

Predicting liquid-liquid phase separation of proteins using graph neural network

Antti Korkealaakso

School of Science

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 24.5.2023

Supervisor

Prof. Nuutti Hyvönen

Advisor

Dr Anssi Laukkanen

Copyright © 2023 Antti Korkealaakso



Author Antti Korkealaakso

Title Predicting liquid-liquid phase separation of proteins using graph neural network

Degree programme Mathematics and Operations Research

Major Systems and Operations Research

Code of major SCI3055

Supervisor Prof. Nuutti Hyvönen

Advisor Dr Anssi Laukkanen

Date 24.5.2023

Number of pages 59

Language English

Abstract

Liquid-liquid phase separation (LLPS) of proteins has been shown to be related to many diseases and biological processes. In LLPS, the proteins become concentrated in some places and form two distinct liquid phases. The occurrence of this phenomenon is highly dependent on the conditions of the protein solution. Determining the LLPS behaviour of a protein through simulations or experiments is expensive and time-consuming which has raised interest in using machine learning methods to predict LLPS. However, previous machine learning methods on LLPS have used small data sets which has limited the methods that can be applied. In addition, the previous machine learning methods have not considered the conditions at which the LLPS occurs.

Therefore, this thesis aims to develop a machine learning model that can predict the LLPS of proteins under different conditions and evaluate the performance of the model. The developed model uses a graph neural network (GNN) to extract information from the three-dimensional structure of the proteins. The two conditions used in the model are temperature and salt concentration. The conditions are incorporated in the model before and after the GNN. Simulated data is used as the training data for the model, and experimental data is used to evaluate the performance of the model.

The results for the simulated data indicate that the GNN model is capable of extracting information from protein structures on a general level. However, the model is not capable of predicting the complicated condition dependencies that are present in the simulated data. This suggests that the model should be developed further to be able to predict the condition dependencies better. The model was not able to predict the experimental data accurately. The main reason for this was the discrepancy between the simulated and experimental data. Therefore, to improve the performance of the model on experimental data, the LLPS simulations and determining the LLPS information from the simulations should be studied further. Nevertheless, this thesis is the first contribution to developing a machine learning model to predict LLPS under different external conditions.

Keywords liquid-liquid phase separation, graph neural network, machine learning

Tekijä Antti Korkealaakso

Työn nimi Proteiinien neste-neste-faasiseparaation ennustaminen käyttämällä graafineuroverkkoja

Koulutusohjelma Mathematics and Operations Research

Pääaine Systems and Operations Research**Pääaineen koodi** SCI3055

Työn valvoja Prof. Nuutti Hyvönen

Työn ohjaaja TkT Anssi Laukkanen

Päivämäärä 24.5.2023**Sivumäärä** 59**Kieli** Englanti

Tiivistelmä

Proteiinien neste-nestefaasierotuksen (LLPS) on osoitettu liittyvän moniin sairauksiin ja biologisiin prosesseihin. LLPS:ssä proteiinit konsentroituvat tiettyihin paikkoihin ja muodostavat kaksi erillistä nestefaasia. Tämän ilmiön esiintyminen on riippuvainen proteiiniliuoksen olosuhteista. Proteiinin LLPS-käyttäytymisen määrittäminen simulaatioiden tai kokeiden avulla on kallista ja aikaa vievää, mikä on nostanut kiinnostusta koneoppimismenetelmiin, joilla voitaisiin ennustaa LLPS. Aiemmissä LLPS:ää koskevissa koneoppimismenetelmissä on käytetty pieniä data-aineistoja, mikä on rajoittanut sovellettavia menetelmiä. Lisäksi aiemmissä koneoppimismenetelmissä ei ole otettu huomioon olosuhteita, joissa LLPS tapahtuu.

Tämän vuoksi tämän tutkimuksen tavoitteena on kehittää koneoppimismalli, jolla voidaan ennustaa proteiinien LLPS:ää eri olosuhteissa, ja arvioida mallin suorituskykyä. Kehitettyssä mallissa käytetään graafineuroverkkoa (GNN), jonka avulla saadaan hyödynnettyä tietoa proteiinien kolmiulotteisesta rakenteesta. Mallissa käytetyt kaksi olosuhdetta ovat lämpötila ja suolapitoisuus. Olosuhteet sisällytetään malliin ennen ja jälkeen GNN:n. Mallin harjoitusdatana käytetään simuloitua dataa, ja mallin suorituskyvyn arviointiin käytetään kokeellista dataa.

Simuloitudulla datalla saadut tulokset osoittavat, että GNN-malli kykenee poimimaan tietoa proteiinirakenteista yleisellä tasolla. Malli ei kuitenkaan kykene ennustamaan simuloitussa datassa esiintyviä monimutkaisia olosuhderiippuvuuksia. Tämä viittaa siihen, että mallia olisi kehitettävä edelleen, jotta se pystyisi ennustamaan olosuhderiippuvuudet paremmin. Kehitetty malli ei pystynyt ennustamaan kokeellista dataa tarkasti. Pääasiallinen syy tähän oli simuloitun ja kokeellisen datan välinen eroavuus. Tästä syystä LLPS-simulaatioita ja LLPS:n määrittämistä simulaatioista on tutkittava lisää, jotta mallien suorituskykyä kokeellisen datan osalta voitaisiin parantaa. Siitä huolimatta tässä tutkimuksessa on kehitetty ensimmäinen koneoppimismalli, jolla ennustetaan LLPS eri olosuhteissa.

Avainsanat neste-neste-faasiseparaatio, graafineuroverkko, koneoppiminen

Contents

Abstract	3
Abstract (in Finnish)	4
Contents	5
Symbols and abbreviations	6
1 Introduction	7
2 Prediction of protein properties	10
2.1 Liquid–liquid phase separation prediction	10
2.2 Graph neural networks in protein property prediction	13
3 LLPS data	16
3.1 Experimental data	17
3.2 Simulated data	18
4 Theory of the model components	23
4.1 Adjacency matrix	23
4.2 Determining the three-dimensional structure	24
4.3 Recurrent neural networks	25
4.4 Graph neural network	27
5 Machine learning model	32
5.1 Amino acid embeddings	32
5.2 Graph convolutional neural network	34
5.3 Pooling layer	35
5.4 Hyperparameter optimization using random search	35
6 Results	37
6.1 Model architecture	37
6.2 Hyperparameter optimization	42
6.3 Training and validation error	45
6.4 Performance on experimental data	49
7 Conclusions	51
References	53

Symbols and abbreviations

Abbreviations

CG	coarse-grained
CNN	convolutional neural network
GCN	graph convolutional neural network
GNN	graph neural network
GRU	gated recurrent unit
IDP	intrinsically disordered proteins
IDR	intrinsically disordered region
LCR	low complexity region
LLPS	liquid-liquid phase separation
LSTM	long short-term memory
MD	molecular dynamics
MLP	multilayer perceptron
PLD	prion-like domain
RNN	recurrent neural network

1 Introduction

In recent years, a phenomenon for proteins called liquid–liquid phase separation (LLPS) has received much attention since it is related to many diseases [1]–[4] and essential for biological processes [5], [6]. It is well known that cells contain compartments that are not bounded by a membrane. These compartments are observed to be able to have liquid–like properties [7]. These membrane–less compartments are also called bio–molecular condensates, and they condensate through a phenomenon known as LLPS. In LLPS, the different components become more concentrated in one place and form a dense and more diluted phase [8].

Many proteins, if not all, can undergo LLPS under the right conditions [9]. However, many of the proteins will not undergo LLPS under the physiologically relevant conditions. Proteins that undergo LLPS under the physiologically relevant conditions generally share some typical features [10]. One of these typical features is the existence of large intrinsically disordered regions (IDR) [11]. The functions of many proteins are directly related to their three–dimensional structures [12], [13]. Proteins with IDRs do not have well–defined three–dimensional structures and are called intrinsically disordered proteins (IDP). The number of IDPs is constantly growing [14]. IDPs can also be only partially unstructured [15].

LLPS of proteins has been previously experimentally studied using a wide range of methods [16] such as visually observing the condensates using light microscopy or turbidity measurements. Determining the LLPS phase diagram for a protein requires many of these experiments under different conditions in which it is observed whether the protein has formed condensates or not. Therefore, determining the phase diagram for protein is time–consuming and expensive. In addition, manufacturing some proteins can be difficult or even more expensive than determining the phase diagram itself. Therefore, experimentally determining the phase diagram for many proteins is not feasible.

A less expensive alternative for the experiments is to simulate LLPS. There are many different methods for modelling LLPS that differ in the resolution of the model [17]. A higher resolution usually leads to a higher level of detail but at a higher computational cost. Since LLPS is observed as the condensation of multiple proteins, modelling LLPS requires the model to simulate dozens of proteins. Therefore, all–atom simulations are usually computationally too expensive. Recently, the coarse–grained (CG) models have been used to simulate LLPS [18]–[20]. The advantage of CG models is their significantly lower computational cost compared to all–atom simulations. In coarse–grained models, multiple atoms are represented by a single bead. Therefore, the CG models are not as detailed as all–atom models. Finding novel proteins with desired LLPS behaviour may require screening of hundreds of candidate proteins in order to find some candidates that satisfy the requirements. Despite the lower computational cost of CG models, they are still too expensive for high–throughput screening of thousands of proteins.

The problem of high computational cost has been overcome by utilizing machine learning [21]–[23] and statistical [24]–[31] methods that do not require simulating LLPS. However, these previous machine learning and statistical methods have ignored

the conditions under which LLPS occurs, which is observed to be an important factor of LLPS [16]. There are multiple conditions that affect LLPS, such as temperature, salt concentration and protein concentration. Moreover, the complexity of the machine learning models has been limited due to the small amount of experimental data available for training such models.

Recently, more complex machine learning methods have been applied to predict other properties of proteins [32]–[35]. Some of these models have used a deep learning method called graph neural network (GNN) [32], [33] that also utilizes information on the structure of the protein. Although these more complex machine learning methods have been used to study protein functions, none of them has been applied to predicting LLPS. As mentioned earlier, the main reason why these more complex models have not been used to predict LLPS is that they require significantly more training data. Most of the databases that contain experimental data either lack information on the experimental conditions or only contain a few hundred data points.

In order to predict LLPS accurately at different conditions, a more complex machine learning method should be applied to the problem. Therefore, the aim of this thesis is to develop a machine learning model that predicts LLPS of proteins under different conditions and to evaluate its performance using simulated LLPS data. In order to predict LLPS accurately under different conditions, the thesis will develop a GNN model and incorporate the conditions into the model. Due to the lack of available data, the model will be trained using simulated data. To evaluate the model’s performance, the results will be compared to experimental data.

The phase separation of proteins can be affected by the presence of other molecules such as other proteins or RNA. Therefore, the phase separating proteins are sometimes classified as scaffold and client proteins in some databases [36]. This thesis focuses only on LLPS which occurs without the presence of other molecules except for the solvent. Moreover, the training data will be limited to small proteins, with a chain length of less than 60 amino acids, since the computational costs of simulations is lower for smaller proteins. In addition, the considered conditions are limited to the temperature and salt concentration. Thus, other conditions such as the pH, protein concentration and pressure will not be considered.

The remainder of this thesis is divided into six chapters. Chapter 2 reviews first the literature on the existing LLPS prediction methods and then the use of GNNs in protein property prediction. Chapter 3 is divided into two sections: Section 3.1 introduces some of the most well-known LLPS databases and discusses their shortcomings, and Section 3.2 explains how the simulated data is acquired and how the proteins were chosen for the simulations. Chapter 4 presents the theory behind the concepts used in the machine learning model. These concepts include calculating the adjacency matrix, determining the three-dimensional structure of the proteins, the key machine learning methods, and the activation maps for LLPS predictions. Chapter 5 describes the architecture of the model developed in this thesis and the alternatives on how the external conditions can be incorporated into the model. Chapter 6 presents and discusses the results of the thesis. This chapter is divided into four sections. Section 6.1 evaluates different model variations. Section 6.2 presents

the results of the hyperparameter optimization and the optimal hyperparameters for the model. Sections 6.3 and 6.4 assess the performance of the model against the simulated data that is used for training and validation as well as the experimental data. Chapter 7 concludes the thesis by discussing the accuracy of the developed model, the limitations of the model, and suggesting possible improvements for the model.

2 Prediction of protein properties

As this thesis aims to develop a machine learning model to predict the LLPS behaviour of proteins, it is necessary to review the methods that have previously been used in the literature for such a task. Furthermore, predicting the LLPS is closely related to the problem of predicting other properties of proteins since any such problem requires the method to extract meaningful information from the amino acid sequence or the structure of the protein. Therefore, it is important to present how other protein functions are predicted using GNNs, which is the main component of the machine learning model developed in this thesis. Section 2.1 provides an overview of the previous LLPS prediction methods and discusses their weaknesses. Section 2.2 presents the use of GNNs in protein property prediction applications and discusses how these methods have succeeded in predicting LLPS.

2.1 Liquid–liquid phase separation prediction

Multiple methods have been used to predict LLPS of proteins. Some of these have relied purely on statistical methods [24]–[31] and some have used more sophisticated machine learning methods [21]–[23]. The methods published before 2019 are sometimes referred to as first-generation predictors. These first-generation methods are only briefly presented since this thesis focuses on the more recent and complex methods. In addition, the specific techniques that these predictors use are not discussed in detail since the range of methods is extremely diverse. These first-generation predictors were compared in detail in [37]. Some of these first-generation predictors were not intended to be used for phase separation prediction, but they later proved to be able to handle that task to a certain extent.

The first-generation predictors comprised of six amino acid sequence-based methods: PLAAC, LARKS, R+Y, DDX4-like, CatGranule and PScore. The prion-like amino acid composition (PLAAC) [24] was the first method that could predict the LLPS to some extent even though it was not originally intended to predict LLPS. PLAAC used Hidden Markov Model to predict prion-like domains in proteins. The training data that was used to train the model contained originally only 4 prion proteins, but this was later increased to 28 [25]. In the second first-generation predictor, low-complexity aromatic-rich kinked segments (LARKS) [26] were observed to be similar to the prion-like domains also used in the PLAAC. In LARKS the proteins of interest were compared to template proteins that were known to have specific low-complexity aromatic-rich kinked segments. In the third predictor, R+Y, the relationship between the concentration at which phase separation occurs and the number of arginine and tyrosine residues were used to predict the phase separation [27]. This observation of the correlation was only done by using three proteins. However, the method was still able to estimate the phase separation of a large portion of the human proteome. The fourth predictor, DDX4-like, was similar to the third method. However, instead of using the number of arginine and tyrosine residues it utilized the frequency of arginine and phenylalanine residues [28]. It also used a different family of proteins for training than R+Y. The fifth predictor, called

CatGRANULE, employed the structural disorder, nucleic acid binding properties and the arginine, glycine and phenylalanine contents to predict the LLPS [29]. The last first-generation predictor PScore, predicted the average pi-contact frequencies that are known to play role in the structural disorder of proteins [30]. The disorder of protein is closely related to the LLPS behaviour, as mentioned in Chapter 1.

As discussed in the comparison of the first-generation phase separation predictors in [37], it is possible to predict phase separation only from a limited number of sequence features. According to this comparison, the predictors had similar predictive abilities. Therefore, the comparison suggests that LLPS depends on a wide range of different forces since the different predictors were based on different protein features. Many of these methods were also trained on a small set of proteins. In particular, the positive test set that experiences phase separation behaviour was only a few proteins for some of the predictors. Therefore, the first-generation methods might not capture the phase separation behaviour of proteins that exhibit phase separation due to other properties of the protein. However, some of the methods were also validated against proteins from different proteomes, suggesting that some of the methods might perform well on data that is not similar to the training data. However, it has not been shown that any of these methods would work for a large and diverse set of proteins. Furthermore, as mentioned, these methods only rely on a few features and cannot represent all aspects of the LLPS phenomenon which depends on a wide range of possibly unknown features. Identifying more features that could be used to predict LLPS would be difficult and require a lot of expertise in the underlying phenomenon. In addition, using more features might help the model to only identify LLPS for some specific set of proteins for which the LLPS is affected by these new features. Thus, predicting LLPS would benefit from more complex models that would use diverse training data.

Since the first generation phase separation predictors, some attempts have been made to apply a more general and accurate prediction method to LLPS prediction. One of the first attempts was the PSPer [31]. PSPer was specifically developed to predict the phase separation of prion-like proteins. Prion proteins can fold into a structure that is different from their usual structure and can cause nearby proteins to also take the misfolded structure. In PSPer, four characteristics that phase separating proteins share were first identified. The domains in a protein that were linked to these characteristics were prion-like domains, disordered domains, RNA recognition motifs and unspecified domains. These characteristics were identified by observing a set of proteins called the FUS family. They were used in a Hidden Markov model that produces a score for the possibility of phase separation based on whether these domains are present in the protein. This model was only trained with 22 proteins that belong to a family of proteins called the FUS family. Even though the FUS family is considered to be diverse [31], it still presents a small set of training data to cover all the different kinds of phase-separating proteins. In addition, as with the first-generation methods, the PSPer relied on some specific features of the data and moreover, the four characteristics may not capture all possible domains that impact the phase separation.

