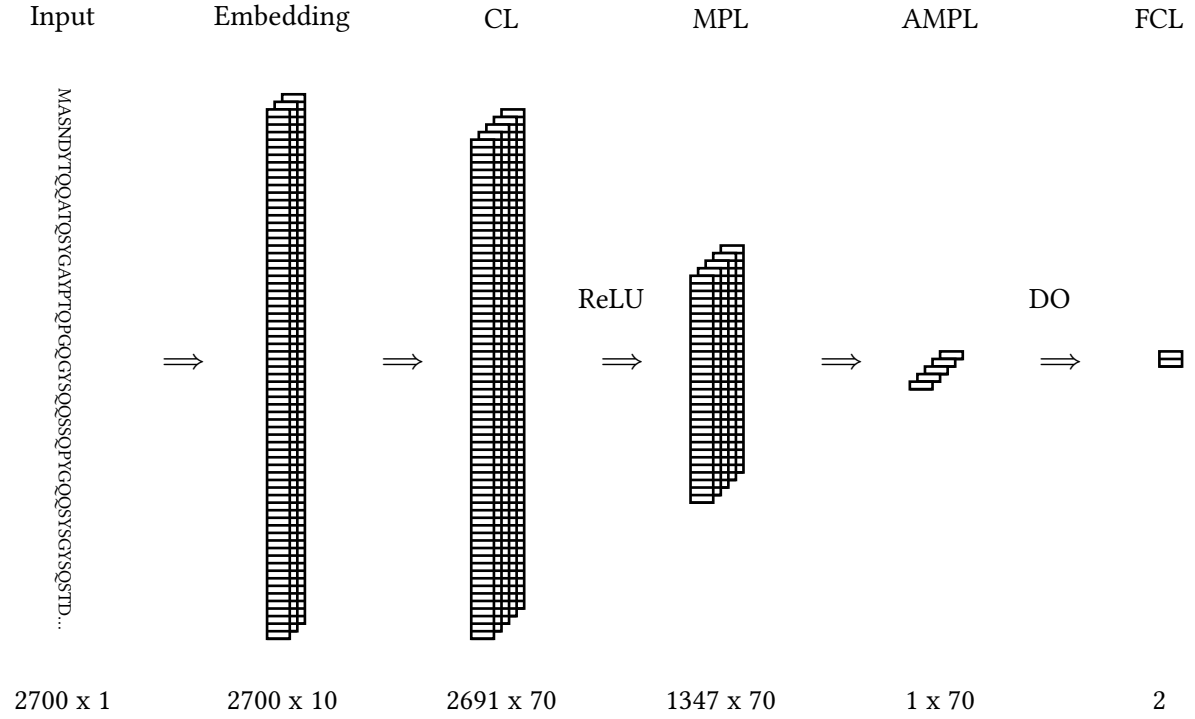


Figure 12: Visualization of the 1 Layer CNN used with the sequence as input.

3.5.1.2 2 Layer Convolutional Neural Network

The parameters of the 2 layer models are summarized in Table 7.

Table 7: Parameters for 2 Layer CNNs.

Input	ED	CL 1				MPL		CL 2				DO
		OC	KS	PD	ST	KS	ST	OC	KS	PD	ST	
Sequence	10	70	10	2	2	2	2	140	10	2	2	0.3
Block Decomposition	3	70	10	2	2	2	2	140	10	2	2	0.3

It only added one additional CL to see if it benefits the model, see the visualization in Figure 13.

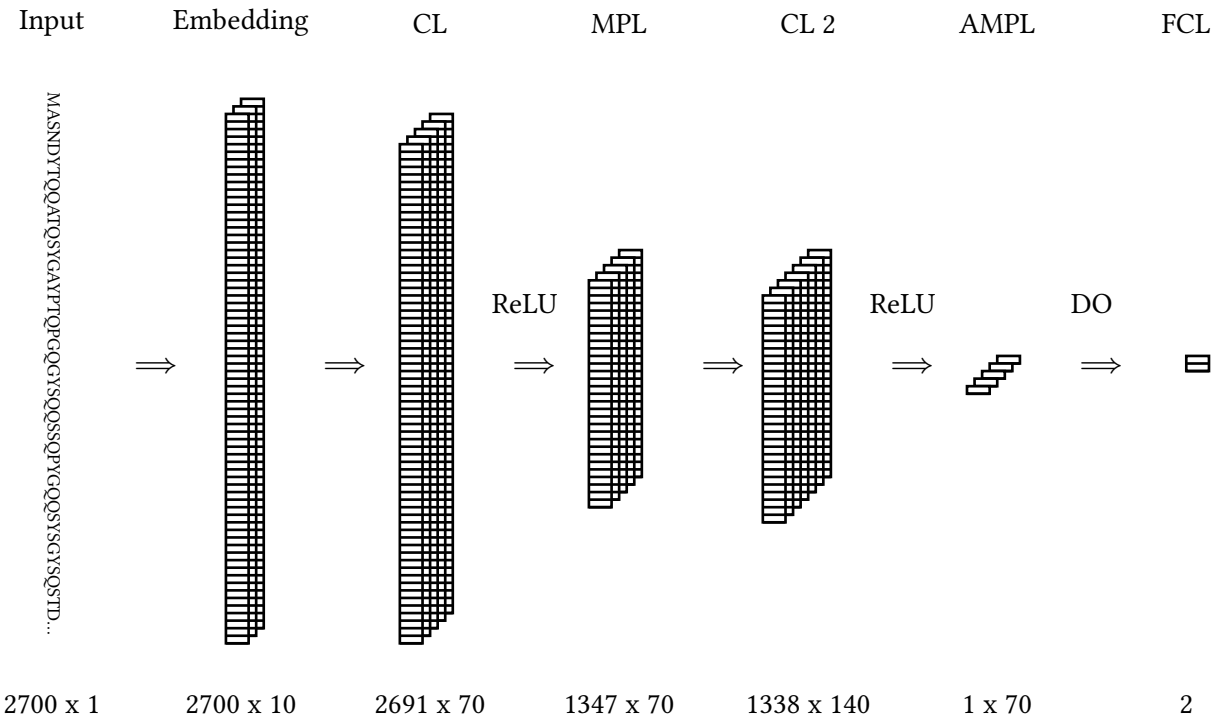


Figure 13: Visualization of the 2 Layer CNN used with the sequence as input.

3.5.2 The second set of Models

After the first set of models was tested, it was decided to discard the idea to use the block decomposition with NNs as they were outperformed by the models that used the raw sequence as input, see Section 4.2. Instead the block decomposition was used to create a model based on the XGBoost algorithm while the NN models based on the raw sequence were focused.

3.5.2.1 XGBoost

To be able to use the block decomposition output for a model like XGBoost, the output had to be adapted again. For every channel representing a mapping the fraction of each label was calculated and used as an input. A simple XGBoost model was created with the parameters in Table 8, for all parameters see appendix.

Table 8: Parameters of the XGBoost model.

Tree Method	Learning Rate	Estimators	Max Depth	Evaluation Metric
Histogram based	0.05	1000	4	LogLoss

3.5.2.2 3 Layer Convolutional Neural Network

This model added one additional CL, ReLU and MPL to the two layer CNN. The parameters are shown in Table 9.

Table 9: Parameters for 3 Layer CNNs.

Input	ED	CL 1				MPL 1 / 2		CL 2				CL 3				DO
		OC	KS	PD	ST	KS	ST	OC	KS	PD	ST	OC	KS	PD	ST	
Sequence	10	70	10	2	2	2	2	140	10	2	2	210	10	2	2	0.3

3.5.2.3 Bidirectional Long Short Term Memory Model

A very basic BiLSTM model was created with the parameters shown in Table 10. The input was embedded and subjected to the LSTM layer. DO and a FCL followed.

Table 10: Parameters of the BILSTM model.

ED	Hidden Dimensions	Layers	DO
12	3	4	0.3

3.5.2.4 Transformer

A basic transformer model was created with the parameters shown in Table 11. The model did integrate a positional encoding.

Table 11: Parameters of the transformer model.

ED	Hidden Dimensions	Heads	Feed Forward Dimensions	Layers	DO
12	3	4	256	2	0.3

3.5.3 Optimizing the Two Layer Convolutional Neural Network

As the second set of models did not provide a better model than the two layer CNN, see Section 4.3, they were discarded. The work was then focused on improving the two layer CNN. The first step was to double the dropout rate and split the model into one that is responsible for learning LLPS prediction for IDR proteins and one for non-IDR proteins. The dropout rate was doubled due to the models tendency of overfitting. The dataset was split to reduce the bias towards IDR proteins and enable the model to learn important features for predicting non-IDR proteins.

In the second step different optimizations were tested independently. BN was added to smoothen the training process. RSA values were added as additional information for the model, both as an additional feature sequence and as a weight vector for the embedded sequence. To obtain the RSA values the structure files of all proteins were downloaded from AlphaFold. Using the tool DSSP [68] the RSA per amino acid was calculated.