The first LLPS predictor that utilized supervised machine learning was PSPredic-

tor [21]. PSPredictor was based on some well-known and common machine learning methods. To find the best combination of methods, PSPredictor was tested using two different protein encoding methods and seven different machine learning algorithms. The best combination that was found was to use word2vec [38] and Gradient Boosting Decision Tree (GBDT) [39]. Word2vec is a natural language processing method that uses a neural network to learn word embeddings and to code the protein amino acid sequence as a vector of numbers. This approach resulted in a more general prediction method than the previous ones that only used the amino acid sequence for prediction instead of incorporating features into the model. Compared to the earlier prediction methods, PSPredictor used a significantly larger positive training data set, i.e. training data that contains proteins that are known to phase separate. The authors acquired 353 protein sequences, from a database called LLPSDB [40], that were used as the positive training dataset for the final model. The negative training data was obtained from PDB [41] by selecting proteins that were unlikely to phase separate. PSPredictor was also compared to the first-generation phase separation predictors and it achieved better results than them.

The word2vec protein embedding method was also used by deePhase [22], and like PSPredictor it also used LLPSDB [40] and PDB [41] to build the data sets. However, instead of positive and negative datasets, the data was divided into three parts. The first two datasets were sequences that had respectively, a high and low propensity to phase separate, and the third one was comprised of the sequences that are unlikely to phase separate. The authors constructed two models: one used eight physical features such as the fraction of the sequence to be a part of an IDR, and the other used a 200-dimensional embedding vector generated by the word2vec. Both models were trained with data that had a high propensity to phase separate and data that was unlikely to phase separate as well as with data that included proteins with both high and low propensities to phase separate. The latter data was used so that the model could discriminate between more similar datasets. All models used a random forest classifier to assess whether the protein is identified to phase separate. The average of the predictions of these four models was used as the final prediction.

The third method used to predict LLPS is PSAP [23]. Despite the fact that earlier LLPS predictors had already used the LLPSDB, which contains several hundreds of phase-separating proteins, for PSAP the earlier research on LLPS was manually curated to find proteins that might exhibit LLPS. This resulted in 90 human proteins that have a high probability of phase-separating. These proteins were compared to the rest of the human proteome. From this comparison, amino acid sequence-related features that can be used to determine proteins that phase-separate were identified. It was concluded that phase-separating proteins are typically enriched by some specific amino acids and have a depletion of some other amino acids. In addition, differences in the low complexity regions (LCR) and differences in the amino acid composition of these regions were identified when the phase-separating proteins were compared to other proteins. Using these analyses of the amino acid sequences, a machine learning method was created, using a total of 55 amino acid sequence-specific features. Based on these features, a random forest classifier that was able to determine the phase separation likelihood of protein was developed.

In this section, several protein LLPS predictors were presented. All of them suffered from two common problems: the models are simple due to the lack of training data and they do not consider the dependence of LLPS on the external conditions. In the next section, protein property prediction with more complex methods, namely graph neural networks, is presented.

2.2 Graph neural networks in protein property prediction

The review of the LLPS prediction methods in Section 2.1 demonstrated that the amount of available data of proteins that are known to phase separate is limited. Therefore, some of the prediction models can only use a few hundred protein sequences that are known to phase-separate. This has led to machine learning models using simple methods in predicting LLPS. Due to this simplicity, the capability to predict LLPS is also limited. However, for other functions of proteins, there has been more data available or it has been easier to create. Therefore, the variety of machine learning methods that have been applied to other protein functions has been wider. One of these methods is the Graph Neural Network (GNN). GNNs are a set of neural network methods that work in the graph domain. There are many variants of the GNNs such as the graph convolutional network (GCN) [42], graph attention network (GAT) [43], graph recurrent network (GRN) and gated graph neural network (GGNN) [44]. Graph neural networks have been applied to a variety of different protein function predictions [32], [33], [45]. In addition, the GCNs have proved to work well for extracting molecule properties [46]. This section presents previous work where GNNs have been used to predict protein properties or functions. The background theory of GCNs is reviewed in Section 4.4.

One application of GCNs is to predict drug-target interaction, i.e. the interaction between proteins and ligands [45]. This application of drug-target interaction requires two adjacency matrices, which can be used to represent the structure of a protein in a two-dimensional form. Adjacency matrices are introduced in more detail in Section 4.1. The first adjacency matrix only contains interactions inside a single molecule decoded as ones and zeros. The second adjacency matrix also represents intermolecular interactions. An interaction within a single molecule is indicated by 1, and if the distance between atoms of different molecules is less than a predefined limit, the distance determines the magnitude of the number in the adjacency matrix. Thus, the closer two atoms are, the larger numbers represent their interactions in the adjacency matrix. The chosen variant of GNN in this application was the graph attention network (GAT), where two adjacency matrices are used as separate inputs for the GAT with the embedding vectors. The node features of these two graphs are subtracted from each other and the resulting vector is then summed up to acquire one number which classifies whether the interaction is active in the drug-target interaction. The data used in [45] contained only approximately 100 proteins. However, the total number of data points from these proteins was significantly larger, with over 15000 data points. This large number of points was achieved by performing simulations to produce data with different orientations and positions of the protein and the ligand. The model performed extremely well within the used data set. However, this good

performance was limited to the considered data set, and the model did not generalize well.

DeepGraphGO [33] is another deep learning model for protein function prediction based on graph neural networks. The model was trained to predict the Gene Ontology (GO) terms for multiple species. The GO terms consist of thousands of terms that describe the molecular actions of proteins. As with other applications that used GNNs for proteins, in DeepGraphGo the GNN used an adjacency matrix and embedding vectors as inputs. Instead of the more commonly used binary adjacency matrix, DeepGraphGO used an adjacency matrix where the elements were in a range between zero and one. The elements were scaled so that larger numbers corresponded to particles that were closer to each other. The embedding vectors were binary feature vectors generated separately from the rest of the model. The GNN method that was selected for the DeepGraphGo was the GCN. The data for the training of the model contained over 100 000 proteins from multiple different species. The GO terms are a common and well-known way to describe functions of proteins. Therefore, there are large databases available for GO terms. DeepGraphGO outperformed all the other GO term prediction methods that it was compared against.

One recent application of GCN on protein function prediction is DeepFRI [32]. DeepFRI was designed to predict the Gene Ontology (GO) terms and Enzyme Commission (EC) numbers of proteins. The architecture of the DeepFRI model consisted of a long short-term memory (LSTM) language model and multiple GCN layers followed by a pooling layer and a mapping so that the output has two activations for each function. These activations are then transformed to positive and negative probabilities using the softmax function. The LSTM language model was trained separately from the rest of the model. The theory behind the LSTM language models is presented in Section 4.3. The LSTM language model was trained using approximately 10 million protein sequences. The training of the LSTM model does not require data on the protein functions or structures. The LSTM model is only used to produce a meaningful numerical representation of the amino acid sequence of a protein, that is, the LSTM is trained to predict the next amino acid in the sequence using the other amino acids as information. The hidden representations inside the model are used as the output of the LSTM language model. These are in turn used as the embedding vectors that are passed as the input for the GCN. With the GCN, the model is able to learn features of residues that are far from each other in the amino acid sequence but close in the three-dimensional structure. The use of a three-dimensional structure for DeepFRI was possible due to the larger amount of data available for the GO terms and EC numbers. The training data for the model was gathered from PDB [41] and SWISS-MODEL [47], and the GO terms and EC numbers were identified using SIFTS [48]. The data was heavily filtered but it still resulted in over 30 000 proteins in the training set. This is a large training data set compared to the LLPS prediction methods presented in Section 2.1. The key to acquiring the data was SIFTS [48]. SIFTS connects the databases so that the structure data can be cross-referenced to the function data between different databases. The DeepFRI model outperformed two other state-of-the-art methods when the same training data was used for all of them. DeepFRI was also used with

class activation maps to see which parts of the protein contributed the most to the predictions. This can give information on what kind of parts a protein should include to have a specific function.

Based on these three applications of predicting protein properties using GNNs, it seems that the GNNs perform well in extracting information from the three-dimensional structure of a protein. Information on the three-dimensional structure of proteins had not been utilized by the earlier LLPS prediction methods presented in Section 2.1. Compared to the methods that were used to predict the LLPS, the GNN models have considerably more training data available in easily accessible public databases, especially for the GO term predictors. The results of the two methods that used GCN were especially promising, and the DeepFRI model could be used to predict other protein properties, such as LLPS, with only small modifications to the model. Therefore, the model developed in this thesis will have similar components as the DeepFRI model. Using the DeepFRI model for a different property naturally requires a lot of data that includes measurements for the property or function that is predicted. Since the GNN methods rely on the three-dimensional structure of the protein, such structural information must also be included in the data. Fortunately, there are databases for protein structures [41] and if there is no experimental data on the structure, it can be predicted with great accuracy [49]. Determining three-dimensional structures of proteins is discussed further in Section 4.2.

3 LLPS data

There are multiple public databases that contain information on the LLPS phenomenon of proteins. However, each of these has some drawbacks that prevent their use as the only training data for methods that require a lot of data such as GNNs. As discussed in Section 1, the LLPS depends on external conditions. The LLPS measurements are usually performed for proteins that are in a saltwater mixture. This thesis considers two external conditions that are important for the LLPS: the temperature at which the experiment is conducted, and the salt concentration of the water. The one common shortcoming in all of the public databases is that phase separation is considered to be a binary variable, meaning that the protein either has formed condensates or not. In reality, depending on the protein and the external conditions, the level of phase separation can vary. Higher levels of LLPS will have larger condensates. The differences in the LLPS level are discussed later in Section 3.1. In this chapter, the most promising public LLPS databases are introduced as well as some of their shortcomings. After that, recent experimental data and the method that was used to acquire the experimental data are presented in Section 3.1. The simulated data and the simulation framework are presented in Section 3.2. The simulated data is based on unpublished simulations carried out at VTT Technical Research Centre of Finland.

The largest database of experimental data is the LLPSDB [40]. The LLPSDB contains approximately 1800 data points, and it has been used in many models that have predicted LLPS [21], [22]. However, some of the data points correspond to systems with multiple different proteins or proteins and RNA. This thesis only focuses on the LLPS of a single protein in a saltwater mixture. Therefore, data points with different proteins or RNA cannot be used. The LLPSDB records a total of eight external conditions for the experiments but for most of the experiments, many of these conditions are missing. This renders the use of these data points meaningless since the experiment conditions may have been such that they would have significant effect on the phase separation behaviour of the protein. Therefore, using a replacement value such as the mean of the condition in question for the missing values does not solve this problem. In addition, using only three external conditions out of the eight possible ones can result in data points that are similar for the considered three conditions, but one of the data points is positive for LLPS and another one is negative due to the difference in the other five external conditions. This sort of inconsistency for the data points is not acceptable. Filtering out the data points that are unreliable for one of the above reasons results in a set of only a few hundred data points. For a data-intensive deep learning method, this is not enough.

Another available database is PhaSepDB [50] that contains approximately 500 proteins that are able to undergo LLPS without other molecules. However, the entries in PhaSepDB do not contain the external conditions at which the LLPS happens. Therefore, using this data is problematic since the aim of this thesis is to develop a model that could also predict the conditions under which the LLPS happens. PhaSePro [51] and DrLLPS [36] databases also suffer from this same problem of

not including the external conditions for the experiments. PhaseSePro also only contains 121 entries which is not enough data. DrLLPS has over 400 000 entries, but these entries are only LLPS-associated proteins that might not undergo LLPS without the presence of other proteins. In addition, DrLLPS contains computationally detected LLPS-related proteins. For the proteins, DrLLPS provides various different annotations that are gathered from different sources. Many proteins contain only some of the possible annotations, and using this database would therefore be difficult. Based on these observations on these well-known LLPS databases, it is clear that they alone are not sufficient for a data-intensive machine learning model that also accounts for the external conditions of the LLPS. More thorough comparison of the databases is available in [52].

3.1 Experimental data

Producing experimental data on LLPS is expensive and time-consuming. Therefore, creating enough experimental data for the training of the model is not feasible. However, a small amount of data can be used to validate the model. A small amount of experimental data was produced at VTT Technical Research Centre of Finland in a project called AIMS [53]. One of the goals of the AIMS project was to create a methodology that enables accelerated engineering of elastin-like polypeptides. The elastin-like polypeptides contain intrinsically disordered regions. In AIMS, the phase behaviour of ten proteins was determined experimentally. The experiments were conducted for different protein concentrations, salt concentrations and temperatures. The salt concentration had six, protein concentration 12 and temperature 14 different values. The protein concentration varied from zero to 0.05mg/ml, the salt concentration from zero to 2M, and the temperature from 20 to 95°C. The contour plots as a function of the salt and protein concentration for one of the proteins at 50°C temperature can be seen in Figure 1. The colour scale in the figure shows the absorbance of the sample. The phase separation behaviour of these ten proteins is still not enough for a data-intensive machine learning model. The number of different kinds of proteins is extremely high and the set of only ten proteins cannot capture the space of all proteins. Therefore, this data cannot be used for training the machine learning model. However, this data can be used to validate the machine learning model by comparing the prediction of the developed model to these experimentally determined phase behaviours.

To experimentally detect the phase separation, the turbidity of the samples was measured [54]. This is a common method to analyze the increase in the optical density of a sample. In turbidity measurement, a beam of light is directed through the sample and the intensity of the light is then measured after it has propagated through the sample. The transmitted light is inversely proportional to the number of assemblies in the solution. It is assumed that the assemblies are so small that they have nearly identical absorbance. The absorbance of a sample is defined as

$$A = \log(I_0/I), \quad (1)$$

where I_0 is the intensity of the incident light and I is the intensity of the transmitted light. The assemblies are assumed to have nearly identical absorbance regardless of the size of the assembly. Thus, absorbance describes the number of assemblies in the sample. Higher absorbance indicates that there are fewer assemblies in the solution which means that on average the assemblies have more proteins in them. Furthermore, more proteins in the assemblies indicates that the proteins have phase separated. Therefore, the absorbance can be used as a measure of LLPS. The experimentally measure absorbance can not be directly linked to the simulated data. As a result, only the trends of the absorbance are compared to the predictions of the machine learning models.

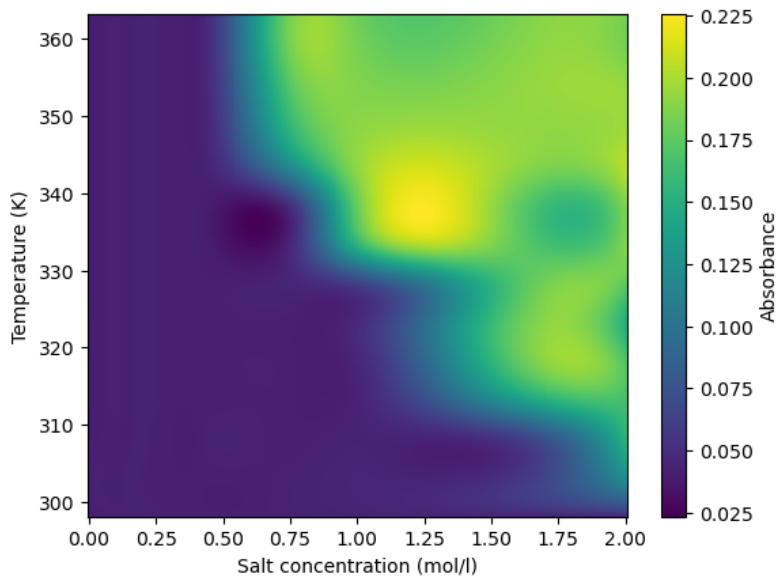


Figure 1: Experimentally determined absorbance with cubic interpolation between the data points.

3.2 Simulated data

The experimental data and the data available in public databases are not enough for a data-intensive machine learning model such as the graph neural network. Therefore, there is a need to acquire more data. Experimentally producing enough data for the model is not feasible since experimentally determining the phase behaviour of protein is time-consuming and expensive. Thus, a database of simulated LLPS data is used as the training data. The database was created for the training of the model developed in this thesis and consists of unpublished simulations conducted at VTT Technical Research Centre of Finland. The simulations and data generation are not the focus of the thesis. Therefore, the database and the simulation framework are only briefly presented. Using simulated data ensures that the amount of data is sufficient. In addition, controlling the conditions is easier than in the experimental setup.

There exist multiple methods for simulating the phase separation behaviour of

proteins [17]. These methods can be classified by the resolution of the simulation. Methods with higher resolutions give more accurate results but require more computational resources. Simulating the LLPS of one protein requires hundreds of these proteins to be simulated at once because to observe LLPS, a large number of proteins is needed to observe the more concentrated and less concentrated phases. In addition, the simulations need to be conducted for hundreds of proteins with tens of conditions for each protein. Therefore, atomistic simulations are not an option due to the high computational cost. The highest resolution with still reasonable computational cost is the CG simulation method. In CG models, a group of atoms is modelled as a coarse-grained bead. There are different levels as to how many atoms are included in a single bead. For proteins, it is natural to model single amino acid as a single bead. This level of coarse-grained resolution has been used successfully to study the phase separation behaviour of proteins, and it is the most detailed level that can practically be used to simulate LLPS [17].

As mentioned earlier, the data used in this thesis is from unpublished simulations performed at VTT. These simulations were conducted using GROMACS [55] which is a software for molecular dynamics that is well suited for simulating proteins. GROMACS can be used with different force fields. The force field for the simulations used in this thesis was the Martini force field [56]. It is a coarse-grain force field that was initially developed to model lipids. However, it has been updated to be able to model proteins [57] and many other biomolecular systems. In these simulations, the latest version called Martini 3 [58] was used. Instead of modelling each amino acid with only a single bead, Martini force field uses a four-to-one mapping for amino acids. Four-to-one mapping means that on average one bead represents four heavy atoms. The exception to this is the amino acids that have ringlike structures. These ringlike structures use less than four heavy atoms per bead. Thus each amino acid is represented with one to five beads. One of the beads represents the backbone of the protein chain, and the rest are used to represent the side chain. More in depth explanation of the Martini model and the model parameters can be found in [57].

The simulations followed a similar approach as a method called slab method [18] which has been shown to be able to model the LLPS. In the slab method, the initial set-up of the simulation is a high-density protein slab. The high-density protein slab is generated by conducting a 100ns simulation that starts from a dispersed state. This dispersed state is then compressed to a high-density slab to achieve the high-density protein slab. In the actual simulation, the simulation box is opened on one side allowing the proteins to reach equilibrium with the low-density phase. The density profile of the system can then be used to determine the phase diagram of the protein. From the phase diagram, the critical temperature of a protein can be inferred and used as an indicator of the phase separation. In the initial configuration of the simulations used in this thesis, the proteins are in a dense-phase as in the slab method. However, instead of opening the simulation box from one side, the box is opened from all sides.

The use of the Martini force field allows the system to be simulated for a relatively long time compared to atomistic molecular dynamics simulations. In the simulations that are used in this thesis, the system is simulated for $0,1\mu\text{s}$. To acquire a stable

initial configuration for the system, an equilibration step is performed for the model. The maximum number of proteins in the system is 50. The exact number of proteins can vary slightly depending on how many proteins GROMACS is able to fit in the simulation box. The rest of the simulation box is filled with water. An example of the initial configuration of a simulation is presented in Figure 2. In this figure, the beads that represent water molecules are removed and each protein chain is colored with a different colour to visualize the distinct protein chains. The green lines represent the edges of the simulation box. Instead of a cube, the simulation box is initially a dodecahedron, but due to the periodic boundary conditions, it becomes a parallelepiped. Periodic boundary conditions are used to avoid any problems with the boundaries of the simulation box. It also allows the system to look more like an infinite system. Thus, for a large set of proteins, some proteins may leave the simulation box from one side, only to appear on the opposite side. As can be seen from Figure 2, the proteins form a fairly dense clump at the start of the simulation which is a desired initial configuration of the system. Starting from this initial configuration, the proteins can interact with each other and form higher or lower-density phases depending on the phase separation properties of the protein.

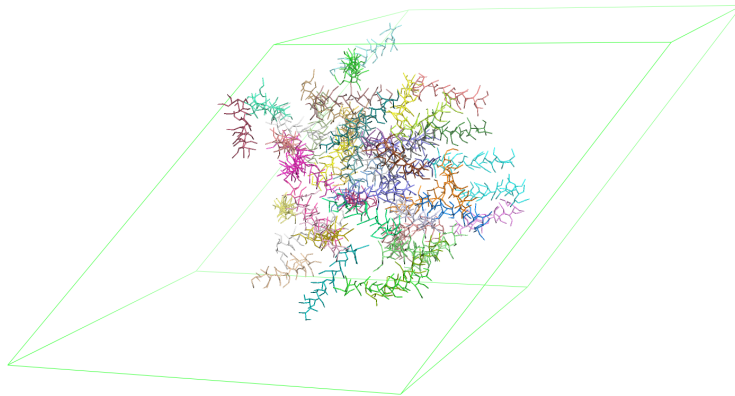


Figure 2: Snapshot of the simulation of one of the proteins at $t = 0$. Each protein chain is colored with different color and water is not displayed.

With the slab method, inferring the phase separation from the simulations was done by analysing the density profile along one of the axes which was elongated at the beginning of the simulation. However, this is only one alternative used in the literature. In the simulations of this thesis, the simulation box is elongated in all directions. Therefore, the same approach is not applicable to these simulations. To assess the phase separation behaviour of the proteins in the simulations, the interest is whether the protein chains form an even more dense cluster or if they spread

out around the simulation box. This can be achieved by manually viewing each simulation. However, with hundreds of simulations, such an approach is not feasible.

In this thesis, determining whether the protein chains form a high-density cluster or spread was solved by calculating the radius of gyration [59] for the proteins. Usually, radius of gyration is used to study the compactness of a single protein. However, it can also be used for multiple proteins and is defined as

$$R_{gyr}^2 = \frac{1}{M} \sum_{i=1}^N m_i (r_i - R)^2, \quad (2)$$

where M is the total mass of the particles $\sum_{i=1}^N m_i = M$ and R is the centre of mass of the particles. As a result, the radius of gyration describes how far the mass of the particles is spread from their centre of mass. Therefore, it can be used as an indicator of the LLPS because it describes how densely the proteins are clustered. As mentioned earlier, the simulations used in this thesis utilized a coarse-grained model. Thus, amino acids can be represented by different amounts of coarse-grained beads. However, to measure the compactness of the proteins so that each amino acid contributes equally to the radius of gyration, only the backbone bead is used for each amino acid. Thus, for the calculation of the radius of gyration, each amino acid is represented by a single bead. A visual example of the initial and final configuration of the proteins for two different proteins with different radius of gyrations at the end of the simulations is shown in Figure 3. The first protein in Figures 3a and 3b has a radius of gyration of approximately 60.0Å at the end, which is considered to be low, and the protein in Figures 3c and 3d has a high radius of gyration compared to the other protein which is approximately 95.6Å. As can be seen from the figures, the simulation leading to a high radius of gyration is considerably more spread out at the end of the simulation. Thus, the radius of gyration seems to perform well as an indicator of the LLPS for the simulation data used in this thesis.

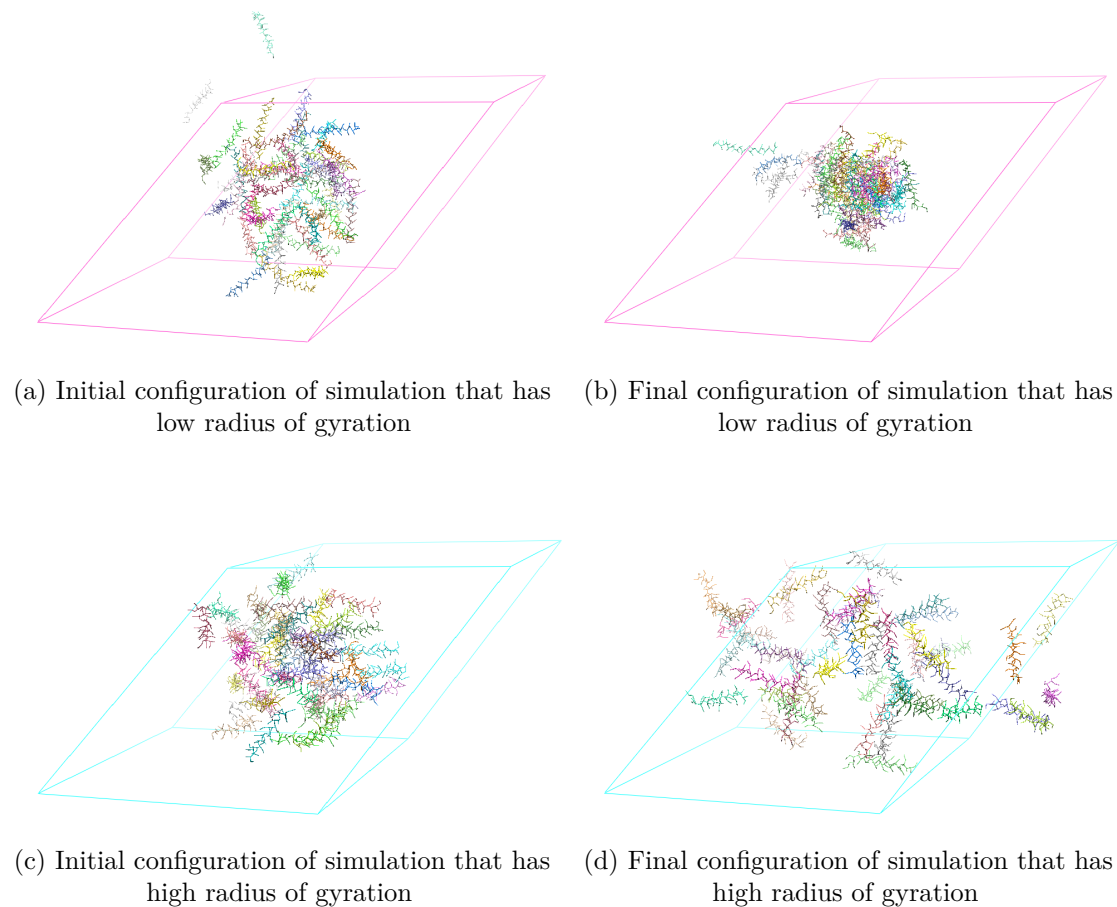


Figure 3: Initial and final configurations of two simulations that have a different radius of gyration at the end.

Since the model is trained using simulated data, the proteins used should be chosen so that they are useful for the model and cover a wide range of different proteins. Therefore, it is reasonable to use proteins that are somewhat similar to the ones that are considered to be interesting when using the model for prediction. The model is more likely to produce high-quality predictions if it has been trained on similar data that it has seen during the training. In addition, using a wide range of proteins might allow the model to generalise better to unseen data. The majority of data used for training the machine learning model comes from the simulations done for the proteins that were used in the AIMS [53] project discussed in Section 3.1. These proteins were chosen because their phase separation behaviour was also an interest for the AIMS project, which introduces an obvious synergy. This set contains 133 proteins, each simulated with approximately six different temperatures and five salt concentrations. All the simulated proteins have fairly short amino acid chains, with the number of amino acids ranging from 20 to 60 amino acids. The protein set is not particularly diverse but has proteins that are potentially interesting in the prediction phase. To achieve a more generalised model this data set can be extended by simulating more diverse proteins.

4 Theory of the model components

The Machine learning model developed in this thesis is comprised of multiple smaller components. In this chapter, the theory of the most important components is presented. This includes the preprocessing steps for the data. The first two sections present the preparation of data for the machine learning model. Section 4.1 explains how the adjacency matrices can be constructed from the structure of the protein and Section 4.2 introduces methods to acquire the structure for a protein. Sections 4.3 and 4.4 present the two major components of the machine learning model. The structure of the model and how these components are organized is presented in the next chapter.

4.1 Adjacency matrix

Most variants of GNNs use an adjacency matrix as the input along with embedding vectors. The adjacency matrix A for protein is a symmetric $L \times L$ matrix, where L is the number of amino acids in a protein. Each entry of the matrix describes the distance between two amino acids. The adjacency matrix can be either weighted or unweighted. If the adjacency matrix is unweighted, a cut-off value has to be chosen for the distance. Amino acids closer to each other than the cut-off value are then considered to be in contact and have the value of one, and amino acid pairs that are not connected will have the value of zero. If the adjacency matrix is weighted, the element corresponding to an amino acid pair is not necessarily binary but can represent the distance between the two amino acids. One possible way to assign elements to a weighted adjacency matrix is to use the reciprocal of the distance between the two amino acids. Then the closer the amino acids are to each other, the larger the value they have in the adjacency matrix. As with the unweighted adjacency matrix, some cut-off value can be chosen so that amino acids far from each other will be assigned a value of zero, which makes the adjacency matrix sparser.

The distance between two amino acids can be defined in multiple ways, such as the distance of the corresponding C_α atoms of the amino acids, the distance of any two atoms of the amino acids that are the closest, or the distance of the so-called Rosetta neighbour atoms [32]. The two latter options are better in capturing the distance between the side chains because the C_α does not provide any information on which direction the side chains are directed and how close the side chains are.

C_α is a carbon atom at the backbone of the protein, and it is located between the amino and carboxyl groups. For most amino acids, the side chain is also connected to the C_α atom. The distance between two consecutive C_α atoms is approximately 3.8Å [60]. Therefore, the cut-off value should be larger than this to capture at least the connections between amino acids that are next to each other in the primary sequence, if the distance between the corresponding C_α atoms is used as the distance measure. The DeepFRI model [32] used the distance between the C_α atoms as the distance between two amino acids and a cut-off value of 6.5Å. In this thesis, two different cut-off values are tested in order to determine which performs the best in the considered application. The first cut-off value is the same as in [32]. The second

is 10.0Å. The use of larger cut-off value may be able to capture longer interaction in proteins.

4.2 Determining the three-dimensional structure

The adjacency matrices that are used in most of the GNN variants are determined from the three-dimensional structure of the protein. For many proteins, the experimentally verified structures can be found in the Protein Data Bank (PDB) database [41]. This database contains protein structures from many natural, engineered, and synthetic source organisms. For example, the database contains approximately 60 000 proteins from the human proteome. The whole database contains approximately 190 000 protein structures. However, this is only a tiny portion of the earth's proteome that is estimated to contain 5 million distinct proteins [61]. Thus, the three-dimensional structure is not experimentally verified even for most of the naturally occurring proteins. In addition, the number of proteins that can be synthetically produced is many orders of magnitude larger than the number of naturally occurring ones.

The PDB contains 190 000 proteins which in many cases is a sufficient number of protein structures. However, when discovering novel proteins with desired properties, the predictions of the model should not be limited to only those proteins that have the experimentally verified structure. Especially if the novel protein candidates that are screened for the LLPS behaviour are not naturally occurring, it is likely that they do not have experimentally verified structures. Therefore, there is a need to use computational methods that can predict the three-dimensional structure of every protein. The protein structure prediction is in itself a very difficult problem and has been a grand challenge in synthetic biology [62].

The leading method in protein structure prediction is the AlphaFold [49] developed by Google Deepmind. In Critical Assessment of protein Structure Prediction (CASP14) [63], which is a competition for methods of protein structure modelling, AlphaFold reached accuracy that clearly outperformed all other methods [49]. However, to achieve this level of accuracy, the model of AlphaFold is complex. One component of the model is multiple sequence alignment (MSA) where the predicted protein is compared to proteins with similar amino acid sequences which is computationally demanding. Therefore, predicting a single structure will take hours even for small proteins with a few dozen amino acids and a moderately powerful computer. Thus, for high-throughput applications, predicting the structure by using AlphaFold might not be the best alternative.

However, there is no need to run the AlphaFold for most of the proteins since AlphaFold predictions have been collected to AlphaFold Protein Structure Database (AlphaFold DB) [64]. This database contains nearly 1 million protein structure predictions. Thus, AlphaFold DB can be used to retrieve the three-dimensional structure of many proteins that are not included in the PDB. New entries are also continuously added to the database. The new proteins are selected from the UniRef90 [65] which contains 100 million proteins.

In case there is a need to screen thousands of novel proteins that are not included in the AlphaFold DB, using AlphaFold to predict the structures might not be feasible.

In this kind of a scenario, some other protein structure predictors such as the AIMS-PROT [53] can predict the three-dimensional structure of the protein considerably faster than AlphaFold. However, this speed comes with the cost of more inaccurate predictions. For secondary structures of small proteins AIMS-PROT was able to give similar predictions as AlphaFold. Therefore, the predictions of AIMS-PROT are considered accurate enough to be used in the creation of the adjacency matrix.

4.3 Recurrent neural networks

In order to use the neural network architecture, described later in this thesis, a method is needed to give each amino acid in the protein an embedding vector. These embedding vectors are numerical representations of the amino acids and should contain useful information about the amino acid and the surrounding amino acids. In its simplest form, this embedding vector can simply be a one-hot encoding of the different amino acids. In one-hot encoding, one of the elements of the embedding vector is set as one and the other ones are set to zero. For example, for the 20 amino acids that are found in the human proteome, an embedding vector of length 20 could be used for each amino acid in the protein. One of the indices in these embedding vectors would correspond to one type of amino acid. The one-hot encoding in itself does not reveal anything about the similarities of the amino acids because each amino acid is assigned its own index in the embedding vector.