Lastly the integration of PTMs was tested. In contrast to previous studies, the inclusion of PTMs was not limited to phosphorylation sites. While phosphorylation was the focus of early studies on LLPS other PTMs like acetylation or ubiquitination could also regulate LLPS [76]. They were downloaded from UniProt. As there are many different PTMs, a mapping for similar PTMs was created. It searched the PTM description for specific strings. Table 12 describes this mapping.

These optimizations were performed on both the PSPire dataset and the PPMC-lab dataset. To test how these optimizations interact with each other, they were combined in every meaningful way. This last test was only conducted on the PSPire dataset.

Table 12: Mapping of PTMs.

Group	Search Patterns
Phosphorylation	phospho
Acetylation	acetyl
Methylation	methyl
Ubiquitin-like	ubiquitin, sumo, nedd8, isg15
Adp-Ribosylation	adp-ribosyl, polyadp
glycosylation	glcnac, galnac, glycos, xyl, fuc, man
Oxidation	sulfoxide, hydroxy, oxid
Other	-

The parameters for the final models are shown in Table 13.

Table 13: Parameters for the final two layer CNNs.

Input	ED	CL 1				MPL		CL 2				DO
		OC	KS	PD	ST	KS	ST	OC	KS	PD	ST	
Sequence	10	70	10	2	2	2	2	140	10	2	2	0.6

As the optimizations affected both models differently, two different architectures were the result. The architecture for the non-IDR model is shown in Figure 14. Due to visual reasons the step of embedding the PTM values is not shown here. In comparison the model for the IDR is missing the concatenation of the PTM layers and the BNs.

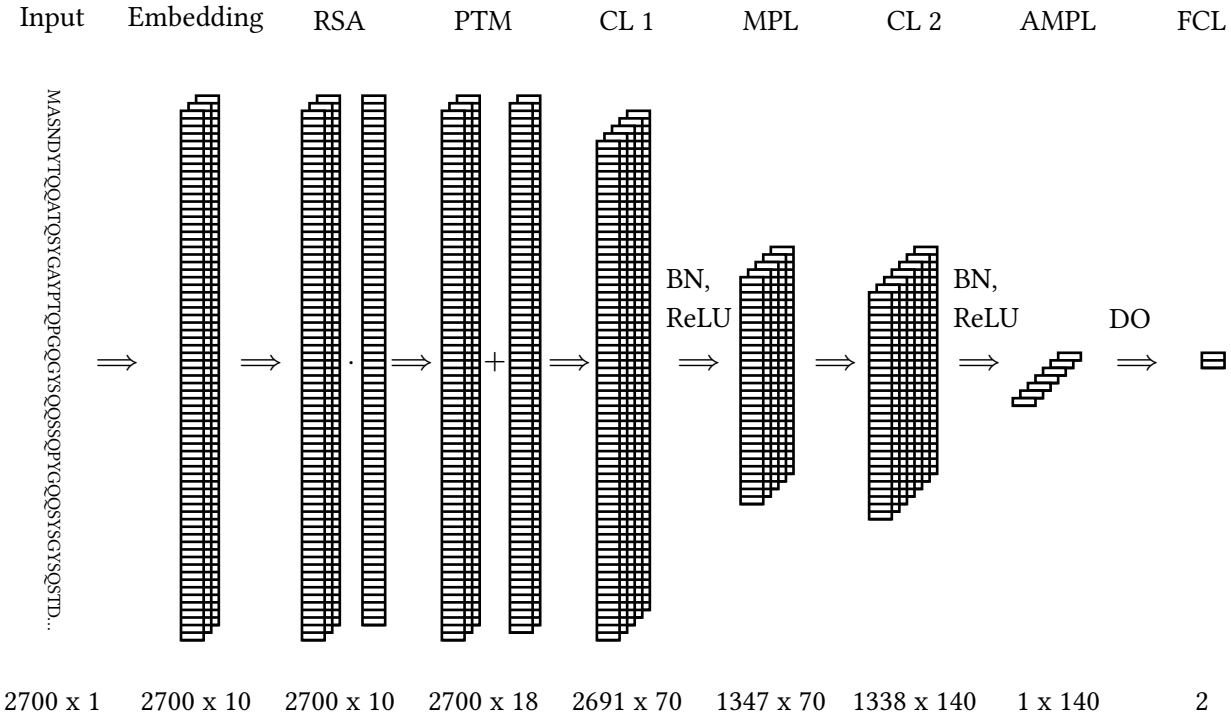


Figure 14: Visualization of the final non-IDR model.

3.6 Evaluation of the models

The final models described in Section 3.5.3 were used for evaluation. This means one model for non-IDR proteins, see Figure 14, and the sightly modified version without PTMs and BN for IDR proteins.

The evaluation of the models was conducted on the PSPire dataset, the PPMC-lab dataset as well as the catGranule 2.0 dataset. For the PSPire dataset and the catGranule 2.0 dataset the models were trained and tested on the same data splits as the models in the respective paper. This way the results are as comparable as possible. As there are no other models that were trained on the PPMC-lab dataset and no other LLPS predictors were run on the test set used here, the evaluation will only show the AUC values obtained by the final model. Only the two best scoring LLPS predictors from the PSPire paper were shown in the evaluation on the PSPire dataset. For the catGranule 2.0 dataset all LLPS predictors compared in its paper were used.

Like in the PSPire paper, the MLO datasets were used as additional evaluation datasets. As the proteins in these are all potential LLPS proteins they were used as the positive testing set. The negative testing set of the PSPire paper was used as the negative testing set for the MLOs. The models trained on the PSPire and PPMC-lab dataset were evaluated on the MLO testing sets. One MLO dataset was also evaluated with the models trained on the PPMC-lab dataset and The negative PPMC-lab test data set, to see if the results differ.

As the catGranule 2.0 dataset as well as the results in its paper do not differ between IDR proteins and non-IDR proteins both in training and testing, a modified version of the final non-IDR model was used for its evaluation. In this modified version the split of IDR and non-IDR proteins was removed.

3.6.0.1 Visualization of Input Features

To investigate the influence of the sequence composition as input feature on the model's predictions, the Python package Captum [67] was used. Captum is a model interpretability library developed for PyTorch, providing a variety of attribution methods that help identify which features contribute most significantly to a prediction. For this work, selected proteins were analyzed using saliency maps, which highlight important positions or patterns in the input sequence that influenced the model's output. These analyses were conducted to gain insights into the model's decision-making process and to assess the biological plausibility of the learned representations.

From the Dr.LLPS MLO dataset twenty random relatively high scoring LLPS proteins were chosen. Ten IDR proteins and ten non-IDR proteins. The salinity map was created for them using both final models trained on the PSPire dataset. One of the IDR proteins and one of the non-IDR proteins was chosen for further comparison. Their saliency maps were compared to their protein features using the UniProt feature viewer. The two proteins were also submitted to the catGranule 2.0 web tool, as it provides a LLPS propensity score over the sequence. These visualizations were also compared to the saliency maps. At last the saliency maps from both the final IDR model and the final non-IDR model of the chosen non-IDR protein were compared to each other, to see how the regions that were most important for their predictions differed.

4 Results

4.1 Data Preparation

During data preparation, proteins that were longer than 2700 residues as well as proteins that contained letters in their amino acid sequence that are not one of the 20 common amino acids were filtered out. Including the RSA values also resulted in filtering the datasets, as some proteins had no structure file available on AlphaFold. The amount of proteins that were filtered out as well as the number of proteins that at least have one annotation for a PTM are shown in Table 14. Only a small fraction of samples were filtered out. Many proteins in the datasets are missing annotations for PTMs.

Table 14: Summary of the number of samples after each filter step. The PTM rows were not filtered, they are only included to show the fraction of proteins that do contain annotations for PTMs.

Dataset	**Filter**	**All**	**Train**		**Test**	
			Positive	Negative	Positive	Negative
PPMC-lab	None	2876	604	1696	151	425
	Length, Unknown Letters	2823	597	1661	149	416
	RSA	2600	572	1508	143	377
	PTM	1072	425	454	99	94
PSPire	None	10801	259	8323	258	1961
	Length, Unknown Letters	10748	259	8275	258	1956
	RSA	10748	259	8275	258	1956
	PTM	5804	70	4591	64	1079
catGranule 2.0	None	9383	3333	3252	1422	1376
	Length, Unknown Letters	9255	3273	3215	1404	1363
	RSA	9255	3273	3215	1404	1363
	PTM	6124	2817	1666	980	679

The distribution of the sequence length for each dataset after the filter first filter step can be seen in Figure 15. It shows that most tested proteins are smaller than 1000 residues.