Another approach is to use some of the properties or combinations of the properties of the amino acids as the embedding vectors. For example, an embedding vector of length of eight for each amino acid based on a principal component analysis of 50 physiochemical variables has been shown to perform reasonably well [66]. Using these smaller embedding vectors can also reduce the model size of the neural network since the input is smaller. Recently the most popular method has been to use unsupervised machine learning models to compute the embedding vectors [67], [68]. These machine learning methods usually also capture the context around the amino acids, meaning that they also take the other amino acids in the protein into consideration. The neural network that is used to compute the embedding vector in this thesis is a long short-term memory (LSTM) neural network [69]. LSTMs are an extension of a recurrent neural networks (RNN) and can therefore process inputs of varying lengths. However, LSTM does not have the same problems with long dependencies as RNN. The LSTM model used in this thesis is called Prose [70].

The recurrent neural networks to process sequential data were first introduced by Rumelhart et al. [71]. In RNN, the hidden state $h_t \in \mathbb{R}^n$ is updated at each iteration using the current input at element t , $x_t \in \mathbb{R}^m$ and the previous hidden state h_{t-1} . The hidden state can be updated by using different functions, such as the following one, which is a simple multilayer perceptron (MLP)

$$f(x, h) = h_t = \tanh(W h_{t-1} + U x_t + b), \quad (3)$$

where matrices $W \in \mathbb{R}^{n \times n}$ and $U \in \mathbb{R}^{n \times m}$, and vector $b \in \mathbb{R}^n$ are the parameters of the model that are learned during the training. However, with the basic RNN,

learning long-term dependencies have proved to be difficult as was discovered by Bengio et al. [72]. The reasons for this are the vanishing and exploding gradients.

In the simple RNN model presented in Equation (3), the longest backward derivative computation for the loss \mathcal{L} at element t is the derivative with respect to the first hidden state h_1 . The partial derivative of the loss with respect to the first hidden state can be computed using the chain rule for derivatives:

$$\frac{\partial \mathcal{L}}{\partial h_1} = \frac{\partial \mathcal{L}}{\partial h_t} \frac{\partial h_t}{\partial h_{t-1}} \cdots \frac{\partial h_2}{\partial h_1} = \frac{\partial \mathcal{L}}{\partial h_t} \prod_{\tau=2}^t \frac{\partial h_\tau}{\partial h_{\tau-1}}. \quad (4)$$

The term inside the product can be simplified as

$$\frac{\partial h_\tau}{\partial h_{\tau-1}} = \tanh'(Wh_{\tau-1} + Ux_\tau + b)W, \quad (5)$$

where \tanh' denotes the derivative of the tanh function. The derivative of the tanh is bounded between 0 and 1. Therefore, if the spectral radius of W , the maximum of its eigenvalues absolute values, is smaller than one, the gradient will be extremely small for large t . More precisely, for this reason the norm of the gradient vanishes exponentially. Additionally, if the spectral radius of W is large enough the gradient will typically explode even though the derivative of the tanh is bounded. These vanishing and exploding gradients can cause problems during the training of the model.

To avoid these problems, GRU units [73] were introduced. In GRU, in addition to the hidden state, there is an update gate $u \in (0, 1)$ that determines which states are updated. The gate at time t is computed using the input x_t and the previous hidden state h_{t-1} ,

$$u_t = \sigma(W_u h_{t-1} + U_u x_t + b), \quad (6)$$

where σ is the logistic function. The hidden state is updated by using the new candidate state \tilde{h}_t ,

$$h_t = (1 - u_t) \odot h_{t-1} + u_t \odot \tilde{h}_t, \quad (7)$$

where \odot is the element-wise product. Thus, the update gate u_t determines how much of the new hidden state is based on the previous hidden state and how much on the new candidate state. The candidate state is computed as

$$\tilde{h}_t = \phi(W(r_t \odot h_{t-1}) + Ux_t + b_h), \quad (8)$$

$$r_t = \sigma(W_r h_{t-1} + U_r x_t + b_r), \quad (9)$$

where ϕ is some nonlinearity, for example the tanh function. The function r_t is called the reset gate. The reset controls which elements of the previous hidden state are used in the candidate state. The GRU may still have issues with vanishing and exploding gradients, but problems do not occur as often as with the simple RNN model.

The next step in recurrent neural networks after the GRU is the long short-term memory (LSTM) neural network [69]. Instead of using only one internal state the LSTM has two states for each element in the input. These are the hidden state h_t and the cell state c_t . The cell state is updated using the previous cell state c_{t-1} , the current input x_t , forget gate $f_t \in (0, 1)$ and the input gate $i_t \in (0, 1)$,

$$c_t = f_t \odot c_{t-1} + i_t \odot \phi_c(W_c h_{t-1} + U_c x_t + b_c). \quad (10)$$

The forget and the input gate are computed as

$$f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f), \quad (11)$$

$$i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i). \quad (12)$$

The difference between GRU and the update of the cell state of LSTM is that the LSTM uses two gates f_t and i_t instead of the one gate u_t that is used in two coefficients $1 - u_t$ and u_t in GRU. The hidden state is a nonlinear transform of the cell state,

$$h_t = o_t \odot \phi_h(c_t), \quad (13)$$

where the nonlinearity ϕ_h is usually chosen to be the tanh function. There is one additional gate o_t . This is called the output gate and it determines which states are passed to the output. It is computed as

$$o_t = \sigma(W_o h_{t-1} + U_o x_t + b_o). \quad (14)$$

The hidden state is the output of the LSTM cell. Thus, for each input x_t there is a corresponding output h_t . The dimension of the output can be determined by the user. LSTM does not experience the problem of vanishing or exploding gradients that were present with the RNN and GRU networks. It is possible to stack multiple LSTM layers, where the input of the next layer is the hidden state of the previous layer. Multiple layer models perform better in some tasks [74].

4.4 Graph neural network

Many types of data can be naturally presented using graphs. This includes data such as molecules, social networks and connections between web pages. The concept of processing data that can be presented as a graph with neural networks was first

introduced by Gori et al. [75]. The methods that were used on graph data gained popularity after Scarselli et al. [76] continued the work and validated their model with experimental data and demonstrated that it can be generalized for many types of data. The model of Scarselli et al. [76] was the first neural network model to be called graph neural network (GNN). It can be used for many types of graphs such as acyclic, cyclic, directed and undirected. GNNs can be used to predict node-level, edge-level or graph-level properties of a graph.

A graph can be defined as a tuple $G = (N, E)$ where N is the set of nodes and E is the set of edges. Nodes and edges can have features attached to them. These can be represented by vectors and denoted by l_n for node n . For edges, the features are denoted as $l_{(n_1, n_2)}$, where the edge is between nodes n_1 and n_2 . These vectors are referred to as labels in this thesis but in literature, they are sometimes called feature vectors. They usually contain some features of the nodes and the relationships between the nodes.

In the first graph neural network model [76], each node had a state $x_n \in \mathbb{R}^s$ attached to it. The state of a node contains information about the neighbourhood of the node. For this model, a local transition function f_w was defined that was used to compute the state for each node,

$$x_n = f_w(l_n, l_{\text{co}(n)}, x_{\text{ne}(n)}, l_{\text{ne}(n)}), \quad (15)$$

where $l_{\text{co}(n)}$ are the labels of the edges connected to the node, $x_{\text{ne}(n)}$ are the states of the neighbour nodes and $l_{\text{ne}(n)}$ are the labels of the neighbour nodes. This function describes the node's dependence on its neighbourhood. Usually, the neighbourhood is defined as the nodes that are connected to considered node by an edge. However, the neighbourhood can also be defined to include nodes that are not direct neighbours. The neighbourhood can for example be nodes that are either one or two edges away. The output from the node can be described by the local output function

$$o_n = g_w(x_n, l_n). \quad (16)$$

Equations (15) and (16) can be written in more compact form by stacking the states, outputs, labels and node labels to get vectors x , o , l and l_N , respectively,

$$x = F_w(x, l), \quad (17)$$

$$o = G_w(x, l_N), \quad (18)$$

where F_w and G_w are the global variants of the local transition and local output functions that are acquired by stacking all the local functions.

Scarselli et al. [76] listed some properties that the functions F_w and G_w should have. First, the states x and the outputs o should be uniquely defined. Secondly, the functions should define a mapping that takes the graph as input and returns output for all the nodes. The uniqueness and existence of the solution is guaranteed if F_w is a contraction mapping, that is, if there exists some real number $0 \leq k \leq 1$ such that

$$\|F_w(x, l) - F_w(y, l)\| < k\|x - y\| \quad (19)$$

for any x and y , where $\|\cdot\|$ is a vector norm.

If F_w is a contraction mapping, the existence and uniqueness of a solution are guaranteed by Banach's fixed point theorem [77]. The fixed point theorem also states that the solution can be computed iteratively

$$x(t+1) = F_w(x(t), l), \quad (20)$$

where $x(t)$ is the t th iteration of x . The solution will converge to the solution regardless of the initial value $x(0)$.

The choice for the global transition and output functions in Equations (17) and (18) depends on the type of the graphs. Graphs can be divided into positional and nonpositional graphs. For positional graphs, each neighbour of a node is assigned a unique integer identifier, which indicates the logical position of the neighbour. Most common graph applications, including graphs that are derived from the adjacency matrix, are nonpositional. Therefore, from here on the theory review considers only nonpositional graphs. For nonpositional graphs, Equation (15) can be expressed with a sum over the neighbours of the node n and using a parametric function h_w ,

$$x_n = \sum_{u \in \text{ne}(n)} h_w(L_n, l_{(n,u)}, x_u, l_u). \quad (21)$$

The local output function g_w does not have any requirements such as the uniqueness of the solution. Therefore, in the first implementation of GNN [76] the local output function is a multi-layer perceptron (MLP). However, as discussed earlier, the global transition function F_w should be a contraction mapping. This requirement on the global transition function restricts the implementation of the local transition function f_w . For the first GNN [76], two neural networks that fulfil this assumption for the nonpositional form of the problem described in Equation (21) were proposed. The first model that was presented was the so-called linear GNN

$$h_w(l_n, l_{(n,u)}, x_u, l_u) = A_{n,u}x_u + b_n, \quad (22)$$

where the matrix $A_{n,u} \in \mathbb{R}^{s \times s}$ and the vector $b_n \in \mathbb{R}^s$ are defined as the parameters of two distinct single layer perceptron neural networks. The second model is the nonlinear GNN. In this model, a three-layered MLP was used as the h_w function. A penalty term was added to the loss function to ensure that the global transition function is a contraction mapping.

There are many more recent variants of the GNN, one of which is the graph convolutional neural network (GCN) [42]. Convolutional neural networks (CNN) are designed to process data that have a specific order such as a two-dimensional

spatial structure. CNNs are the state-of-the-art method in image recognition tasks, but CNNs have also been used on protein structure data [78]. In CNNs, the input is multiplied by a filter that is a set of weights. This filter is used as a sliding window over the whole input. Usually, CNN models utilize multiple filters. The convolutional neural network layers were first used on graph structures by Bruna et al. [79]. GCNs use similar convolutional operations as CNNs on graph structures. Instead of the ordered structure that the input for CNN usually has, the GCN is a generalization where nodes can have a different number of neighbours. GCNs have proved to perform well in analysing features of graphs [80].

For graph convolutional neural networks, the objective is to produce a node-level output of the graph. The graph level features can then be extracted using some pooling methods. GCNs usually contain multiple layers. One layer can be presented as a function of the adjacency matrix A and the matrix of activations of the l th layer H^l ,

$$H^{(l+1)} = f(A, H^{(l)}). \quad (23)$$

The matrix $H^{(l)} \in \mathbb{R}^{N \times c_l}$ denotes the states of each node and it can be interpreted as the feature vectors of length c_l of each node in a stacked form. For the first layer $H^{(0)}$, an initial feature matrix X can be used that includes some features of the nodes, but it does not have to include information on the relationships between the nodes. This initial feature matrix can be the amino acid embeddings produced by the LSTM. The output of the model is the matrix of activations of the last layer $H^{(N_l)}$, where N_l is the number of GCN layers.

One of the simplest examples of the GCN layer in Equation (23) is the following

$$f(A, H^{(l)}) = \sigma(AH^{(l)}W^{(l)}), \quad (24)$$

where σ is any activation function and $W^{(l)}$ is the trainable weight matrix of layer l . However, the simple model presented in Equation (24) has a few problems. First, using the adjacency matrix A only sums the feature matrices of the neighbours since the diagonal elements are zero. Therefore, instead of A , the identity matrix is added to the adjacency matrix: $\hat{A} = A + I$. Another problem is that if the matrix A is not normalized, the scale of the matrix of activations will change. Therefore, a symmetric matrix normalization is performed for the adjacency matrix A . As a result, instead of \hat{A} the symmetric matrix normalization is used $\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}$, where \hat{D} is the node degree matrix of \hat{A} . Thus, the Equation (24) of the simple GCN layer can be altered as

$$f(A, H^{(l)}) = H^{(l+1)} = \sigma(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}). \quad (25)$$

With this formulation, the feature vector of each amino acid is updated by taking a weighted sum of its neighbours and the amino acid itself. By changing the size

of the weight matrix $W^{(l)}$, the size of the next matrix of activations can be altered. The dimension of the weight matrix is $W^{(l)} \in \mathbb{R}^{c_l \times c_{l+1}}$, where c_{l+1} is the desired length of the feature vectors in the next amino acid embedding layer $H^{(l+1)}$. One important property of this model is that the weight matrix does not depend on the size of the input and can therefore be used for input of any size. Equation (25) is the same as proposed by Kipf and Welling [42] for the GCN layer. However, Kipf and Welling derive this GCN layer through the spectral graph convolutions and, Equation (25) is an approximation of that result.

5 Machine learning model

The model developed in this thesis employs a similar architecture as the DeepFRI model [32]. DeepFRI was used to predict the protein functions or the so-called Gene Ontology terms of proteins as well as the enzyme commission number, which is a classification system for enzymes. The model has two parts: the LSTM and the GCN. The LSTM is used to produce the embedding vectors for the amino acids. These embedding vectors are then used as the initial feature matrix which is the input for the GCN. The goal of the created model in this thesis is to predict the LLPS of proteins which is highly dependent on the external conditions as mentioned in Chapter 1. Therefore, the DeepFRI model is not directly applicable since it did not consider external conditions. In this chapter, the two parts of the model are presented as well as the integration of the external conditions to the model. The external conditions are added to the model at two different stages. A schematic overview of the model is presented in Figure 4, which shows that the external conditions are added before and after the GCN. The hyperparameter optimization strategy is presented at the end of this chapter. The performance of the two alternatives and the results of the hyperparameter optimization are presented in Chapter 6.

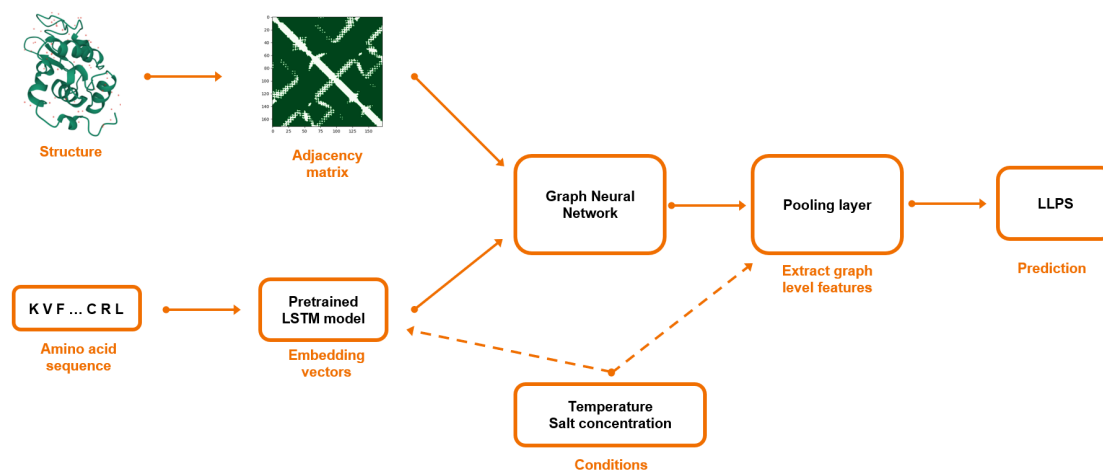


Figure 4: Schematic representation of the model.

5.1 Amino acid embeddings

In order to use the GCN, the amino acids need to be given numerical representation. Amino acid sequences of proteins can be represented as words by denoting each amino acid by its one-letter abbreviation. By representing the protein by a word that consists of the amino acid abbreviations, the protein sequences can be processed using machine learning models that are conventionally used for natural language processing. The DeepFRI [32] used a pre-trained language model [81]. Pre-trained means that the language model is trained separately before training the rest of the model, and when the rest of the model is trained, the parameters of the language

model are fixed. Including pre-trained models has been shown to perform well on biological sequence data [82]. In the DeepFRI language model, the encoder is a stack of LSTM layers [83]. The stack has two layers, and the length of the embedding vector at each layer is 512. A linear layer is used to project the output of the last layer to the embedding space. Using this type of a model means that residues in similar sequences will be close to each other in the embedding space.

In the model developed in this thesis, a newer version of the LSTM considered in the DeepFRI model [81] is used as the language model to compute the residue embeddings [70]. In [70], two language models were developed. The first one uses only the amino acid sequence information, whereas the second model incorporates information on the structure of the smaller set of proteins in addition to the sequence information. The second language model is called multi-task language model (MT-LSTM), and it will be used in this thesis as such. The MT-LSTM is trained to solve three tasks simultaneously, and the total loss function is their weighted sum. The remainder of this section provides a short description on the tasks in the MT-LSTM model of [70].

The first task is the language model where only the sequence information is used. Furthermore, the model tries to predict one amino acid at a time from the amino acid sequence using only the preceding amino acids in the sequence. This model consists of three bidirectional LSTM layers. Bidirectional means that the sequence is processed from both directions using the LSTM. The output of the bidirectional LSTM is acquired by combining the outputs of the forward and backward LSTM layers. In addition, the hidden state, which is used as the embedding between the layers and as the final output of the language model, is a concatenation of the hidden states of the forward and backward layers. The model consists of three of these bidirectional LSTM layers where the input of each layer is the output of the previous one. For the first layer, the one-hot encoded amino acid sequence is used. The final output is the concatenation of the hidden state of each layer,

$$Z = [h_0, h_1, h_2] \in \mathbb{R}^{L \times 3D}, \quad (26)$$

where h_i is the hidden state of layer i of the bidirectional LSTM and D is the dimension of one bidirectional LSTM layer. The dimension of the hidden layers was chosen to be $D = 1024$ in [70]. Therefore, in the initial feature matrix each amino acid is represented by a vector of length 3072. The dimension of the hidden layers could be modified and optimized to suit this particular task of predicting LLPS. However, the training time is long, and therefore the effect of the hidden layer dimension is not studied further. The hidden layers of the language model have been observed to encode meaningful sequence representations for proteins without any additional tasks [81].

The second and third tasks of the MT-LSTM model utilize the structure of the protein in addition to the amino acid sequence. The second task is the prediction of the contacts between the amino acids. The contacts are predicted using a bilinear projection of the output from the first task Z . Thus, the objective is to predict the log-likelihood of contact between amino acids i and j by calculating the bilinear

projection $z_i W z_j + b$, where z_i is the i th row of the Z matrix, and W and b are the learnable parameters of the model. These contacts are then compared to the real contacts of the protein. This is done for all amino acid pairs.

The third task is to compare the structural similarity of proteins. In this task, the embedding of the query sequence Z_1 and the target sequence Z_2 are used to predict the structural similarity. The true structural similarity of the target and query sequence is obtained from the Structural Classification of Proteins [84]. Proteins with similar amino acid sequences usually have similar structures and functions, and they should therefore be close to each other in the embedding space. The pairwise distance of the embeddings $d_{i,j} = \|Z_{1,i} - Z_{2,j}\|$ is used as the measure of the similarity of the embeddings. This task incorporates information into the model about proteins that are not similar in their amino acid sequences but still have a similar spatial structure.

The total loss of the multi-task model is a weighted sum between all the losses of the three tasks of the model,

$$L_{MT} = \lambda_m L_m + \lambda_c L_c + \lambda_s L_s, \quad (27)$$

where λ denotes the weights given to a task, L denotes the loss of a task, and subscripts m , c and s denote the first second and third task respectively. In the Prose model [70], the optimal weights that correctly emphasized each task were determined to be $\lambda_m = 0.5$, $\lambda_c = 0.9$, $\lambda_s = 0.1$. The training dataset of the Prose model contained approximately 76 million sequences from the UniRef90 database [65] that were used for the language model of the first task. For the two structure tasks, SCOPe ASTRAL 2.06 [85] database was used. This database contained approximately 22 000 proteins for training.

As shown in Figure 4, the conditions are added to the model at two different stages. In the first stage, the conditions are added before the GCN and concatenated to each embedding vector. Thus, the dimension of the input for the GCN is $H^{(0)} \in \mathbb{R}^{L \times 3075}$. The conditions are encoded to all the amino acids, and therefore they are used at each layer of the GCN and in the weighted averages over the neighbours. This sort of approach has not been widely used for GCNs.

5.2 Graph convolutional neural network

For the graph convolutional layers, the formulation in Equation (25) is used. There are different graph convolutional neural network layers in the literature, but this one has shown to work well in protein function applications [32]. In the model developed in this thesis, the number of convolutional layers is fixed to three. However, the residue embedding dimensions are tuned by testing different alternatives. The activation matrices of each layer are then concatenated to achieve the feature matrix of a protein

$$H = [H^{(1)}, \dots, H^{(N_l)}] \in \mathbb{R}^{L \times \sum_{i=1}^{N_l} c_i}, \quad (28)$$

where N_l is the number of GCN layers and c_l is the amino acid embedding dimension of layer l .

5.3 Pooling layer

The output of the model should not depend on the length of the input protein. However, the size of the output of the graph convolutional network is dependent on the length of the amino acid sequence as shown in Equation (28). To achieve a prediction that does not depend on the length of the protein, a pooling layer is used to sum up the embedding vectors over the residues,

$$h = \sum_{i=1}^L H_i, \quad (29)$$

where H_i denotes the i th row of the H matrix and $h \in \mathbb{R}^{\sum_{l=1}^{N_l} c_l}$. Thus, one element in the vector h now represents the embedding of that element for all the amino acids.

As discussed at the beginning of Chapter 5, the external conditions are added to the model at two stages. The first one was already discussed in Section 5.1. In the second stage, the external conditions are concatenated to the feature vector h that was presented in Equation (29), which leads to a feature vector of dimension $h \in \mathbb{R}^{2 + \sum_{l=1}^{N_l} c_l}$ with two external conditions concatenated to the initial feature vector. After this, there are three fully connected linear layers. The first layer has the same dimension as h and the last layer is used to map the output to a single number. This number is the radius of gyration of the protein under the given conditions. The model is tested with three different variations: one of these variations uses the ReLU activation function between the linear layers, and the other two do not use any activation functions between the linear layers.

As the loss function of the model mean squared error loss is used. It is a popular loss function for regression problems where the predicted values are continuous. The mean squared error loss is the average of the squared differences between the predicted and the target values,

$$L = \frac{1}{N_{\text{batch}}} \sum_{i=1}^{N_{\text{batch}}} (y_{\text{pred}}^{(i)} - y_{\text{true}}^{(i)})^2, \quad (30)$$

where N_{batch} is the batch size, $y_{\text{pred}}^{(i)}$ is the prediction of the i th data point and $y_{\text{true}}^{(i)}$ is the true value of the i th data point. During the training the parameters are updated using the Adam optimizer [86].

5.4 Hyperparameter optimization using random search

One of the simplest hyperparameter optimization strategies is grid search and therefore, it has been widely utilized. In grid search, each parameter is given a set of

possible values, and the network is trained with all the possible combinations of these values to see which parameter combination would be optimal. However, with this approach, each parameter is tested only at the specific grid values, the number of which is usually low to reduce the computational burden. The computational cost increases exponentially with the number of hyperparameters. On the other hand, the low number of values does not represent well the whole space for that hyperparameter. In addition, if some hyperparameter is unimportant, meaning that its value does not affect the performance of the model significantly, a large number of computations are performed in vain, without learning any new information on the optimal hyperparameter values.

An alternative for grid search is random search. Random search has been shown to find as good or even better hyperparameters than grid search with smaller computational time [87]. In random search, the parameter values are chosen randomly for each parameter from an interval given to each parameter. With this method, the hyperparameter space can be explored more thoroughly. For the model created in this thesis, the hyperparameters that are considered are the learning rate, weight decay, length of the residue embeddings in all the GNN layers, and the size of the hidden layers of the two latter layers in the pooling block. As discussed in Section 5.3, the size of the first hidden layer of the pooling block is connected to the size of the residue embeddings, as shown by Equation (29). The learning rate and weight decay are parameters that are related to the optimizer. Commonly optimized hyperparameters, such as batch size and the number of layers are not considered in this thesis. Batch size is not considered, since the number of data points is relatively low, and the available computational resources are capable of handling the whole data at once. Thus, there is no need for dividing the data into batches. The number of layers is fixed since the three layers for the GNN and the pooling block are considered to be sufficient.

The intervals for each parameter are presented in Table 1. The number of randomly drawn hyperparameter combinations that are tested is 20 for each model variation that is tested in Section 6.1 and for the best model variation, the number of hyperparameter combinations is 30 in the final hyperparameter optimization in Section 6.2.

Table 1: Value intervals for the hyperparameters used in the random search hyperparameter optimization.

Hyperparameter	Value interval
GNN embedding 1	[10, 2000]
GNN embedding 2	[10, 1000]
GNN embedding 3	[10, 500]
Linear layer size 1	[10, 500]
Linear layer size 2	[10, 200]
Learning rate	$[10^{-4}, 10^{-2}]$
Weight decay	$[10^{-5}, 1.5 \cdot 10^{-3}]$

6 Results

In this chapter, the results obtained using the machine learning model are presented. This includes determining the details of the model architecture, as well as the hyperparameters of the model. Section 6.1 presents the result of the three different model architecture alternatives, each of them with two different cut-off values for the adjacency matrix. Section 6.2 evaluates the models performance with different hyperparameter combinations and proposes the optimal hyperparameters for the model. After determining the optimal set of hyperparameters the model's performance is assessed in more detail in Section 6.3. Lastly, the model's predictions are compared to experimentally measured absorbance values in Section 6.4.

6.1 Model architecture

As discussed in Chapter 5, three variations of the model are tested during the evaluation of the model. The first variant only has ReLU activation functions between the GCN layers and there are no activation functions between the linear layers. The first variant also has a constant learning rate. The second variant is similar to the first one regarding the activation functions. However, instead of using a constant learning rate, it utilizes a learning rate scheduler. The third variant also uses the learning rate scheduler. It also uses the ReLU activation function between the linear layers. Each of the model variants is trained using 20 sets of randomly determined hyperparameters. The hyperparameters are drawn from uniform distributions for which the intervals are defined in Section 5.4.

In order to assess the performance of the models also for unseen data and validate them for the simulated data, the data was divided into training and validation sets. Since the data set contains multiple different conditions for a single protein, it is necessary to divide it so that there are no data points for one protein in both data sets. If the model would have data points of one protein in both data sets, the data points in the validation data set could not be considered to be entirely unseen by the model. Therefore, when dividing the data to the training and validation sets all the data points of each protein are entirely in one of the data sets. In addition, the same training and validation data sets are used for each of the variations of the model. This is done to ensure that the division of data does not affect the performance of the model. Since the entire data set is not particularly large, the training data set was chosen to be as large as possible. Therefore, only two proteins out of the 133 were randomly chosen to be in the validation data set. Thus, the validation data set only contains 62 data points from the total 4017 data points. With such a small validation data set, the validation errors might be highly dependent on the selection of the proteins for the validation data set.

The training errors for models that used an adjacency matrix cut-off value of 6.5\AA are presented in Figure 5 and those for the models that used a cut-off value of 10.0\AA are presented in Figure 6. In these figures, logarithmic scales are used on both axes since the values of the training errors have a wide range. To highlight the errors at the end of the training between epochs 400 and 500, an inset is also plotted in

each figure for these epochs with linear scales on the axes. From these figures, it can be seen that all the model variants and adjacency matrix cut-off values have high oscillations in the training error for most of the models at the beginning. This is likely due to the learning rate being too large in the beginning. However, especially with the model variants with learning rate schedulers in Figures 5b, 5c, 6b and 6c, the oscillations stop after 100 epochs. The learning rate scheduler decreases the learning rate for the first time after the first 50 epochs. Based on these figures, the oscillations continue slightly longer for the variants that used the adjacency matrix cut-off value of 6.5Å. However, these differences are small and can be due to the different hyperparameters of the models that were chosen randomly.

Based on Figures 5a and 6a for the models with constant learning rates, the training error still oscillates at the end for many of the models, especially for the models that used the adjacency matrix cut-off value of 6.5Å. This is likely due to the fact that without the learning rate scheduler, the models might have too large learning rate at the end so that they do not settle to a local minimum that well. On the other hand, with the constant learning rate the minimum training errors are lower than for other variants. With the adjacency cut-off value of 6.5Å, the minimum training error that is achieved during the 500 epoch with a constant learning rate is approximately 41, and with the cut-off value of 10.0Å, the minimum training error is approximately 45.

For the other two variants and both cut-off values, the errors are nearly constant during the last 100 epochs which can be seen from the insets of Figures 5b, 5c, 6b and 6c. The minimum training errors for the model variants with the learning rate scheduler and ReLU activation only between the GCN layers are approximately 49 and 47 for the models with 6.5Å and 10.0Å adjacency cut-off values, respectively. For the models with ReLU activation between all the layers and learning rate scheduler, the minimum training error for the models that used an adjacency cut-off value of 6.5Å is approximately 50 and for the models that used a cut-off value of 10.0Å it is approximately 47. The minimum training errors with the means squared error loss function are summarized in Table 2. The values in the table represent the average of the squared differences between the predicted and simulated values. The relative difference of the minimum training errors between the models with learning rate scheduler and adjacency matrix cut-off value 6.5Å is approximately 1.1% and for the models that used a cut-off value of 10.0Å is approximately 0.02%. Based on these values, if the model uses a training scheduler, adding ReLU activations between the linear layers does not improve the performance significantly. From Table 2 it can be observed that for the two model variants with learning rate schedulers, increasing the adjacency matrix cut-off value seems to lower the minimum training error slightly. However, for the model variant with a constant learning rate, increasing the adjacency cut-off value has the opposite effect and the training error is larger.

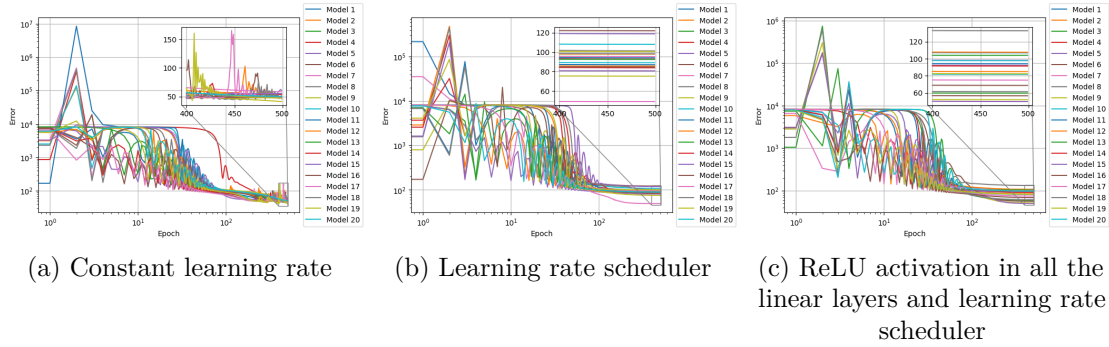


Figure 5: Training errors of the three different model architectures for 500 epochs with logarithmic scales on both axis and with adjacency matrix cut-off value of 6.5\AA . The different models correspond to different randomly chosen sets of hyperparameter values.

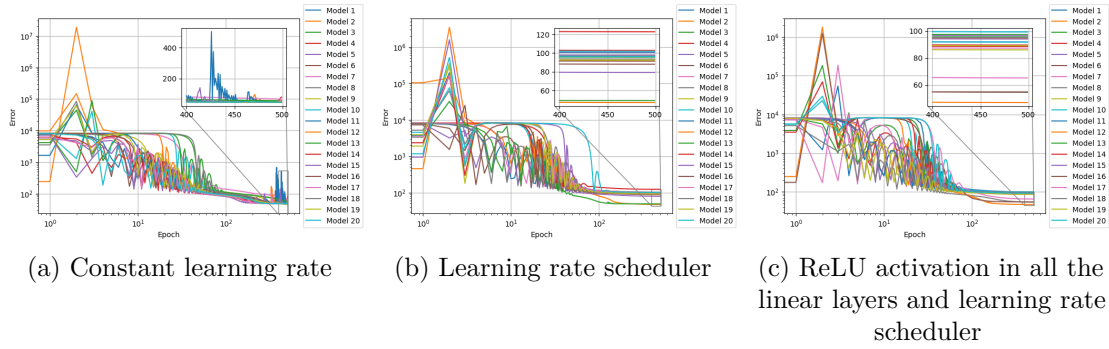


Figure 6: Training errors of the three different model architectures for 500 epochs with logarithmic scales on both axis and with adjacency matrix cut-off value of 10.0\AA . The different models correspond to different randomly chosen sets of hyperparameter values.

Table 2: Minimum training and validation error of the 20 models for each model variation using the mean squared error loss function.