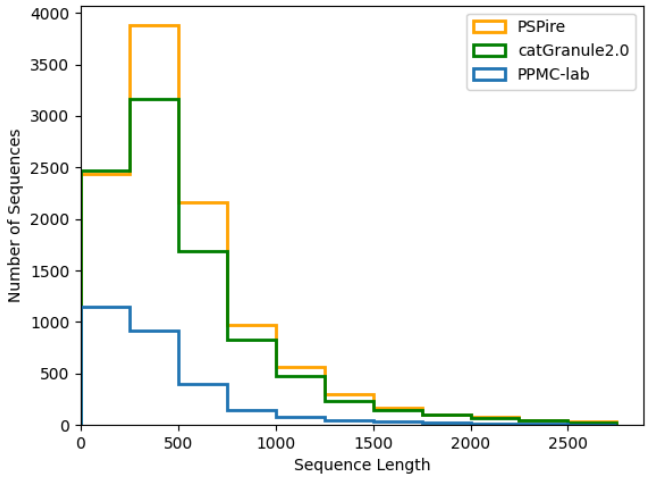


Figure 15: Histogram of the sequence length in the tested datasets.

4.2 The First Set of Models

The first set of models, consisting of the one and two layer CNNs, were compared on both the PPMC-lab dataset and the PSPire dataset. The results are shown in Table 15. The models using the raw sequence mostly outperformed the models using the block decomposition as input.

Table 15: Comparison of the AUC values for the one and two layer CNN with the block decomposition and the raw sequence as input. The one layer models that used the sequence as input where used as the baseline. Better performance is visualized with green, worse with red filling.

Data Set / Model	**AUC**	**IDR**		**non-IDR**	
		Sequence	Block Decomposition	Sequence	Block Decomposition
PPMC-lab / 1 Layer	ROC	0.87	0.88	0.65	0.61
	PRC	0.66	0.63	0.72	0.71
PPMC-lab / 2 Layer	ROC	0.88	0.86	0.69	0.69
	PRC	0.68	0.63	0.73	0.78
PSPire / 1 Layer	ROC	0.70	0.68	0.63	0.59
	PRC	0.20	0.20	0.06	0.04
PSPire / 2 Layer	ROC	0.79	0.72	0.62	0.56
	PRC	0.37	0.26	0.05	0.04

4.3 The Second Set of Models

The second set of models was only run on the PSPire dataset. The results of these runs are shown in Table 16. While the three layer CNN performed slightly better on the non-IDR proteins it performed worse on the proteins containing IDRs. None of the new models outperform the two layer model.

Table 16: Comparison of the AUC values for the models created during the second phase and the two layer CNN. The two layer model was used as baseline. Better performance is visualized with green, worse with red filling.

model	**AUC**	**IDR**	**non-IDR**
Two Layer CNN	ROC	0.79	0.62
	PRC	0.37	0.05
Three Layer CNN	ROC	0.73	0.68
	PRC	0.29	0.07
XGBoost (Block Decomposition)	ROC	0.74	0.54
	PRC	0.25	0.03
BiLSTM	ROC	0.78	0.49
	PRC	0.25	0.03
Transformer	ROC	0.80	0.55
	PRC	0.29	0.03

4.4 Optimizing the Two Layer Convolutional Neural Network

The first optimizations to the two layer CNN, that consisted of doubling the dropout value as well as splitting the model into IDR and non-IDR proteins, were carried out on the PSPire dataset and the PPMC-lab dataset. The results are shown in Table 17. While the results for the proteins with IDRs almost remained the same, the results for the proteins without IDRs improved on the PSPire dataset.

Table 17: Results of adjusting the dropout to 0.6 and splitting the dataset into IDR and non-IDR. Better performance is visualized with green, worse with red filling.

Dataset	**AUC**	**IDR**		**non-IDR**	
		Base	Dropout + Split	Base	Dropout + Split
PPMC-lab	ROC	0.88	0.85	0.69	0.66
	PRC	0.68	0.66	0.73	0.74
PSPire	ROC	0.79	0.78	0.62	0.71
	PRC	0.37	0.38	0.05	0.13

The results of the further optimization on the PSPire dataset are displayed in Table 18. Here the different integrations of the RSA values as well as BN and the integration of PTMs was tested.

Table 18: Results of optimizing the model using different approaches. The two layer CNN is used as base model. Better performance is visualized with green, worse with red filling.

Dataset	**AUC**	**IDR**					**non-IDR**				
		Base	RSA	RSA weight	BN	PTM	Base	RSA	RSA weight	BN	PTM
PPMC-lab	ROC	0.85	0.88	0.88	0.79	0.88	0.66	0.69	0.71	0.65	0.67
	PRC	0.66	0.66	0.69	0.53	0.70	0.74	0.82	0.84	0.73	0.76
PSPire	ROC	0.78	0.78	0.80	0.73	0.67	0.71	0.84	0.85	0.77	0.79
	PRC	0.37	0.39	0.42	0.25	0.20	0.13	0.20	0.18	0.19	0.09

The last step of this optimization was to test the combination of multiple approaches. This was only conducted on the PSPire dataset. The results for this are shown in Table 19.

Table 19: Results of the combined approaches. Better performance is visualized with green, worse with red filling.

AUC	**IDR**				**non-IDR**			
	Base	RSA w + BN	RSA w + PTM	all	Base	RSA w + BN	RSA w + PTM	all
ROC	0.78	0.82	0.67	0.79	0.71	0.86	0.87	0.88
PRC	0.37	0.42	0,21	0.37	0.13	0.17	0.19	0.25

4.5 Evaluation of the model

The final evaluation was carried out on the PSPire dataset, the PPMC-lab dataset, the MLO datasets and the catGranule 2.0 dataset.

4.5.1 Evaluation on the PSPire Dataset

Figure 16 shows the ROCs and PRCs of the final models trained on the PSPire dataset. Table 20 shows the AUC values for this model and compares them to the values that PSPire and PdPS achieved on the same test sets. Their results are taken from the PSPire article [2].

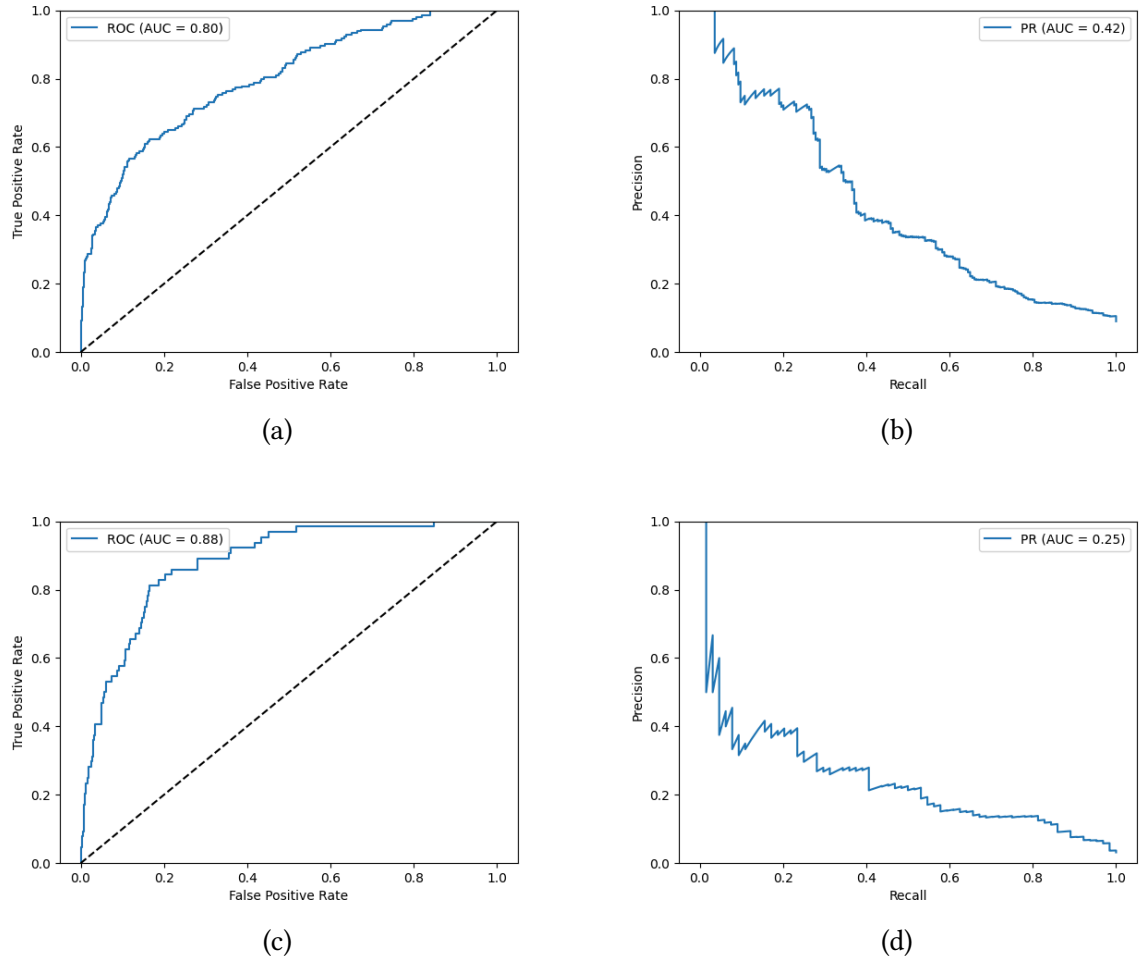


Figure 16: Results of the final model on the PSPire data. (a, b) ROCAUC and PRCAUC for Proteins containing IDRs. (c, d) ROCAUC and PRCAUC for Proteins containing no IDRs.