Model variation	Adjacency matrix cut-off value (\AA)	Minimum training error	Minimum validation error
Constant learning rate	6.5	41.395	71.951
	10.0	44.967	73.579
Learning rate scheduler	6.5	49.275	85.933
	10.0	46.870	75.852
ReLU activation in all the linear layers and learning rate scheduler	6.5	49.838	79.436
	10.0	46.859	86.515

The validation errors for models that used an adjacency matrix cut-off value of 6.5\AA are presented in Figure 7 and those for the models that used a cut-off value of 10.0\AA are presented in Figure 8. As with the figures of the training errors, these figures use logarithmic scales on both axes since the values of the validation errors have a wide range. To highlight the errors at the end of the training, between epochs 400 and 500 an inset is also plotted in each figure for these epochs with linear scales on the axes. As with the training errors, some of the models with the constant learning rate are still fluctuating during the last 100 epochs as can be seen from the insets of Figures 7a and 8a. This fluctuation occurs for more models when the adjacency matrix cut-off value of 6.5\AA is used. Figures 7b, 7c, 8b and 8c show that the validation errors also oscillate for the other model variants and adjacency matrix cut-off values. When compared to the training errors of the corresponding model variants the oscillation lasts slightly longer. However, it also diminishes quickly after the first 100 epochs since the learning rate scheduler decreases the learning rate at that point for the second time.

For the models with learning rate schedulers, the validation errors of the 20 models have a larger variation during the last 100 epochs. For example, for the model with the learning rate scheduler and without ReLU activations between the linear layers, the training errors for the last 100 epochs with both of the cut-off values are approximately between 47 and 125 as can be seen from Figures 5b and 6b. However, for the corresponding validation errors, shown in Figures 7b and 8b, the errors are approximately between 75 and 230 during the last 100 epochs.

The minimum validation errors of all 20 models for each model variant and adjacency matrix cut-off value are presented in Table 2. From the table, it can be seen that the lowest validation errors are achieved by the models with constant learning rates. Out of these two model variants, the models using the adjacency matrix cut-off value of 6.5\AA have slightly lower minimum validation error. For the two model variants that used the learning rate scheduler, the adjacency matrix cut-off value does have a large effect on the validation error. However, the effect of increasing the adjacency matrix cut-off value is the opposite for the two model variants. For the model without the ReLU activations between the linear layers, increasing the adjacency matrix cut-off value decreases the minimum validation error and vice versa for the model variant with the ReLU activation between the linear layers. This inconsistency can be due to the fact that the validation data set is small. Therefore there can be large variations in the validation errors.

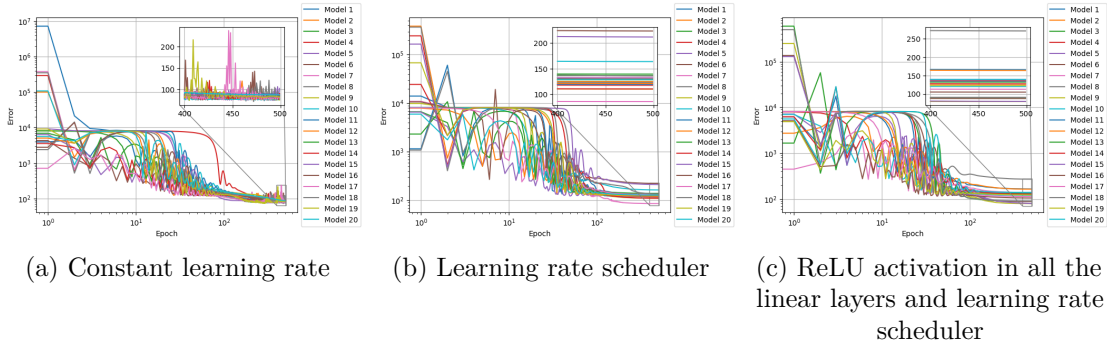


Figure 7: Validation errors of the three different model architectures for 500 epochs with logarithmic scales on both axis and with adjacency matrix cut-off value of 6.5\AA . The different models correspond to different randomly chosen sets of hyperparameter values.

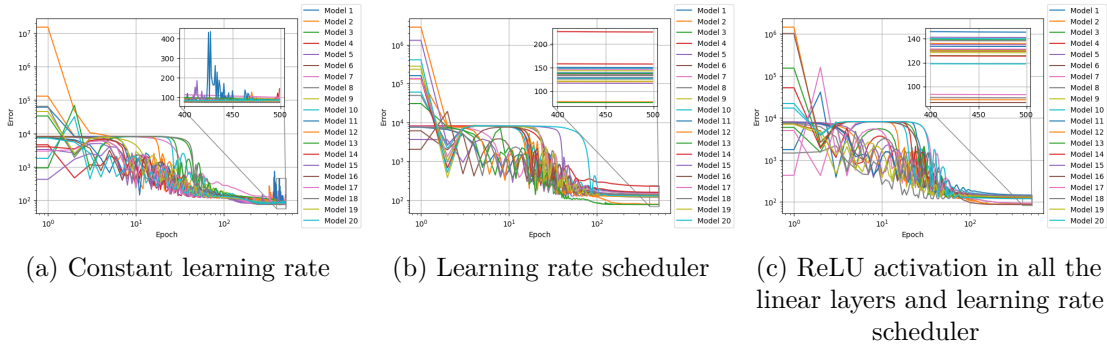


Figure 8: Validation errors of the three different model architectures for 500 epochs with logarithmic scales on both axis and with adjacency matrix cut-off value of 10.0\AA . The different models correspond to different randomly chosen sets of hyperparameter values.

Based on the training and validation errors, the model variant with the learning rate scheduler and without the ReLU activations between the linear layers that used the adjacency matrix cut-off value of 10.0\AA is chosen for further investigation. This model variant had the best overall performance of all the model variants. The chosen model variant does not have as low training and validation errors as the models with constant learning rates. However, the difference is fairly small and the chosen model variant might reach similar errors with more epochs or different parametrization for the learning rate scheduler. In addition, the models with constant learning rates had fluctuations in the errors even at the end of the training, and therefore, they are not considered further. Compared to the other model variants with the learning rate scheduler, the chosen one has the smallest validation error. Furthermore, the training error is almost the same as with the model variant that has ReLU activations between the linear layers and the adjacency matrix cut-off value of 10.0\AA . In the next section, the hyperparameters of the chosen model variant are analysed further.

6.2 Hyperparameter optimization

In this section, the hyperparameters of the model variant that was chosen in the previous section are analysed further. The hyperparameter optimization is conducted using random search as discussed in Section 5.4. The parameter value intervals are presented in Table 1. Instead of using the 20 models with random parameters as in the previous section, the number of models was slightly increased to 30 models. In addition, the number of epochs was increased to 800 and the step size of the learning rate scheduler was increased to 100. Increasing the step size of the learning rate scheduler to 100 means that the learning rate is updated every 100th epoch instead of the 50 used in Section 6.1. The reason for increasing the step length was to have more epochs with higher learning rates and as a result, to achieve smaller errors at the end. During the hyperparameter optimization, the data is divided into training and validation data sets. These data sets are the same as in Section 6.1. Thus, the validation data set only contains the entire data of two proteins which means 62 data points.

Training and validation errors for the 30 models are presented in Figure 9. As with the errors for the model variations, the errors are visualized with logarithmic scales on both axes except for the inset that presents the last 100 epochs with linear scales on both of the axes. As can be seen from the training errors in Figure 9a, model number 30 clearly has the lowest training error at the end, approximately 35.4. This is also significantly lower than the lowest training error for the similar model variant in Figure 5b. Thus, the small modifications of increasing the number of epochs and the step size of the learning rate scheduler that were done after the model variant selection appear to provide smaller training errors. As can be seen from Figure 9b, model 30 also has the lowest validation error out of the 30 models. At the end, the validation error is approximately 59.2. As with the training error, this is significantly lower than the lowest validation error of the same model variant in Figure 7b. Both of the errors experience some oscillation, especially before the 100 epoch mark. However, the oscillations do not continue notably longer than the oscillations for the corresponding model variants in Figures 5b and 7b. Thus, increasing the learning rate scheduler's step size does not increase the oscillations to a point that would make the model unacceptable only based on the oscillations.

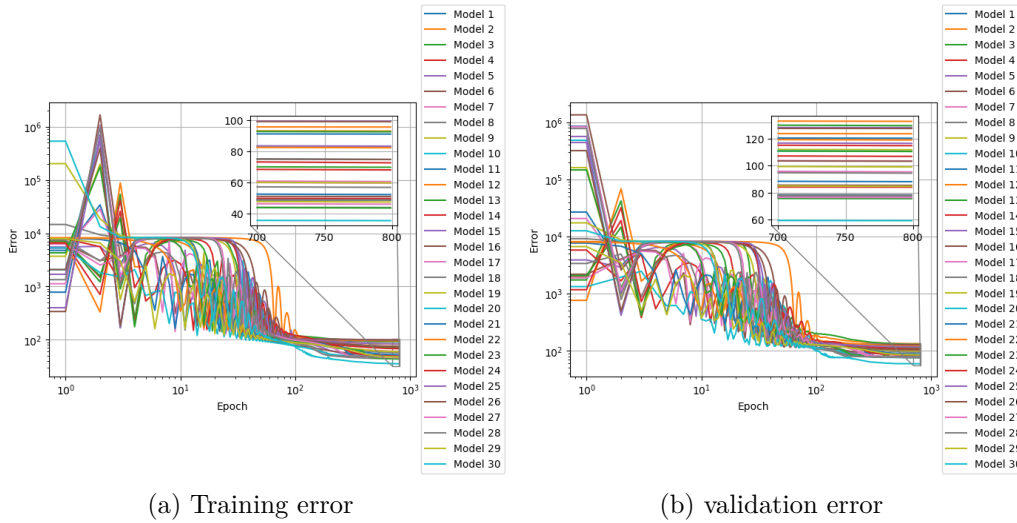


Figure 9: Training and validation error for 30 models with random hyperparameters, 10.0\AA adjacency matrix cut of value, learning rate scheduler and no activation functions between the linear layers.

To further analyse the effect of the hyperparameters to the model performance, each hyperparameter is analysed separately. The hyperparameters effect is analyzed by calculating the correlation coefficients between the hyperparameter and the minimum training error for all the 30 models. Table 3, presents the Pearson correlation coefficients. The positive correlations in the table correspond to hyperparameters that decrease the minimum error when the hyperparameters value decreases and for the negative correlations the minimum error decreases when the value of the hyperparameter increases.

Based on Table 3, the hyperparameter that has the largest effect on the model performance is the weight decay. As described in more detail in Section 5.4, the weight decay is a regularization method for neural networks. The weight decay is a penalty term that is assigned to the weights of the model and incorporated to the total loss function. Thus, the model is not as likely to use single large weights because they are penalized. The second parameter that is also related to the optimizer of the model is the learning rate, which determines how large steps the model takes towards the negative gradient at each step. However, the Adam optimizer also takes into account the previous gradient, but in general the learning rate can be considered as the speed at which the optimizer moves towards the minimum. For both of these hyperparameters, the correlation is negative meaning that the model has lower training error the higher these hyperparameters are. The maximum values for these two hyperparameters were fairly small. For sufficiently large values the behaviour might change. For the model 30 that performed the best, the weight decay and learning rate had approximately values 0.0015 and 0.010, respectively. Both of these values were close to the upper bound of the uniform distribution where these values were obtained.

Table 3: Pearson correlation coefficients between the minimum training errors and the hyperparameter values.

Hyperparameter	Correlation coefficient
GNN embedding 1	-0.3841
GNN embedding 2	0.0952
GNN embedding 3	0.1258
Linear layer size 1	-0.0963
Linear layer size 2	-0.0097
Learning rate	-0.1936
Weight decay	-0.6007

From the hyperparameters that control the architecture of the model, meaning the length of the embedding vectors in the GNN layers or the size of the linear layers, the size of the embeddings in the first GNN layer has the largest correlation with the minimum training error. For this hyperparameter, the correlation would suggest that increasing the value would make the minimum training errors smaller. This agrees well with the assumption that training the model can fit better to the data when there is a higher number of parameters. For the model 30 which had the lowest training error the size of the embedding in the first GNN layer was 1752. This is close to the upper bound of that hyperparameter which was 2000. The other two GNN embedding hyperparameters were 966 and 194. The sizes of the linear layers were 300 and 96.

The correlation coefficients for the other architecture parameters are much closer to zero, which means that their effect on the minimum training error is smaller. However, all of the parameters are related to each other and are not independent. Therefore, drawing conclusions about the effects of single hyperparameters is difficult. Figure 10 shows the minimum training error as a function of the weight decay. As can be seen from the figure, even for the hyperparameter with the largest absolute value of correlation the dependency is not that clearly visible. And as stated earlier, drawing conclusions for single hyperparameters is not that simple. The hyperparameters are not optimized any further, and in the upcoming sections the model 30 will be used to analyse how the method performs on single proteins.

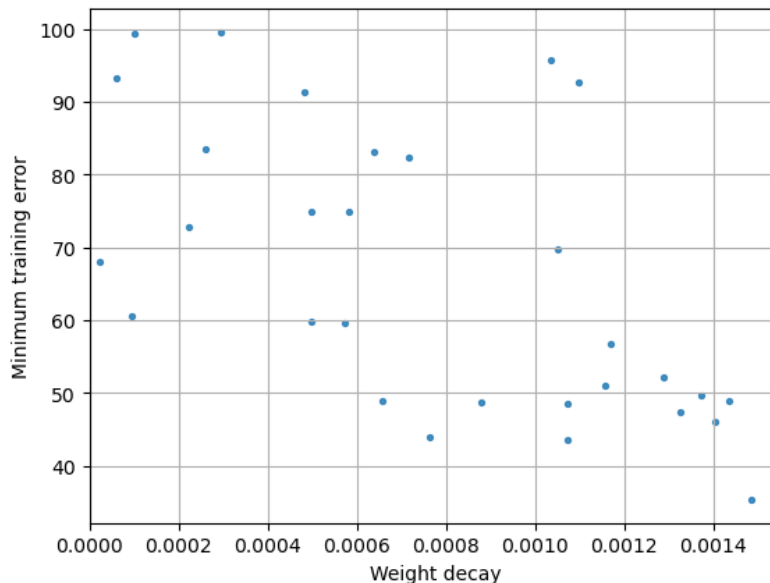


Figure 10: The minimum training error as a function of the weight decay hyperparameter.

6.3 Training and validation error

In this section the model that was determined to perform the best in Section 6.2 is analysed further. The performance of the model for the entire training data and the unseen validation data are analysed first. After that, few proteins and their predictions are analysed in more detail.

The difference between the predicted and the simulated values of the radius of gyration for the training data are presented in Figure 11. In this figure, the points where the predictions are exactly the same as the simulated values, considered as the true values, are visualized with a red dashed line. From the figure, it can be seen that overall the points are fairly evenly distributed on both sides of the line. This suggests that the model does not over or underpredict. In addition, the average of the absolute values of the differences between the predicted and simulated values is only 4.6Å.

However, the differences in the predicted and true values seem to be larger when the true values are either low or high. This can be seen from the ranges of the points. The simulated values range from approximately 40Å to 135Å and the predicted values only range from approximately 55Å to 115Å. This is considerably smaller range and implies that the model predicts values more close to the average of the training data. This is somewhat expected since the values of the radius of gyration are not evenly distributed. As can be seen from Figure 11, there are only few data points with extremely low and high simulated radii of gyration. Moreover, these points are the same points that have the largest differences between the predicted and simulated values.

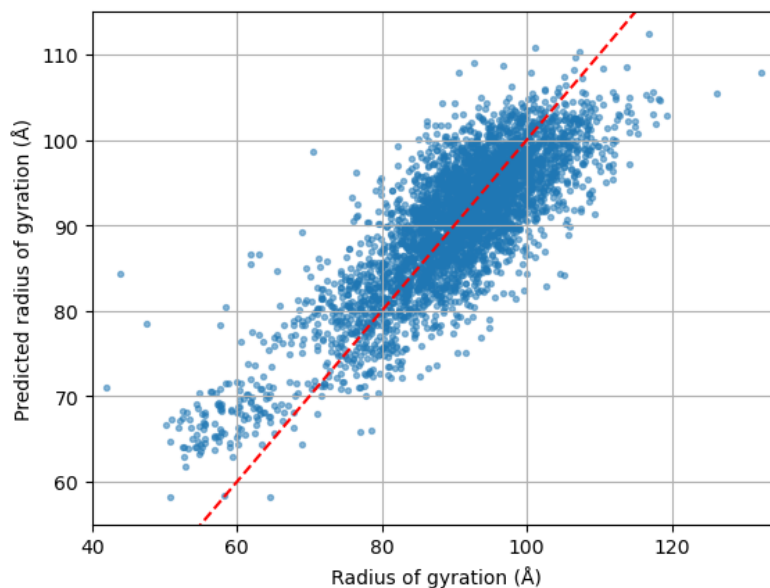


Figure 11: The predicted radius of gyrations as a function of the simulated radius of gyrations for the training data set and a dashed line where the predictions are equal to the simulated values.

The difference between the predicted and the simulated values of the radius of gyration for the 62 validation data points are presented in Figure 12. These data points are considered as unseen data for the model since they were not used in the training of the model, even though they slightly guided the selection of the model variant and hyperparameters. As with the training data, the points are evenly distributed on both sides of the line which represents the perfect predictions. The average of the absolute values of the differences between the predicted and simulated radii of gyration is approximately 6.2. This is significantly larger than the corresponding number for the training data. However, this difference can be highly dependent on the selection of the validation data set especially when the validation set is this small. As with the training data, the model seems to have larger errors for the predictions when the simulated value is high or low. This further implies that the model is not as capable of predicting the extreme cases where the radius of gyration is small or high.

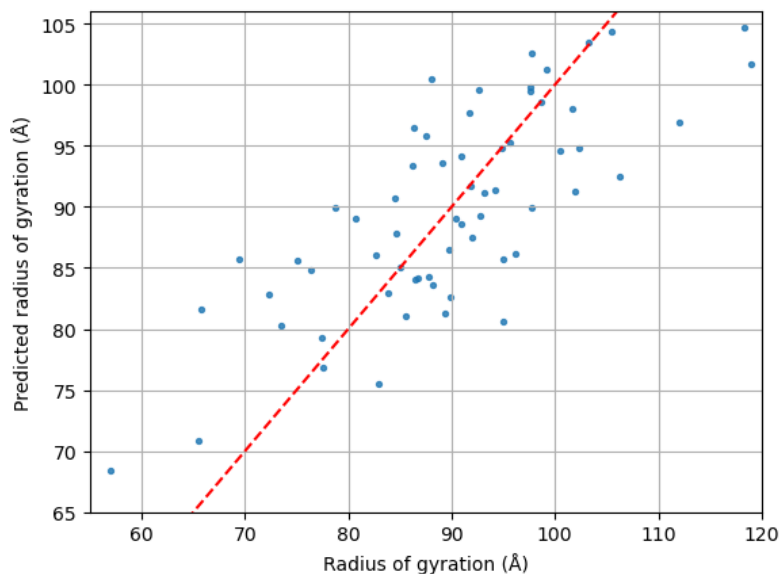


Figure 12: The predicted radius of gyrations as a function of the simulated radius of gyrations for the validation data set and a dashed line where the predictions are equal to the simulated values.

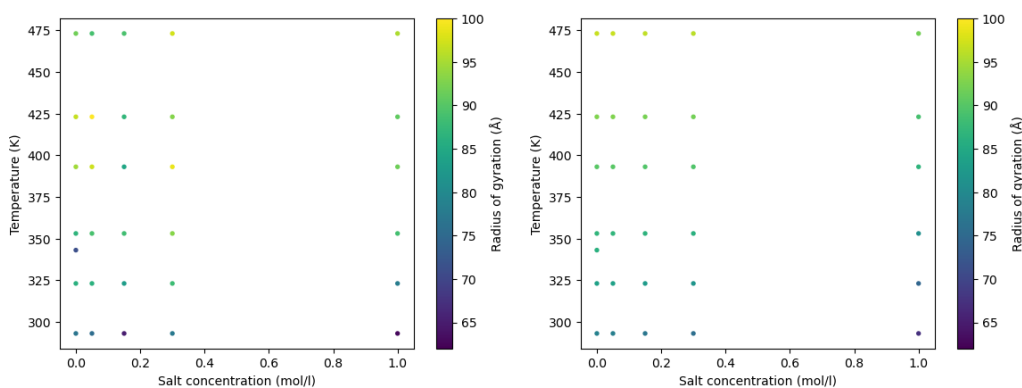
To further assess the capabilities of the model, one protein from the training and validation data set are analysed further. The two proteins are chosen so that there is also experimental data available for them. Figure 13 presents the simulated and predicted values for the radius of gyration as a function of salt concentration and temperature for protein number 87. This protein was in the training data set. For a protein that is in the training data set, the model should be able to predict the gyration well for different temperatures and salt concentrations.

For this particular protein, it seems that the model was not able to learn the radius of gyration that well for different temperatures and salt concentrations. For the simulated values, the effect of the conditions on the radius of gyration seems to be more complicated than for the predicted values. For the predicted radius of gyration, the values seem to increase almost linearly as the salt concentration decreases and the temperature increases. However, the simulated values do not have as clear dependency between the radius of gyration and the conditions. For example, for the simulated values with a temperature of approximately 425K, the radius of gyration first increases then decreases and again increases as a function of the salt concentration. For the predicted values, the radius of gyration only decreases as a function of the salt concentration. This observation indicates that the model is not able to predict the nonlinear effect of the conditions that is seen in the simulated values of the radius of gyration. However, the model seems to be able to capture the trend that can be seen in the simulated values. In general, the radius of gyration seems to be growing when the temperatures increases and salt concentration decreases for the simulated data in Figure 13a. This similar trend can be seen from the predicted values in Figure 13b, and it is more clear since the radius of gyration values are in ascending order when the temperature increases and salt

concentration decreases.

Figure 14 presents the simulated and predicted values for the radius of gyration as a function of salt concentration and temperature for protein number 142. This protein was in the validation data set. As expected, the predictions suffer from the same limitation as the example protein that was in the training data set: for the predictions the values of the radius of gyration increase when the temperature increases and salt concentration decreases. However, for the simulated values, the behaviour of the radius of gyration is far more complicated as a function of the conditions. As with the protein from the training data, the model still predicts the trend of the radius of gyration somewhat correctly.

The predictions for both of the proteins seem to have very similar trends. This could mean that the model is not capable of predicting different condition dependencies for the proteins and each protein would have a similar trend in the prediction. However, the simulated values of these proteins are not that different regarding the trend of the radius of gyration, and therefore the predictions might look similar.



(a) Simulated values for radius of gyration (b) Predicted values for radius of gyration

Figure 13: Simulated and predicted values for the radius of gyration as a function of salt concentration and temperature for protein number 87.

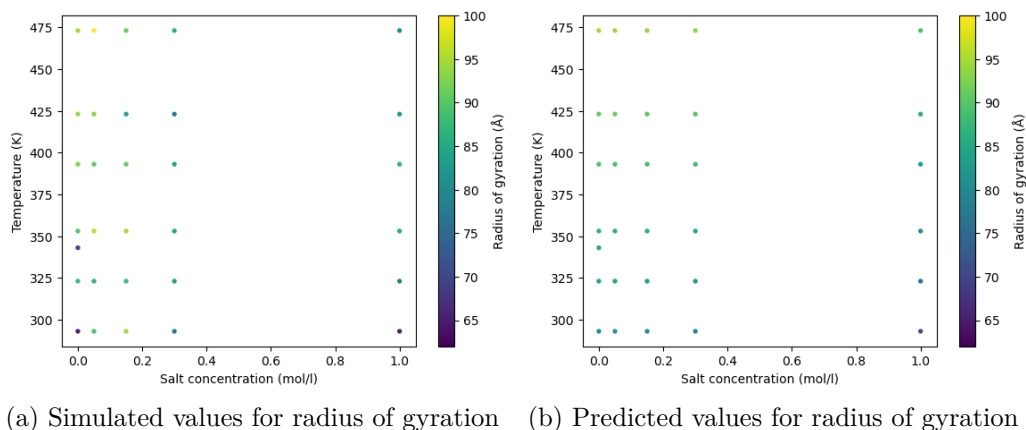


Figure 14: Simulated and predicted values for the radius of gyration as a function of salt concentration and temperature for protein number 142.

6.4 Performance on experimental data

The performance of the model with experimental data depends on how well the simulated data corresponds to the experimental data. For the comparison with the experimental data, the protein 87 from the previous section is used. Since the experiments used absorbance as the measure of the LLPS and the simulations used the radius of gyration, the comparison of the magnitudes of these numbers is meaningless. However, a more qualitative comparison can be made by comparing the trends of these two metrics.

Figure 15 presents the absorbance of the protein 87. As can be seen from the figure, the ranges for the conditions are different for the experimental data: the salt concentration range is from 0mol/l to 2mol/l which is twice the size of the range for the simulated data. On the other hand, for the temperature the simulated data has significantly larger range.

By comparing the simulated values for the radius of gyration for the protein 87 in Figure 13a and the experimental data for the same protein in Figure 15, it can be seen that there are large differences in the trends of the LLPS metrics. For the experimental data, increasing salt concentration increases the absorbance. As discussed earlier, the simulated data has opposite behaviour as a function of the salt concentration. For this protein, the temperature's effect is that it increases the absorbance until around 350K after which increasing temperature starts to decrease the absorbance. Altogether, the simulated values do not have as clear an area where the LLPS metric is high. This could indicate that the chosen metric for the LLPS is sensitive for small fluctuations in the simulations or that the metric itself is not suitable as a metric for LLPS.

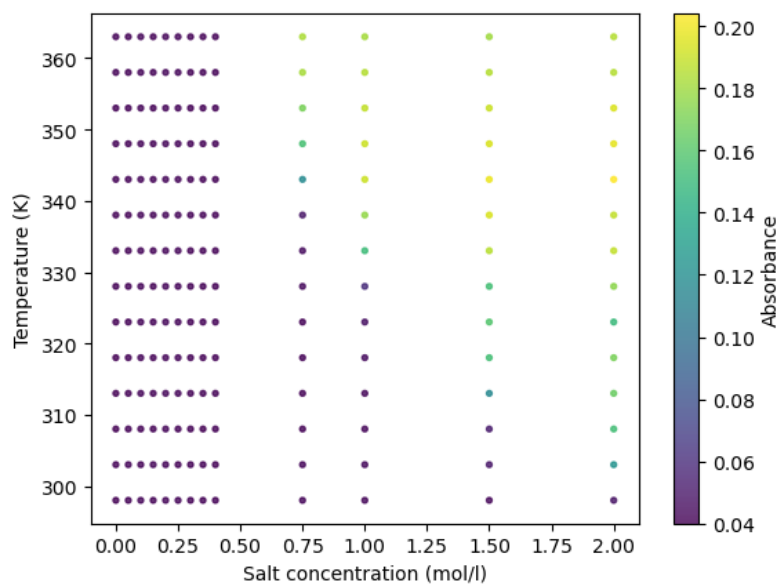


Figure 15: Experimentally measured absorbance of protein 87.

7 Conclusions

A phenomenon for proteins called liquid-liquid phase separation (LLPS) has been shown to have a role in many diseases and biological processes. Many proteins can undergo LLPS. However, whether or not LLPS occurs depends on the conditions of the protein solution. Determining the LLPS behaviour of proteins through experimental or simulation methods is time-consuming and not feasible for a large number of proteins. Thus, there is a need for a machine learning model that could reliably predict the LLPS under different conditions. The previous machine learning approaches have not considered the effect of the conditions on the LLPS behaviour of proteins. Therefore, this thesis aimed to develop a machine learning model to predict the LLPS of proteins under different conditions. The conditions chosen to be used in this thesis were the temperature and salt concentration. The studied machine learning model consists of a graph neural network that utilizes the structure of the proteins in the predictions. In addition, the model is able to take into account the conditions under which the LLPS occurs by incorporating the conditions at two points into the model. A few different architecture variations of the general model were tested to assess which performs best.

The experimental data available was not sufficient to train a graph neural network. Therefore, simulated data was used to train the model. To quantify the LLPS from the simulations, the radius of gyration was used as a metric for the LLPS. Therefore, the model was designed to predict the radius of gyration. The predictions were also compared to experimental data. However, due to the different metrics used for the LLPS between the experimental and simulated data, only a qualitative comparison could be made between the model predictions and the experimental data.

The differences between the predictions and the simulated values suggested that the model was able to learn some features from the protein structures that affected the radius of gyration. However, the model was not able to learn the complicated condition dependencies that were present in the simulated data. For the predicted radii of gyration, the values seemed to behave linearly as a function of the conditions, whereas the simulated values had more complicated temperature and salt concentration dependencies. In addition, the predictions were compared to experimental data for one protein. This experimental data was not used in the model training or validation, although simulated data for the same protein was. Due to a large difference between the experimental and simulated data for the protein, the model was not able to predict the LLPS accurately. The predictions could not be compared to any previous prediction methods since none of them has considered the conditions under which the LLPS occurs.

With the current architecture, the model is not suitable for accurate predictions of LLPS. The radius of gyration values calculated from the simulations significantly differ from the absorbance values of the experimental data. In addition, the simulated values have larger fluctuations between different conditions when compared to the experimental values that behave smoothly as a function of the conditions. Thus, even if the model could predict the radius of gyration correctly, the prediction would still not be reliable for the LLPS. This can be due to the radius of gyration not

being the correct metric for the LLPS. Moreover, the radius of gyration might be affected by the time scale of the simulation. As a result, even though the simulations seemed to reach equilibrium based on visually inspecting the simulations the radius of gyration might have not reached equilibrium. Additionally, the time that it took for the proteins to reach equilibrium in the experiments might have been considerably higher. The utilization of the model for LLPS is also limited by the accuracy of the simulation itself.

The performance of the model was also limited by the available data. The data set that was used for training the model was small compared to other applications that have used graph neural networks, where the data set may have contained millions of data points. In addition, the proteins in the data set were all fairly similar. Thus, there is no guarantee on how the model would perform in predicting proteins with significantly different structures or properties.

Since the model was not capable of predicting the complicated condition dependencies of the proteins, the architecture of the model would need improvement to be able to predict the LLPS accurately. This could be done by adding more GNN or linear layers to the model. In addition, nonlinear activation functions could be used between these layers. Another approach would be to use different layers in the GNN instead of the graph convolutional layers.

Due to the difference between the simulated and experimental LLPS metrics, some other metric for the simulated LLPS could be considered. For the model to be able to predict the LLPS, the training data should correspond to the experimental data. These metrics could be some clustering methods where the number of clusters would correspond to the amount of phase separation in the simulation. In addition, only calculating the radius of gyration at the last step of the model is not a robust method since there is fluctuation in the simulations. Thus, one possibility would be to use some average of the metric over multiple time steps.

To further improve the performance of the model, a larger and more diverse data set should be used. This would allow the model to be able to also predict the LLPS of proteins that are not similar to the proteins that were used to train the model developed in this thesis. Making the protein data set even larger could even help the model to predict better proteins that are different from all the proteins that were used for training. This could be achieved by covering a wide range of proteins with different structures and properties which would allow the model to predict also proteins between these points rather than have to rely on extrapolating outside the training data set.

References

- [1] Y. Shin and C. P. Brangwynne, “Liquid phase condensation in cell physiology and disease,” *Science*, vol. 357, no. 6357, 2017.
- [2] J. M. Shulman, P. L. De Jager, and M. B. Feany, “Parkinson’s disease: Genetics and pathogenesis,” *Annual Review of Pathology: Mechanisms of Disease*, vol. 6, pp. 193–222, 2011.
- [3] P. Brundin, R. Melki, and R. Kopito, “Prion-like transmission of protein aggregates in neurodegenerative diseases,” *Nature reviews Molecular cell biology*, vol. 11, no. 4, pp. 301–307, 2010.
- [4] W. Robberecht and T. Philips, “The changing scene of amyotrophic lateral sclerosis,” *Nature Reviews Neuroscience*, vol. 14, no. 4, pp. 248–264, 2013.
- [5] S. F. Banani, H. O. Lee, A. A. Hyman, and M. K. Rosen, “Biomolecular condensates: Organizers of cellular biochemistry,” *Nature reviews Molecular cell biology*, vol. 18, no. 5, pp. 285–298, 2017.
- [6] A. Boija, I. A. Klein, B. R. Sabari, A. Dall’Agnese, E. L. Coffey, A. V. Zamudio, C. H. Li, K. Shrinivas, J. C. Manteiga, N. M. Hannett, *et al.*, “Transcription factors activate genes through the phase-separation capacity of their activation domains,” *Cell*, vol. 175, no. 7, pp. 1842–1855, 2018.
- [7] C. P. Brangwynne, C. R. Eckmann, D. S. Courson, A. Rybarska, C. Hoegge, J. Gharakhani, F. Jülicher, and A. A. Hyman, “Germline p granules are liquid droplets that localize by controlled dissolution/condensation,” *Science*, vol. 324, no. 5935, pp. 1729–1732, 2009.
- [8] A. A. Hyman, C. A. Weber, and F. Jülicher, “Liquid-liquid phase separation in biology,” *Annual review of cell and developmental biology*, vol. 30, pp. 39–58, 2014.
- [9] E. Gomes and J. Shorter, “The molecular language of membraneless organelles,” *Journal of Biological Chemistry*, vol. 294, no. 18, pp. 7115–7127, 2019.
- [10] B. Shen, Z. Chen, C. Yu, T. Chen, M. Shi, and T. Li, “Computational screening of phase-separating proteins,” *Genomics, proteomics & bioinformatics*, vol. 19, no. 1, p. 13, 2021.
- [11] V. N. Uversky, “Intrinsically disordered proteins in overcrowded milieu: Membraneless organelles, phase separation, and intrinsic disorder,” *Current opinion in structural biology*, vol. 44, pp. 18–30, 2017.
- [12] C. A. Orengo, A. E. Todd, and J. M. Thornton, “From protein structure to function,” *Current opinion in structural biology*, vol. 9, no. 3, pp. 374–382, 1999.
- [13] H. Hegyi and M. Gerstein, “The relationship between protein structure and function: A comprehensive survey with application to the yeast genome,” *Journal of molecular biology*, vol. 288, no. 1, pp. 147–164, 1999.