Table 20: Comparison of the AUC values for the final models, PSPire and PdPS. The values for PSPire and PdPS are taken from the PSPire article. The final models are used as base line. Better performance is visualized with green, worse with red filling.

AUC	**IDR**			**non-IDR**		
	Final Model	PSPire	PdPS	Final Model	PSPire	PdPS
ROC	0.80	0.86	0.84	0.88	0.84	0.68
PRC	0.42	0.51	0.42	0.25	0.24	0.08

4.5.2 Evaluation on the PPMC-lab Dataset

Table 21 shows the final values of the PPMC-lab dataset.

Table 21: Results from the final model on the PPMC-lab dataset.

AUC	IDR	non-IDR
ROC	0.88	0.74
PRC	0.69	0.86

4.5.3 Evaluation on the Membraneless Organells data sets

Table 22 compares the AUC values for the five MLO datasets, that were received using the final models trained on the PSPire training dataset. They are compared to the values of the PSPire and PdPS predictor. Their values are taken from the PSPire article [2]. For proteins that do contain IDRs the model created in this work does perform worse than both PSPire and PdPS. For proteins containing no IDRs however this model outperforms PdPS consistently and has comparable performance to PSPire.

Table 22: Evaluation Summary of the final model on the MLO datasets. The values for PSPire and PdPS are taken from the PSPire article. The final models of this work are used as base line. Better performance is visualized with green, worse with red filling.

Dataset	AUC	IDR			non-IDR		
		Final Model	PSPire	PdPS	Final Model	PSPire	PdPS
G3BP1 proximity labeling	ROC	0.78	0.91	0.86	0.96	0.93	0.81
	PRC	0.34	0.58	0.41	0.51	0.66	0.18
DACT1-particulate proteome	ROC	0.72	0.88	0.85	0.90	0.93	0.81
	PRC	0.22	0.35	0.33	0.49	0.60	0.18
RNAgranuleDB Tier1	ROC	0.77	0.84	0.82	0.88	0.90	0.68
	PRC	0.42	0.48	0.42	0.18	0.28	0.08
PhaSepDB low and high throughput MLO	ROC	0.70	0.72	0.74	0.85	0.80	0.65
	PRC	0.70	0.79	0.80	0.73	0.71	0.47
DrLLPS MLO	ROC	0.68	0.75	0.76	0.80	0.85	0.68
	PRC	0.72	0.78	0.77	0.72	0.74	0.45

Table 23 compares the results obtained when using the models trained on the PSPire dataset to the results obtained using the PPMC-lab dataset. The final models that was trained on the PSPire dataset outperform the models trained on the PPMC-lab dataset significantly. Especially for the non-IDR proteins, where the model trained on the PPMC-lab performs similar to random guessing.

Table 23: Comparison of the models performance trained on the PSPire dataset and the models trained on the PPMC-lab dataset.

Dataset	AUC	IDR		non-IDR	
		PSPire	PPMC-lab	PSPire	PPMC-lab
G3BP1 proximity labeling	ROC	0.78	0.65	0.96	0.49
	PRC	0.34	0.18	0.51	0.05
DACT1-particulate proteome	ROC	0.72	0.60	0.90	0.51
	PRC	0.22	0.11	0.49	0.07
RNAgranuleDB Tier1	ROC	0.77	0.77	0.88	0.53
	PRC	0.42	0.42	0.18	0.03
PhaSepDB low and high throughput MLO	ROC	0.70	0.66	0.85	0.46
	PRC	0.70	0.70	0.73	0.31
DrLLPS MLO	ROC	0.68	0.66	0.80	0.49
	PRC	0.72	0.67	0.72	0.33

As both models in the previous comparison used the negatives of the PSPire test datasets, it was tested if the performance of the model trained on the PPMC-lab dataset would improve if the negatives of its own dataset were used instead. The DACT1-particulate proteome was used for this. The results are shown in Table 24. There were no improvements.

Table 24: Results of evaluating the models trained on the PPMC-lab dataset on the DACT1-particulate MLO dataset while using the PPMC-lab negative test set.

AUC	IDR	non-IDR
ROC	0.54	0.54
PRC	0.29	0.30

4.5.4 Evaluation on the catGranule 2.0 Dataset

Table 25 compares the results of the final models trained on the catGranule 2.0 training dataset to the results of the other predictors that were evaluated in the catGranule 2.0 paper [17]. The non-IDR model of this work is able to slightly outperform the other models. This model achieved a PRCAUC value of 0.81 for the non-IDR and 0.71 for the IDR model.

Table 25: Comparison of the ROCAUC of several LLPS predictors on the catGranule 2.0 test data set. The values for all predictors but my own are taken from the catGranule 2.0 article.

AUC	catGranule 1.0	MaGS	PS-PHunter	PICNIC	PICNIC-GO	catGranule 2.0	non-IDR model	IDR model
ROC	0.66	0.74	0.74	0.73	0.75	0.76	0.80	0.74

4.5.5 Visualization of Input Features

Figure 17 shows the saliency scores visualized along the amino acid sequence of P04264, which is labeled as a IDR LLPS protein [2]. Therefore, the results of the IDR model are shown here. The predicted probability for it to be a LLPS protein was 98 % with this works model. There are two bright colored bands visible at around the residues 190 to 200 and 555 to 570. Many more less bright bands are visible as well. The left bright band lies within a intermediate filament rod domain [77]. The right bright band is near an IDR [78].

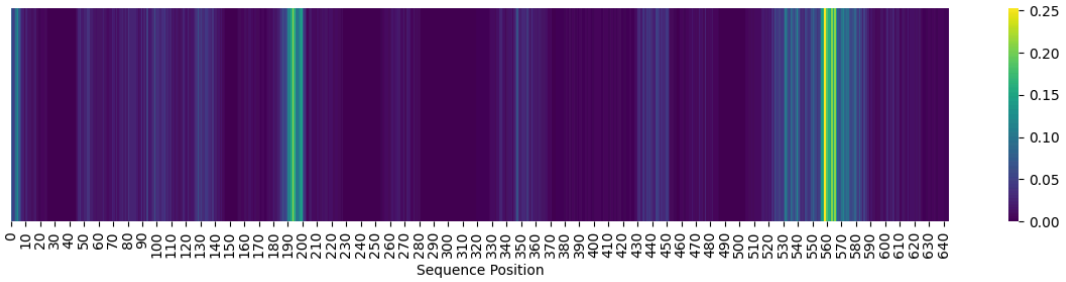


Figure 17: Visualization of the Input Features using the IDR model on the protein P04264.

For comparison, using the catGranule 2.0 web app yielded a score of 0.845 (1 is the highest possible value) and the LLPS propensity profile seen in Figure 18 [17]. While they both show some similar regions to be important for LLPS, the left bright band in the saliency map is missing in the plot of catGranule 2.0.

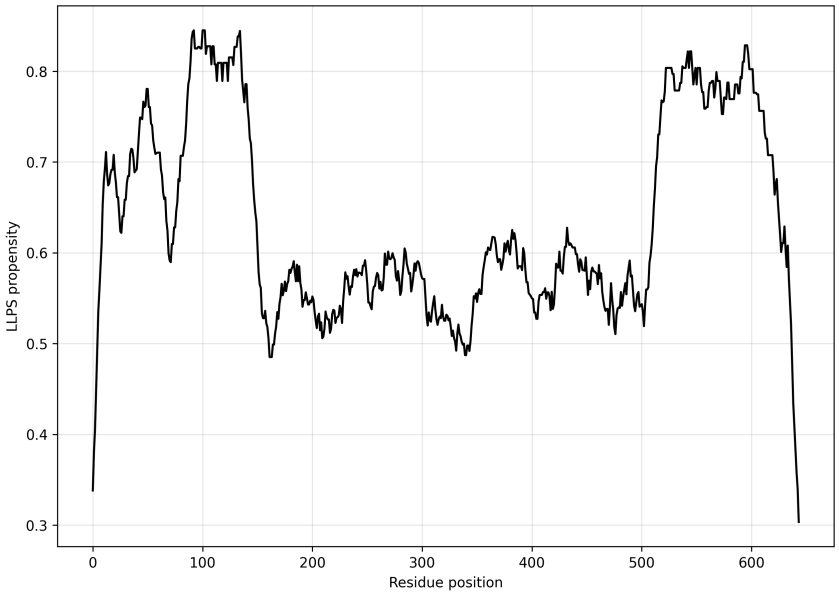
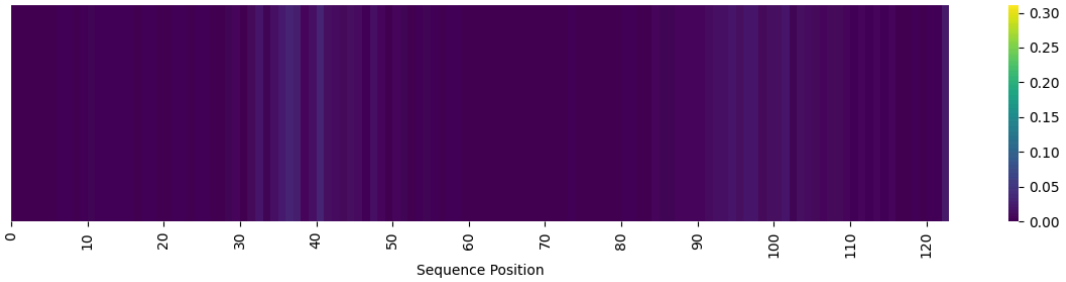
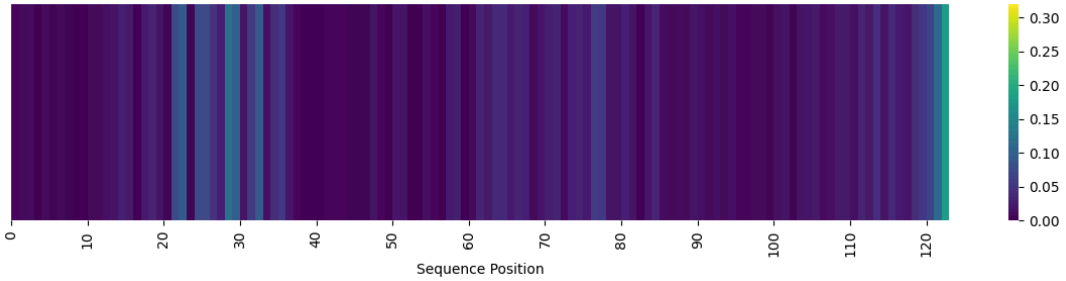


Figure 18: LLPS Propensity profile of the protein P04264 predicted by catGranule 2.0.

Figure 19 shows the same graph for a different protein. In this case the protein P42766, which is a non-IDR labeled LLPS protein [2]. The figure shows the saliency score visualizations for both models. It is important to mention that saliency scores are neither comparable between samples nor between models. The non-IDR model scored a probability of 44 % while the IDR model scored a probability of 21 %. Viewing this protein with the UniProt feature viewer revealed only one entry under domains. An IDR from residue 86 to the end [79].



(a)



(b)

Figure 19: Comparison of the feature relevance for both the non-IDR model (a) and the IDR model (b) on the protein P42766.

The protein was also tested with catGranule 2.0s web app and yielded a score of 0.835 as well as the LLPS propensity profile seen in Figure 20 [17]. There are again similarities as well as differences between these representations.

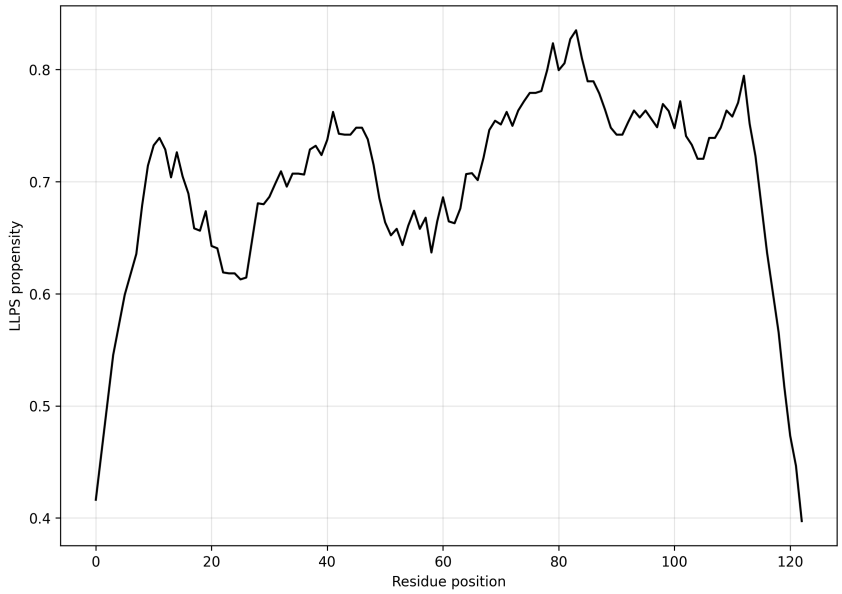


Figure 20: LLPS Propensity profile of the protein P42766 predicted by catGranule 2.0.

5 Discussion

5.1 Block Decomposition vs. Raw Sequence

The comparison between simple models using either block decomposition or the raw sequence input revealed that using the block decomposition does not benefit the models. In the case of NNs, which are capable of automatically learning and extracting complex features from raw data, applying block decomposition effectively limits the available information. This reduction in input detail likely explains the slight performance decrease observed with these models.

Similarly, using the block decomposition with the ML model XGBoost did not yield strong results either. This suggests that the simplified representation introduced by the block decomposition algorithm is not suitable for LLPS prediction.

5.2 Choosing the best Model

After the simple two layer CNN performed the best in the first tests, some more complex alternatives, including a three layer CNN, a BiLSTM model and a Transformer, were tested. Although some of these models achieved results comparable to the two layer model, none was able to surpass its performance. A previous study that focused on DNA-protein binding also came to the conclusion that the performance of CNNs decreases with network complexity when there is insufficient training data. In their case the more complex models only started to outperform the simple models when they used a training set of around 40,000 samples [80]. This reasoning is probably applicable for BiLSTM models and Transformers too, as they are also more complex. The datasets used in this study contained only around 10,000 entries at most. This limited size is likely insufficient for effectively training deeper or more complex models.

It should also be noted that the alternative models were only evaluated using a single set of hyperparameters. It is possible that with more extensive tuning, some of these architectures could achieve better performance. However, given the small dataset and the greater data requirements of these models, a more exhaustive evaluation was not conducted in this study.

5.3 Optimizing the final Model

The two-layer model was selected based on its strong performance. Several experiments were conducted to further enhance it. As the models exhibited a tendency to overfit, seen by the train loss, the dropout rate was increased. A higher dropout rate reduces the model's reliance on individual neurons, encouraging it to learn more generalizable features and thereby mitigating overfitting.

Since the features contributing to LLPS in proteins often differ between proteins with and without IDRs, the positive dataset was split accordingly [2]. Separate models were then trained on each subset. On the PPMC-lab dataset, this separation had minimal impact on performance. However, for the PSPire dataset, the performance of the non-IDR model improved notably. As a result, the split-model approach was adopted as the new baseline, and subsequent experiments were built upon this version.

Both models incorporating RSA values demonstrated improved performance. However, the model that used RSA values as weights applied to the embedded sequence outperformed the one that simply concatenated the RSA values with the embeddings. These performance gains were observed on both datasets, though they were more pronounced on the PSPire dataset. This improvement is intuitive, as amino acids located on the protein's surface, which is represented by the RSA values, are primarily responsible for molecular interactions. Applying RSA values as weights integrates this information directly into the model.

The addition of BN yielded mixed results. On the PSPire dataset, it improved the performance of the non-IDR model, but led to performance declines when applied to the IDR models. A likely explanation

for this inconsistency lies in the use of sequence padding. Since protein sequences vary in length, all sequences were padded to a uniform length of 2700. As seen in the distribution of sequence length, most proteins in this study were smaller than 1000 residues, which leads to many sequences consisting mostly of padded values. BN calculates the mean and standard deviation per channel across the batch, including the padded values. When batches include many short sequences, these padding values disproportionately affect the normalization statistics, skewing the model's behavior. Batches with longer sequences are less affected. This imbalance can result in unstable or degraded performance.

The inclusion of PTM values did not yield significant improvements on the PPMC-lab dataset. On the PSPire dataset, the performance of the IDR model decreased slightly, while the non-IDR model experienced a minor improvement. Although PTMs are known to play a role in LLPS [76] and their inclusion should theoretically enhance the models performance, the current databases for PTMs are incomplete. Many proteins likely contain PTMs that have not yet been experimentally identified. The absence of this data introduces noise, which can interfere with model training and prediction accuracy.

In the final stage, the different enhancements were tested in various combinations. For the IDR model, no combination outperformed the two-layer model with RSA values used as weights. Consequently, this configuration was selected as the final model for IDR proteins. In contrast, for the non-IDR model, the combination of RSA values as weights, BN, and the inclusion of PTM values produced the best overall performance. This model was therefore chosen as the final version for non-IDR proteins.

5.4 Evaluation of the final Model

To further assess and compare the performance of the final models, they were trained and tested across multiple datasets. The first comparison was conducted using the PSPire dataset. In this setting, the IDR model was unable to match the performance of the PSPire model, though it achieved results that were comparable to those of the PdPS model. Notably, the non-IDR model outperformed both the PSPire and PdPS models, although its advantage over PSPire was marginal. A similar pattern emerged when the models trained on the PSPire dataset were evaluated using the MLO datasets. Once again, the IDR model underperformed relative to both the PSPire and PdPS models. Meanwhile, the non-IDR model surpassed the PdPS model and performed at a level comparable to the PSPire model.