- [14] P. Tompa, “Intrinsically disordered proteins: A 10-year recap,” *Trends in biochemical sciences*, vol. 37, no. 12, pp. 509–516, 2012.
- [15] P. E. Wright and H. J. Dyson, “Intrinsically unstructured proteins: Re-assessing the protein structure-function paradigm,” *Journal of molecular biology*, vol. 293, no. 2, pp. 321–331, 1999.
- [16] S. Alberti, A. Gladfelter, and T. Mittag, “Considerations and challenges in studying liquid-liquid phase separation and biomolecular condensates,” *Cell*, vol. 176, no. 3, pp. 419–434, 2019.
- [17] G. L. Dignon, W. Zheng, and J. Mittal, “Simulation methods for liquid-liquid phase separation of disordered proteins,” *Current Opinion in Chemical Engineering*, vol. 23, pp. 92–98, 2019.
- [18] G. L. Dignon, W. Zheng, R. B. Best, Y. C. Kim, and J. Mittal, “Relation between single-molecule properties and phase behavior of intrinsically disordered proteins,” *Proceedings of the National Academy of Sciences*, vol. 115, no. 40, pp. 9929–9934, 2018.
- [19] R. M. Regy, W. Zheng, and J. Mittal, “Using a sequence-specific coarse-grained model for studying protein liquid-liquid phase separation,” in *Methods in Enzymology*, vol. 646, Elsevier, 2021, pp. 1–17.
- [20] A. C. Murthy, G. L. Dignon, Y. Kan, G. H. Zerze, S. H. Parekh, J. Mittal, and N. L. Fawzi, “Molecular interactions underlying liquid-liquid phase separation of the fus low-complexity domain,” *Nature structural & molecular biology*, vol. 26, no. 7, pp. 637–648, 2019.
- [21] T. Sun, Q. Li, Y. Xu, Z. Zhang, L. Lai, and J. Pei, “Prediction of liquid-liquid phase separation proteins using machine learning,” *Available at SSRN 3515387*, 2020.
- [22] K. L. Saar, A. S. Morgunov, R. Qi, W. E. Arter, G. Krainer, T. Knowles, *et al.*, “Machine learning models for predicting protein condensate formation from sequence determinants and embeddings,” *bioRxiv*, 2020. DOI: [10.1101/2020.10.26.354753](https://doi.org/10.1101/2020.10.26.354753).
- [23] G. van Mierlo, J. R. Jansen, J. Wang, I. Poser, S. J. van Heeringen, and M. Vermeulen, “Predicting protein condensate formation using machine learning,” *Cell Reports*, vol. 34, no. 5, p. 108705, 2021.
- [24] S. Alberti, R. Halfmann, O. King, A. Kapila, and S. Lindquist, “A systematic survey identifies prions and illuminates sequence features of prionogenic proteins,” *Cell*, vol. 137, no. 1, pp. 146–158, 2009.
- [25] 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, 2014.

- [26] M. P. Hughes, M. R. Sawaya, D. R. Boyer, L. Goldschmidt, J. A. Rodriguez, D. Cascio, L. Chong, T. Gonen, and D. S. Eisenberg, “Atomic structures of low-complexity protein segments reveal kinked β sheets that assemble networks,” *Science*, vol. 359, no. 6376, pp. 698–701, 2018.
- [27] J. Wang, J.-M. Choi, A. S. Holehouse, H. O. Lee, X. Zhang, M. Jahnel, S. Maharana, R. Lemaitre, A. Pozniakovsky, D. Drechsel, *et al.*, “A molecular grammar governing the driving forces for phase separation of prion-like rna binding proteins,” *Cell*, vol. 174, no. 3, pp. 688–699, 2018.
- [28] T. J. Nott, E. Petsalaki, P. Farber, D. Jervis, E. Fussner, A. Plochowitz, T. D. Craggs, D. P. Bazett-Jones, T. Pawson, J. D. Forman-Kay, *et al.*, “Phase transition of a disordered nuage protein generates environmentally responsive membraneless organelles,” *Molecular cell*, vol. 57, no. 5, pp. 936–947, 2015.
- [29] B. Bolognesi, N. L. Gotor, R. Dhar, D. Cirillo, M. Baldrighi, G. G. Tartaglia, and B. Lehner, “A concentration-dependent liquid phase separation can cause toxicity upon increased protein expression,” *Cell reports*, vol. 16, no. 1, pp. 222–231, 2016.
- [30] R. M. Vernon, P. A. Chong, B. Tsang, T. H. Kim, A. Bah, P. Farber, H. Lin, and J. D. Forman-Kay, “Pi-pi contacts are an overlooked protein feature relevant to phase separation,” *elife*, vol. 7, e31486, 2018.
- [31] 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, 2019.
- [32] V. Gligorijević, P. D. Renfrew, T. Kosciolk, J. K. Leman, D. Berenberg, T. Vatanen, C. Chandler, B. C. Taylor, I. M. Fisk, H. Vlamakis, *et al.*, “Structure-based protein function prediction using graph convolutional networks,” *Nature communications*, vol. 12, no. 1, pp. 1–14, 2021.
- [33] R. You, S. Yao, H. Mamitsuka, and S. Zhu, “Deepgraphgo: Graph neural network for large-scale, multispecies protein function prediction,” *Bioinformatics*, vol. 37, no. Supplement_1, pp. i262–i271, 2021.
- [34] M. Kulmanov and R. Hoehndorf, “Deepgoplus: Improved protein function prediction from sequence,” *Bioinformatics*, vol. 36, no. 2, pp. 422–429, 2020.
- [35] A. Sureyya Rifaioglu, T. Doğan, M. Jesus Martin, R. Cetin-Atalay, and V. Atalay, “Deepred: Automated protein function prediction with multi-task feed-forward deep neural networks,” *Scientific reports*, vol. 9, no. 1, pp. 1–16, 2019.
- [36] W. Ning, Y. Guo, S. Lin, B. Mei, Y. Wu, P. Jiang, X. Tan, W. Zhang, G. Chen, D. Peng, *et al.*, “Drllps: A data resource of liquid–liquid phase separation in eukaryotes,” *Nucleic acids research*, vol. 48, no. D1, pp. D288–D295, 2020.
- [37] R. M. Vernon and J. D. Forman-Kay, “First-generation predictors of biological protein phase separation,” *Current opinion in structural biology*, vol. 58, pp. 88–96, 2019.

- [38] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, “Distributed representations of words and phrases and their compositionality,” in *Advances in neural information processing systems*, 2013, pp. 3111–3119.
- [39] J. H. Friedman, “Greedy function approximation: A gradient boosting machine,” *Annals of statistics*, pp. 1189–1232, 2001.
- [40] Q. Li, X. Peng, Y. Li, W. Tang, J. Zhu, J. Huang, Y. Qi, and Z. Zhang, “Llpsdb: A database of proteins undergoing liquid–liquid phase separation in vitro,” *Nucleic acids research*, vol. 48, no. D1, pp. D320–D327, 2020.
- [41] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne, “The protein data bank,” *Nucleic acids research*, vol. 28, no. 1, pp. 235–242, 2000.
- [42] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” *arXiv preprint arXiv:1609.02907*, 2016.
- [43] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, “Graph attention networks,” *stat*, vol. 1050, p. 20, 2017.
- [44] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, “Gated graph sequence neural networks,” *arXiv preprint arXiv:1511.05493*, 2015.
- [45] J. Lim, S. Ryu, K. Park, Y. J. Choe, J. Ham, and W. Y. Kim, “Predicting drug–target interaction using a novel graph neural network with 3d structure-embedded graph representation,” *Journal of chemical information and modeling*, vol. 59, no. 9, pp. 3981–3988, 2019.
- [46] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, “Convolutional networks on graphs for learning molecular fingerprints,” *Advances in neural information processing systems*, vol. 28, 2015.
- [47] A. Waterhouse, M. Bertoni, S. Bienert, G. Studer, G. Tauriello, R. Gumienny, F. T. Heer, T. A. P. de Beer, C. Rempfer, L. Bordoli, *et al.*, “Swiss-model: Homology modelling of protein structures and complexes,” *Nucleic acids research*, vol. 46, no. W1, W296–W303, 2018.
- [48] J. M. Dana, A. Gutmanas, N. Tyagi, G. Qi, C. O’Donovan, M. Martin, and S. Velankar, “Sifts: Updated structure integration with function, taxonomy and sequences resource allows 40-fold increase in coverage of structure-based annotations for proteins,” *Nucleic acids research*, vol. 47, no. D1, pp. D482–D489, 2019.
- [49] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Zidek, A. Potapenko, *et al.*, “Highly accurate protein structure prediction with alphafold,” *Nature*, vol. 596, no. 7873, pp. 583–589, 2021.
- [50] K. You, Q. Huang, C. Yu, B. Shen, C. Sevilla, M. Shi, H. Hermjakob, Y. Chen, and T. Li, “Phasepdb: A database of liquid–liquid phase separation related proteins,” *Nucleic acids research*, vol. 48, no. D1, pp. D354–D359, 2020.

- [51] B. Mészáros, G. Erdős, B. Szabó, É. Schád, Á. Tantos, R. Abukhairan, T. Horváth, N. Murvai, O. P. Kovács, M. Kovács, *et al.*, “Phasepro: The database of proteins driving liquid–liquid phase separation,” *Nucleic acids research*, vol. 48, no. D1, pp. D360–D367, 2020.
- [52] R. Pancsa, W. Vranken, and B. Mészáros, “Computational resources for identifying and describing proteins driving liquid–liquid phase separation,” *Briefings in Bioinformatics*, vol. 22, no. 5, bbaa408, 2021.
- [53] T. Laakko, A. Korkealaakso, G. Zhuoran, B. F. Yildirim, P. Batys, V. Liljeström, A. Hokkanen, Nonappa, I. Maasilta, M. Penttilä, A. Laukkanen, C. Södergård, and P. Mohammadi, “Hybrid biomimetic - de novo designed elastin-like polypeptides,” unpublished, 2023.
- [54] B. G. Kitchener, J. Wainwright, and A. J. Parsons, “A review of the principles of turbidity measurement,” *Progress in Physical Geography*, vol. 41, no. 5, pp. 620–642, 2017.
- [55] M. J. Abraham, T. Murtola, R. Schulz, S. Páll, J. C. Smith, B. Hess, and E. Lindahl, “Gromacs: High performance molecular simulations through multi-level parallelism from laptops to supercomputers,” *SoftwareX*, vol. 1, pp. 19–25, 2015.
- [56] S. J. Marrink, H. J. Risselada, S. Yefimov, D. P. Tieleman, and A. H. De Vries, “The martini force field: Coarse grained model for biomolecular simulations,” *The journal of physical chemistry B*, vol. 111, no. 27, pp. 7812–7824, 2007.
- [57] L. Monticelli, S. K. Kandasamy, X. Periole, R. G. Larson, D. P. Tieleman, and S.-J. Marrink, “The martini coarse-grained force field: Extension to proteins,” *Journal of chemical theory and computation*, vol. 4, no. 5, pp. 819–834, 2008.
- [58] P. C. Souza, R. Alessandri, J. Barnoud, S. Thallmair, I. Faustino, F. Grünewald, I. Patmanidis, H. Abdizadeh, B. M. Bruininks, T. A. Wassenaar, *et al.*, “Martini 3: A general purpose force field for coarse-grained molecular dynamics,” *Nature methods*, vol. 18, no. 4, pp. 382–388, 2021.
- [59] R. Stepto, T. Chang, P. Kratochvíl, M. Hess, K. Horie, T. Sato, and J. Vohlídal, “Definitions of terms relating to individual macromolecules, macromolecular assemblies, polymer solutions, and amorphous bulk polymers (iupac recommendations 2014),” *Pure and Applied Chemistry*, vol. 87, no. 1, pp. 71–120, 2015.
- [60] S. Chakraborty, R. Venkatramani, B. J. Rao, B. Asgeirsson, and A. M. Dandekar, “Protein structure quality assessment based on the distance profiles of consecutive backbone α atoms,” *F1000Research*, vol. 2, 2013.
- [61] C. Perez-Iratxeta, G. Palidwor, and M. A. Andrade-Navarro, “Towards completion of the earth’s proteome,” *EMBO reports*, vol. 8, no. 12, pp. 1135–1141, 2007.
- [62] K. A. Dill, S. B. Ozkan, M. S. Shell, and T. R. Weikl, “The protein folding problem,” *Annual review of biophysics*, vol. 37, p. 289, 2008.

- [63] J. Pereira, A. J. Simpkin, M. D. Hartmann, D. J. Rigden, R. M. Keegan, and A. N. Lupas, “High-accuracy protein structure prediction in casp14,” *Proteins: Structure, Function, and Bioinformatics*, vol. 89, no. 12, pp. 1687–1699, 2021.
- [64] M. Varadi, S. Anyango, M. Deshpande, S. Nair, C. Natassia, G. Yordanova, D. Yuan, O. Stroe, G. Wood, A. Laydon, *et al.*, “AlphaFold protein structure database: Massively expanding the structural coverage of protein-sequence space with high-accuracy models,” *Nucleic acids research*, vol. 50, no. D1, pp. D439–D444, 2022.
- [65] B. E. Suzek, H. Huang, P. McGarvey, R. Mazumder, and C. H. Wu, “Uniref: Comprehensive and non-redundant uniprot reference clusters,” *Bioinformatics*, vol. 23, no. 10, pp. 1282–1288, 2007.
- [66] H. Mei, Z. H. Liao, Y. Zhou, and S. Z. Li, “A new set of amino acid descriptors and its application in peptide qsars,” *Peptide Science: Original Research on Biomolecules*, vol. 80, no. 6, pp. 775–786, 2005.
- [67] E. Asgari and M. R. Mofrad, “Continuous distributed representation of biological sequences for deep proteomics and genomics,” *PloS one*, vol. 10, no. 11, e0141287, 2015.
- [68] E. C. Alley, G. Khimulya, S. Biswas, M. AlQuraishi, and G. M. Church, “Unified rational protein engineering with sequence-based deep representation learning,” *Nature methods*, vol. 16, no. 12, pp. 1315–1322, 2019.
- [69] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [70] T. Bepler and B. Berger, “Learning the protein language: Evolution, structure, and function,” *Cell systems*, vol. 12, no. 6, pp. 654–669, 2021.
- [71] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *nature*, vol. 323, no. 6088, pp. 533–536, 1986.
- [72] Y. Bengio, P. Simard, and P. Frasconi, “Learning long-term dependencies with gradient descent is difficult,” *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.
- [73] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, “Learning phrase representations using rnn encoder-decoder for statistical machine translation,” *arXiv preprint arXiv:1406.1078*, 2014.
- [74] Y. Goldberg, “A primer on neural network models for natural language processing,” *Journal of Artificial Intelligence Research*, vol. 57, pp. 345–420, 2016.
- [75] M. Gori, G. Monfardini, and F. Scarselli, “A new model for learning in graph domains,” in *Proceedings. 2005 IEEE international joint conference on neural networks*, vol. 2, 2005, pp. 729–734.
- [76] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, “The graph neural network model,” *IEEE transactions on neural networks*, vol. 20, no. 1, pp. 61–80, 2008.

- [77] M. Khamsi, “Introduction to metric fixed point theory,” in *Topics in Fixed Point Theory*, Springer, 2014, pp. 1–32.
- [78] J. Jimenez, S. Doerr, G. Martinez-Rosell, A. S. Rose, and G. De Fabritiis, “Deepsite: Protein-binding site predictor using 3d-convolutional neural networks,” *Bioinformatics*, vol. 33, no. 19, pp. 3036–3042, 2017.
- [79] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, “Spectral networks and locally connected networks on graphs,” *arXiv preprint arXiv:1312.6203*, 2013.
- [80] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, “Geometric deep learning: Going beyond euclidean data,” *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 18–42, 2017.
- [81] T. Bepler and B. Berger, “Learning protein sequence embeddings using information from structure,” *arXiv preprint arXiv:1902.08661*, 2019.
- [82] B. Song, Z. Li, X. Lin, J. Wang, T. Wang, and X. Fu, “Pretraining model for biological sequence data,” *Briefings in functional genomics*, vol. 20, no. 3, pp. 181–195, 2021.
- [83] A. Graves, “Generating sequences with recurrent neural networks,” *arXiv preprint arXiv:1308.0850*, 2013.
- [84] T. J. Hubbard, A. G. Murzin, S. E. Brenner, and C. Chothia, “Scop: A structural classification of proteins database,” *Nucleic acids research*, vol. 25, no. 1, pp. 236–239, 1997.
- [85] N. K. Fox, S. E. Brenner, and J.-M. Chandonia, “Scope: Structural classification of proteins—extended, integrating scop and astral data and classification of new structures,” *Nucleic acids research*, vol. 42, no. D1, pp. D304–D309, 2014.
- [86] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.
- [87] J. Bergstra and Y. Bengio, “Random search for hyper-parameter optimization.,” *Journal of machine learning research*, vol. 13, no. 2, 2012.