One possible explanation for the weaker performance of the IDR model lies in the intrinsic characteristics of IDRs. Unlike CNNs, which are capable of identifying recurring patterns, IDRs are not constrained by any fixed or consistent structural motifs. Two IDR sequences that lead to LLPS can appear entirely different from each other. As a result, relying solely on structural similarity or pattern recognition may not be sufficient for accurate modeling or requires more data. In contrast, using only the fractional composition of amino acids, combined with additional scalar features, appears to be a more effective strategy at present [2]. The non-IDR model, already performs competitively with existing models, though there is still considerable room for improvement in its overall performance.

Using the models trained on the PPMC-lab dataset to evaluate their performance on the MLO data yielded unexpected results. Compared to the models trained on the PSPire dataset, a significant drop in performance was observed. One plausible reason for this drop could be the smaller size of the PPMC-lab dataset, particularly its limited number of negative samples. With only a third of total samples the models were probably not able to learn the features in a general manner. For the MLO evaluation, the negative test set from PSPire was reused. To investigate whether this influenced performance, one evaluation was conducted using the negative test set from the PPMC-lab dataset instead. This did not change the observation. It would be interesting to see how the more traditional ML predictors would perform using the PPMC-lab dataset as they require fewer data and as the PPMC-lab dataset does feature higher quality selection of non LLPS proteins.

Lastly, the evaluation using the catGranule 2.0 dataset did again show that the model created in this work does yield a competitive performance. It was able to slightly outperform all models that were evaluated in the catGranule 2.0 paper. It is important to mention that due to lack of time, the dataset was not split into IDPs and non-IDPs. This split could lead to further improvements as it did on the PSPire dataset.

The visualization of the saliency scores provided a preliminary look at which regions of a sequence might influence classification. Comparing these plots with the LLPS propensity profiles from catGranule 2.0 showed that the regions with high saliency are similar to the regions catGranule 2.0 identified to have high LLPS propensity. Scaling up the analysis of the saliency scores may holds potential for uncovering meaningful patterns or sequence features associated with LLPS. Another interesting insight was to compare the saliency between the IDR model and the non-IDR model. It revealed, that both models relied on very different areas of the proteins to form their prediction. This could be seen as supportive to create different models for the different LLPS proteins. Comparing the results from the saliency scores with experimental data could further confirm if the learned features actually play important roles in LLPS. Having the ability to visualize the important areas of the sequences also is an advantage over the some of the other LLPS predictors that do not have a similar representation like PSPire.

Due to time constraints, several promising ideas could not be explored in this study. One such is the optimization of hyperparameters, for example through five-fold cross-validation. This technique would likely improve the robustness and generalization ability of the final models and might lead to measurable performance gains. Additionally, alternative strategies for dividing the training data should be investigated, for instance, by distinguishing between self-assembling proteins and those that depend on interaction partners, like the PhasePred study did. Comparing these two grouping strategies could help further studies on LLPS prediction to choose the most impactful.
Another worthwhile extension would be the construction of a new dataset that combines the larger positive data from the catGranule 2.0 study with the large negative set used in the PSPire dataset. Lastly, making the developed tool accessible for external users, for example, as a web application or command-line utility, would increase its practical value and enable its use by other researchers.

5.5 Conclusion

This work has shown that a relatively simple, sequence-based CNN model can perform competitively and sometimes even outperform traditional ML models. While both the CNN models and the ML models yielded similar performance, they offer different advantages and limitations.

One of the clearest benefits of using NN models lies in the significant reduction of feature engineering. The final models in this study relied on only three main inputs: the raw amino acid sequence, PTM annotations, and RSA values. Of these, both the sequence and PTMs information are readily accessible via databases like UniProt, although the PTM features required some minor preprocessing. The RSA values, while effective in improving performance, do require structural predictions and additional processing using tools like AlphaFold and DSSP. In contrast, PSPire used 44 features, many of which are derived from simple amino acid compositions but also include complex features requiring external computation. Despite this feature richness, the CNN model achieved comparable results.

A second major benefit of NN models is their ability to process whole sequences, identifying complex patterns that are inaccessible to models restricted to flat, tabular inputs. This is particularly relevant in the context of LLPS, where interactions between sequence motifs, disorder, and post-translational modifications play significant roles. However, there are also certain drawbacks to these model types. Neural networks typically demand more computational resources and longer training times. Processing sequences of different lengths requires padding which can introduce noise.

When it comes to interpretability both ML models and NN models have their benefits. Traditional ML models are generally more transparent, with well-established methods that allow for clear feature attribution. Neural networks, on the other hand, are inherently more complex due to their depth and non-linear structure. In this study, it was experimented with saliency-based visualization techniques to inspect model behavior. The comparison of these saliency maps showed similarities to LLPS propensity profiles created by catGranule 2.0. This approach may hold potential for further analysis that could be used to find relevant sequence sections for LLPS.

An important theme throughout this work is the limited availability of curated data. The success of the CNN model, despite being relatively lightweight, is encouraging, but likely bounded by the current size and quality of existing datasets. The absence of a well-defined negative dataset also remains a bottleneck. Efforts, such as the PPMC-lab dataset, are therefore important contributions and will help to mitigate this problem. Despite the small dataset itself, the incompleteness of PTMs may also limit current models.

Another major insight was the value of dividing the model to specific protein characteristics. Dividing the data into proteins with and without IDRs and developing specialized models for each group led to consistent improvements in predictive performance. The saliency maps further confirmed that the two models focused on different regions of the protein sequences. The addition of RSA values as attention weights also proved beneficial. While BN and PTM features showed mixed impact, likely due to incomplete annotations and the variability introduced by padding, the overall framework remains extensible and can be adapted as better data and annotations become available.

In summary, this study contributes a practical LLPS predictor showing that NN approaches can match or outperform traditional models while simplifying the feature engineering process. The strong performance achieved with relatively few features and a compact architecture suggests that there is significant potential in neural network-based methods, especially as datasets grow.

Bibliography

[1] Z. Xu, W. Wang, Y. Cao, and B. Xue, "Liquid-liquid phase separation: Fundamental physical principles, biological implications, and applications in supramolecular materials engineering," *Supramolecular Materials*, vol. 2, p. 100049, Dec. 2023, doi: 10.1016/j.supmat.2023.100049.

[2] S. Hou, J. Hu, Z. Yu, D. Li, C. Liu, and Y. Zhang, "Machine learning predictor PSPire screens for phase-separating proteins lacking intrinsically disordered regions," *Nature Communications*, vol. 15, no. 1, p. 2147, Mar. 2024, doi: 10.1038/s41467-024-46445-y.

[3] Y. Li, C. Huang, L. Ding, Z. Li, Y. Pan, and X. Gao, "Deep learning in bioinformatics: Introduction, application, and perspective in the big data era," *Methods*, vol. 166, pp. 4–21, Aug. 2019, doi: 10.1016/j.ymeth.2019.04.008.

[4] S. Alberti, "Phase separation in biology," *Current Biology*, vol. 27, no. 20, pp. R1097–R1102, Oct. 2017, doi: 10.1016/j.cub.2017.08.069.

[5] Z. Chen *et al.*, "Screening membraneless organelle participants with machine-learning models that integrate multimodal features," *Proceedings of the National Academy of Sciences*, vol. 119, no. 24, p. e2115369119, Jun. 2022, doi: 10.1073/pnas.2115369119.

[6] C. Pintado-Grima, O. Bárcenas, V. Iglesias, E. Arribas-Ruiz, M. Burdukiewicz, and S. Ventura, "Confident protein datasets for liquid-liquid phase separation studies." Accessed: Jun. 02, 2025. [Online]. Available: https://www.researchsquare.com/article/rs-4594179/v1

[7] A. K. Lancaster, A. Nutter-Upham, S. Lindquist, and O. D. King, "PLAAC: a web and command-line application to identify proteins with prion-like amino acid composition," *Bioinformatics*, vol. 30, no. 17, pp. 2501–2502, Sep. 2014, doi: 10.1093/bioinformatics/btu310.

[8] B. Bolognesi *et al.*, "A Concentration-Dependent Liquid Phase Separation Can Cause Toxicity upon Increased Protein Expression," *Cell Reports*, vol. 16, no. 1, pp. 222–231, Jun. 2016, doi: 10.1016/j.celrep.2016.05.076.

[9] R. M. Vernon *et al.*, "Pi-Pi contacts are an overlooked protein feature relevant to phase separation," *eLife*, vol. 7, p. e31486, Feb. 2018, doi: 10.7554/eLife.31486.

[10] G. Orlando, D. Raimondi, F. Tabaro, F. Codicè, Y. Moreau, and W. F. Vranken, "Computational identification of prion-like RNA-binding proteins that form liquid phase-separated condensates," *Bioinformatics*, vol. 35, no. 22, pp. 4617–4623, Nov. 2019, doi: 10.1093/bioinformatics/btz274.

[11] M. Hardenberg, A. Horvath, V. Ambrus, M. Fuxreiter, and M. Vendruscolo, "Widespread occurrence of the droplet state of proteins in the human proteome," *Proceedings of the National Academy of Sciences*, vol. 117, no. 52, pp. 33254–33262, Dec. 2020, doi: 10.1073/pnas.2007670117.

[12] E. R. Kuechler, P. M. Budzyńska, J. P. Bernardini, J. Gsponer, and T. Mayor, "Distinct Features of Stress Granule Proteins Predict Localization in Membraneless Organelles," *Journal of Molecular Biology*, vol. 432, no. 7, pp. 2349–2368, Mar. 2020, doi: 10.1016/j.jmb.2020.02.020.

[13] G. v. Mierlo, J. R. G. Jansen, J. Wang, I. Poser, S. J. v. Heeringen, and M. Vermeulen, "Predicting protein condensate formation using machine learning," *Cell Reports*, vol. 34, no. 5, Feb. 2021, doi: 10.1016/j.celrep.2021.108705.

[14] X. Chu *et al.*, "Prediction of liquid–liquid phase separating proteins using machine learning," *BMC Bioinformatics*, vol. 23, no. 1, p. 72, Feb. 2022, doi: 10.1186/s12859-022-04599-w.

[15] J. Sun *et al.*, "Precise prediction of phase-separation key residues by machine learning," *Nature Communications*, vol. 15, no. 1, p. 2662, Mar. 2024, doi: 10.1038/s41467-024-46901-9.

[16] A. Hadarovich *et al.*, "PICNIC accurately predicts condensate-forming proteins regardless of their structural disorder across organisms," *Nature Communications*, vol. 15, no. 1, p. 10668, Dec. 2024, doi: 10.1038/s41467-024-55089-x.

[17] M. Monti *et al.*, "catGRANULE 2.0: accurate predictions of liquid-liquid phase separating proteins at single amino acid resolution," *Genome Biology*, vol. 26, no. 1, p. 33, Feb. 2025, doi: 10.1186/s13059-025-03497-7.

[18] I. Walsh, A. J. M. Martin, T. Di Domenico, and S. C. E. Tosatto, "ESpritz: accurate and fast prediction of protein disorder," *Bioinformatics*, vol. 28, no. 4, pp. 503–509, Feb. 2012, doi: 10.1093/bioinformatics/btr682.

[19] J. C. Wootton and S. Federhen, "Statistics of local complexity in amino acid sequences and sequence databases," *Computers & Chemistry*, vol. 17, no. 2, pp. 149–163, Jun. 1993, doi: 10.1016/0097-8485(93)85006-X.

[20] R. Vedantam, C. L. Zitnick, and D. Parikh, "CIDEr: Consensus-based image description evaluation," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2015, pp. 4566–4575. doi: 10.1109/CVPR.2015.7299087.

[21] J. Ludwiczak, A. Winski, K. Szczepaniak, V. Alva, and S. Dunin-Horkawicz, "DeepCoil—a fast and accurate prediction of coiled-coil domains in protein sequences," *Bioinformatics*, vol. 35, no. 16, pp. 2790–2795, Aug. 2019, doi: 10.1093/bioinformatics/bty1062.

[22] "Learning the molecular grammar of protein condensates from sequence determinants and embeddings | PNAS." Accessed: Jul. 18, 2025. [Online]. Available: https://www.pnas.org/doi/10.1073/pnas.2019053118

[23] P. V. Hornbeck, B. Zhang, B. Murray, J. M. Kornhauser, V. Latham, and E. Skrzypek, "PhosphoSitePlus, 2014: mutations, PTMs and recalibrations," *Nucleic Acids Research*, vol. 43, no. Database issue, pp. D512–520, Jan. 2015, doi: 10.1093/nar/gku1267.

[24] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Francisco California USA: ACM, Aug. 2016, pp. 785–794. doi: 10.1145/2939672.2939785.

[25] J. Mihel, M. Šikić, S. Tomić, B. Jeren, and K. Vlahoviček, "PSAIA – Protein Structure and Interaction Analyzer," *BMC Structural Biology*, vol. 8, no. 1, p. 21, Apr. 2008, doi: 10.1186/1472-6807-8-21.

[26] B.-J. Yoon, "Hidden Markov Models and their Applications in Biological Sequence Analysis," *Current Genomics*, vol. 10, no. 6, pp. 402–415, Sep. 2009, doi: 10.2174/138920209789177575.

[27] Y. Zhou, S. Zhou, Y. Bi, Q. Zou, and C. Jia, "A two-task predictor for discovering phase separation proteins and their undergoing mechanism," *Briefings in Bioinformatics*, vol. 25, no. 6, p. bbae528, Nov. 2024, doi: 10.1093/bib/bbae528.

[28] H. Jamialahmadi, G. Khalili-Tanha, E. Nazari, and M. Rezaei-Tavirani, "Artificial intelligence and bioinformatics: a journey from traditional techniques to smart approaches," *Gastroenterology and Hepatology From Bed to Bench*, vol. 17, no. 3, pp. 241–252, 2024, doi: 10.22037/ghfbb.v17i3.2977.

[29] "AI vs. Machine Learning vs. Deep Learning vs. Neural Networks | IBM." Accessed: Jun. 30, 2025. [Online]. Available: https://www.ibm.com/think/topics/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks

[30] N. Elmobark, "Evaluating the Trade-offs Between Machine Learning and Deep Learning: A Multi-Dimensional Analysis," *Journal of Computer, Software, and Program*, vol. 2, no. 1, pp. 10–18, Jun. 2025, doi: 10.69739/jcsp.v2i1.254.

[31] J. Jumper *et al.*, "Highly accurate protein structure prediction with AlphaFold," *Nature*, vol. 596, no. 7873, pp. 583–589, Aug. 2021, doi: 10.1038/s41586-021-03819-2.

[32] S.-H. Han, K. W. Kim, S. Kim, and Y. C. Youn, "Artificial Neural Network: Understanding the Basic Concepts without Mathematics," *Dementia and Neurocognitive Disorders*, vol. 17, no. 3, pp. 83–89, Sep. 2018, doi: 10.12779/dnd.2018.17.3.83.

[33] S. R. Dubey, S. K. Singh, and B. B. Chaudhuri, "Activation functions in deep learning: A comprehensive survey and benchmark," *Neurocomputing*, vol. 503, pp. 92–108, Sep. 2022, doi: 10.1016/j.neucom.2022.06.111.

[34] P. Ratan, "What is the Convolutional Neural Network Architecture?." Accessed: Jul. 02, 2025. [Online]. Available: https://www.analyticsvidhya.com/blog/2020/10/what-is-the-convolutional-neural-network-architecture/

[35] S. Indolia, A. K. Goswami, S. P. Mishra, and P. Asopa, "Conceptual Understanding of Convolutional Neural Network- A Deep Learning Approach," *Procedia Computer Science*, vol. 132, pp. 679–688, Jan. 2018, doi: 10.1016/j.procs.2018.05.069.

[36] "Everything about Pooling layers and different types of Pooling." Accessed: Jul. 02, 2025. [Online]. Available: https://iq.opengenus.org/pooling-layers/

[37] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, doi: 10.1162/neco.1997.9.8.1735.

[38] "Deep learning Recurrent Neural Networks - Skforecast Docs." Accessed: Jul. 02, 2025. [Online]. Available: https://skforecast.org/0.15.0/user_guides/forecasting-with-deep-learning-rnn-lstm.html

[39] "Bidirectional LSTM in NLP." Accessed: Jul. 02, 2025. [Online]. Available: https://www.geeksforgeeks.org/nlp/bidirectional-lstm-in-nlp/

[40] P. P. Ray, "ChatGPT: A comprehensive review on background, applications, key challenges, bias, ethics, limitations and future scope," *Internet of Things and Cyber-Physical Systems*, vol. 3, pp. 121–154, Jan. 2023, doi: 10.1016/j.iotcps.2023.04.003.

[41] A. Vaswani *et al.*, "Attention is all you need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, in NIPS'17. Red Hook, NY, USA: Curran Associates Inc., Dec. 2017, pp. 6000–6010.

[42] J. Nyandwi, "The Transformer Blueprint: A Holistic Guide to the Transformer Neural Network Architecture," *Deep Learning Revision*, Jul. 2023, Accessed: Jul. 02, 2025. [Online]. Available: https://deeprevision.github.io/posts/001-transformer/

[43] X. Yao, X. Fu, and C. Zong, "Short-Term Load Forecasting Method Based on Feature Preference Strategy and LightGBM-XGboost," *IEEE Access*, vol. 10, pp. 75257–75268, 2022, doi: 10.1109/ACCESS.2022.3192011.

[44] "Word Embeddings: Encoding Lexical Semantics — PyTorch Tutorials 2.7.0+cu126 documentation." Accessed: Jul. 03, 2025. [Online]. Available: https://docs.pytorch.org/tutorials/beginner/nlp/word_embeddings_tutorial.html

[45] "Files · dev · Martin Girard / Words · GitLab." Accessed: Jun. 18, 2025. [Online]. Available: https://gitlab.mpcdf.mpg.de/mgirard/Words/-/tree/dev?ref_type=heads

[46] The UniProt Consortium, "UniProt: the Universal Protein Knowledgebase in 2025," *Nucleic Acids Research*, vol. 53, no. D1, pp. D609–D617, Jan. 2025, doi: 10.1093/nar/gkae1010.

[47] C. Hou *et al.*, "PhaSepDB in 2022: annotating phase separation-related proteins with droplet states, co-phase separation partners and other experimental information," *Nucleic Acids Research*, vol. 51, no. D1, pp. D460–D465, Jan. 2023, doi: 10.1093/nar/gkac783.

[48] X. Wang *et al.*, "LLPSDB v2.0: an updated database of proteins undergoing liquid–liquid phase separation in vitro," *Bioinformatics*, vol. 38, no. 7, pp. 2010–2014, Mar. 2022, doi: 10.1093/bioinformatics/btac026.

[49] B. Mészáros *et al.*, "PhaSePro: the database of proteins driving liquid–liquid phase separation," *Nucleic Acids Research*, vol. 48, no. D1, pp. D360–D367, Jan. 2020, doi: 10.1093/nar/gkz848.

[50] L. Fu, B. Niu, Z. Zhu, S. Wu, and W. Li, "CD-HIT: accelerated for clustering the next-generation sequencing data," *Bioinformatics*, vol. 28, no. 23, pp. 3150–3152, Dec. 2012, doi: 10.1093/bioinformatics/bts565.

[51] W. Ning *et al.*, "DrLLPS: a data resource of liquid–liquid phase separation in eukaryotes," *Nucleic Acids Research*, vol. 48, no. D1, pp. D288–D295, Jan. 2020, doi: 10.1093/nar/gkz1027.

[52] P. Yang *et al.*, "G3BP1 Is a Tunable Switch that Triggers Phase Separation to Assemble Stress Granules," *Cell*, vol. 181, no. 2, pp. 325–345, Apr. 2020, doi: 10.1016/j.cell.2020.03.046.

[53] M. Esposito *et al.*, "TGF-β-induced DACT1 biomolecular condensates repress Wnt signalling to promote bone metastasis," *Nature Cell Biology*, vol. 23, no. 3, pp. 257–267, Mar. 2021, doi: 10.1038/s41556-021-00641-w.

[54] J.-Y. Youn *et al.*, "Properties of Stress Granule and P-Body Proteomes," *Molecular Cell*, vol. 76, no. 2, pp. 286–294, Oct. 2019, doi: 10.1016/j.molcel.2019.09.014.

[55] N. Rostam *et al.*, "CD-CODE: crowdsourcing condensate database and encyclopedia," *Nature Methods*, vol. 20, no. 5, pp. 673–676, May 2023, doi: 10.1038/s41592-023-01831-0.

[56] M. C. Aspromonte *et al.*, "DisProt in 2024: improving function annotation of intrinsically disordered proteins," *Nucleic Acids Research*, vol. 52, no. D1, pp. D434–D441, Jan. 2024, doi: 10.1093/nar/gkad928.

[57] H. M. Berman *et al.*, "The Protein Data Bank," *Nucleic Acids Research*, vol. 28, no. 1, pp. 235–242, Jan. 2000, doi: 10.1093/nar/28.1.235.

[58] A. Vandelli *et al.*, "The PRALINE database: protein and Rna humAn singLe nucleotIde variaNts in condEnsates," *Bioinformatics*, vol. 39, no. 1, p. btac847, Jan. 2023, doi: 10.1093/bioinformatics/btac847.

[59] E. R. Kuechler, A. Huang, J. M. Bui, T. Mayor, and J. Gsponer, "Comparison of Biomolecular Condensate Localization and Protein Phase Separation Predictors," *Biomolecules*, vol. 13, no. 3, p. 527, Mar. 2023, doi: 10.3390/biom13030527.

[60] C. R. Harris *et al.*, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020, doi: 10.1038/s41586-020-2649-2.

[61] P. J. A. Cock *et al.*, "Biopython: freely available Python tools for computational molecular biology and bioinformatics," *Bioinformatics*, vol. 25, no. 11, pp. 1422–1423, Jun. 2009, doi: 10.1093/bioinformatics/btp163.

[62] W. McKinney, "Data Structures for Statistical Computing in Python," presented at the Python in Science Conference, Austin, Texas, 2010, pp. 56–61. doi: 10.25080/Majora-92bf1922-00a.

[63] J. D. Hunter, "Matplotlib: A 2D Graphics Environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, May 2007, doi: 10.1109/MCSE.2007.55.

[64] F. Pedregosa *et al.*, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, no. 85, pp. 2825–2830, 2011, Accessed: Jun. 02, 2025. [Online]. Available: http://jmlr.org/papers/v12/pedregosa11a.html

[65] M. L. Waskom, "seaborn: statistical data visualization," *Journal of Open Source Software*, vol. 6, no. 60, p. 3021, Apr. 2021, doi: 10.21105/joss.03021.

[66] "PyTorch 2 paper and tutorial @ ASPLOS 2024 – PyTorch." Accessed: Jun. 02, 2025. [Online]. Available: https://pytorch.org/blog/pytorch-pytorch-2-paper-tutorial/

[67] N. Kokhlikyan *et al.*, "Captum: A unified and generic model interpretability library for PyTorch." Accessed: Jul. 09, 2025. [Online]. Available: https://arxiv.org/abs/2009.07896

[68] "PDB-REDO/dssp: Application to assign secondary structure to proteins." Accessed: Jun. 02, 2025. [Online]. Available: https://github.com/PDB-REDO/dssp?tab=readme-ov-file

[69] P. A. Chong, R. M. Vernon, and J. D. Forman-Kay, "RGG/RG Motif Regions in RNA Binding and Phase Separation," *Journal of Molecular Biology*, vol. 430, no. 23, pp. 4650–4665, Nov. 2018, doi: 10.1016/j.jmb.2018.06.014.

[70] A. Campen, R. M. Williams, C. J. Brown, J. Meng, V. N. Uversky, and A. K. Dunker, "TOP-IDP-Scale: A New Amino Acid Scale Measuring Propensity for Intrinsic Disorder," *Protein and peptide letters*, vol. 15, no. 9, pp. 956–963, 2008, doi: 10.2174/092986608785849164.

[71] N. Cannata, S. Toppo, C. Romualdi, and G. Valle, "Simplifying amino acid alphabets by means of a branch and bound algorithm and substitution matrices," *Bioinformatics*, vol. 18, no. 8, p. 1102, 2002, Accessed: Apr. 25, 2025. [Online]. Available: https://www.academia.edu/14913388/Simplifying_amino_acid_alphabets_by_means_of_a_branch_and_bound_algorithm_and_substitution_matrices

[72] O. Rainio, J. Teuho, and R. Klén, "Evaluation metrics and statistical tests for machine learning," *Scientific Reports*, vol. 14, no. 1, p. 6086, Mar. 2024, doi: 10.1038/s41598-024-56706-x.

[73] T. Saito and M. Rehmsmeier, "The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets," *PLoS ONE*, vol. 10, no. 3, p. e118432, Mar. 2015, doi: 10.1371/journal.pone.0118432.

[74] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization." Accessed: Jul. 19, 2025. [Online]. Available: http://arxiv.org/abs/1412.6980

[75] "BatchNorm1d — PyTorch 2.7 documentation." Accessed: Jul. 19, 2025. [Online]. Available: https://docs.pytorch.org/docs/stable/generated/torch.nn.BatchNorm1d.html

[76] J. Li *et al.*, "Post-translational modifications in liquid-liquid phase separation: a comprehensive review," *Molecular Biomedicine*, vol. 3, p. 13, May 2022, doi: 10.1186/s43556-022-00075-2.

[77] "ProRule PRU01188." Accessed: Jul. 21, 2025. [Online]. Available: https://prosite.expasy.org/rule/PRU01188

[78] "MobiDB Entry." Accessed: Jul. 21, 2025. [Online]. Available: https://mobidb.org/P04264

[79] "MobiDB Entry." Accessed: Jul. 21, 2025. [Online]. Available: https://mobidb.org/P42766

[80] H. Zeng, M. D. Edwards, G. Liu, and D. K. Gifford, "Convolutional neural network architectures for predicting DNA–protein binding," *Bioinformatics*, vol. 32, no. 12, pp. i121–i127, Jun. 2016, doi: 10.1093/bioinformatics/btw255.

Appendix

Full raw data and code are available at: https://github.com/derRiesenOtter/bachelor.

Declaration of Originality

I hereby confirm that the submitted work is original and was written by me without any additional assistance. If work by others was referenced or used, it has been properly cited. My work has not been previously graded or published. The electronically submitted version corresponds to the printed version.

Signature	Place and Date

Declaration of Ownership and Copyright

I hereby give my consent for the Bingen University of Applied Sciences to make this work available to other students and interested third parties, and to publish it in my name (Robin Ender).

Signature	Place and Date