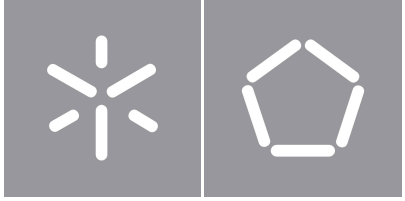


University of Minho
School of Engineering

Joana Oliveira Gonçalves

Exploration of Machine Learning and Deep Learning Approaches to Predicting Enzyme Cofactor Binding

december 2024



University of Minho
School of Engineering

Joana Oliveira Gonçalves

Exploration of Machine Learning and Deep Learning Approaches to Predicting Enzyme Cofactor Binding

Master's Dissertation in Bioinformatics

Dissertation supervised by
Óscar Manuel Lima Dias
Pedro Miguel Macedo Geada

december 2024

Copyright and Terms of Use for Third Party Work

This dissertation reports on academic work that can be used by third parties as long as the internationally accepted standards and good practices are respected concerning copyright and related rights.

This work can thereafter be used under the terms established in the license below.

Readers needing authorization conditions not provided for in the indicated licensing should contact the author through the RepositóriUM of the University of Minho.

License granted to users of this work:



CC BY-NC

`https://creativecommons.org/licenses/by-nc/4.0/` *[Esta licença permite que outros remisturem, adaptem e criem a partir do seu trabalho para fins não comerciais, e embora os novos trabalhos tenham de lhe atribuir o devido crédito e não possam ser usados para fins comerciais, eles não têm de licenciar esses trabalhos derivados ao abrigo dos mesmos termos.]*

Acknowledgements

First and foremost, I sincerely thank my supervisors, Oscar Dias and Pedro Geada, for their invaluable support and guidance throughout this dissertation. Their expertise was crucial, and I am deeply grateful for their time and dedication.

A special thanks to Emanuel, a PhD student, for his consistent help and encouragement during the process. His assistance was essential to the final outcome.

I also thank my friends and colleagues at BiSBi and "Biosala." The BiSBi group for their insights, and the "Biosala" group (my master's friends) for making this journey enjoyable with snacks, laughter, and card games.

A heartfelt thanks to my close friends from my Bachelor's—Gonçalo, Eduardo, Ruben, and Duarte—for always being there. I hope our friendship lasts a lifetime. And to Daniela, Bia, and Mariana, I'm grateful for your unwavering support and shared memories.

Furthermore, I am deeply grateful for my family's unwavering love and encouragement. Their belief in me has been a constant source of strength and motivation, lifting me and inspiring me to push through even the toughest challenges. I could not have reached this milestone without their support. I want to express my gratitude to my boyfriend for his patience and support throughout this journey.

This dissertation was supported by the Portuguese Foundation for Science and Technology (FCT) under the scope of the project CH4GREEN, with FCT reference 2022.05196.PTDC (DOI: 10.54499/2022.05196.PTDC).

FCT Fundação para a Ciência e a Tecnologia
MINISTÉRIO DA EDUCAÇÃO E CIÊNCIA

CH4GREEN

Statement of Integrity

I hereby declare having conducted this academic work with integrity.

I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

University of Minho, Braga, december 2024

Joana Oliveira Gonçalves

Abstract

Enzymes may depend on cofactors to carry out essential biochemical reactions, making accurate cofactor prediction crucial for understanding metabolism and enhancing Genome-Scale Metabolic (GSM) models. Traditional homology-based approaches often struggle to capture the complexity and diversity of enzyme-cofactor interactions across different species.

This dissertation presents a machine learning (ML) and deep learning (DL) workflow, employing Convolutional Neural Networks (CNNs), to predict enzyme cofactor binding based on protein sequence embeddings such as ESM2, SeqVec, and FlashProt. The dataset, consisting of 73,307 protein sequences annotated with 13 cofactors, was obtained from key databases including UniProt, BRENDA, Rhea, and ChEBI.

The CNN models, particularly those leveraging ESM2 embeddings, outperformed others in multi-label classification tasks, achieving high accuracy, low hamming loss, and high F1 scores. These models excelled in predicting cofactor binding for well-characterized organisms like *Escherichia coli*, where extensive data is available. However, predictions for less-studied organisms, such as *Helicobacter pylori* and *Synechocystis* sp. PCC 6803, showed reduced accuracy, underscoring the need for more comprehensive species-specific data.

In addition to the prediction workflow, a user-friendly web service was developed, allowing researchers to upload protein sequences and GSM models to receive cofactor predictions and update model reconstructions.

This workflow provides a valuable tool for advancing metabolic research, enabling more accurate cofactor predictions and improving GSM model reconstructions. Future work may focus on expanding the dataset, integrating additional data types like protein structures and enzyme activity, and further exploring advanced DL techniques such as transformers to enhance performance and generalizability across species.

Keywords Machine learning, Deep learning, Multi-label classification, Convolutional Neural Networks, Cofactors

Resumo

As enzimas podem depender de cofatores para realizar reações bioquímicas essenciais, tornando a previsão precisa dos cofatores fundamental para compreender o metabolismo e melhorar os modelos metabólicos à escala genómica (MEG). As abordagens tradicionais baseadas em homologia, não capturam a complexidade e diversidade das interações enzima-cofator entre diferentes espécies.

Esta dissertação apresenta um fluxo de trabalho de aprendizagem automática e aprendizagem profunda, utilizando redes neuronais convolucionais (RNCs), para prever a ligação de cofatores a partir de *embeddings* de sequências de proteínas, como ESM2, SeqVec e FlashProt. O conjunto de dados, composto por 73.307 sequências proteicas anotadas com 13 cofatores, foi compilado a partir de bases de dados como UniProt, BRENDA, Rhea e ChEBI.

Os modelos de RNCs, especialmente aqueles que utilizam embeddings de ESM2, destacaram-se nas tarefas de classificação multi-categoria, alcançando alta precisão, baixa perda de hamming loss e valores de pontuação F1 elevados. Estes modelos mostraram-se particularmente eficazes na previsão de cofatores para organismos bem caracterizados, como *Escherichia coli*. No entanto, as previsões para organismos menos estudados, como *Helicobacter pylori* e *Synechocystis* sp. PCC 6803, revelaram menor precisão, sublinhando a necessidade de dados mais abrangentes e específicos de outras espécies.

Para além do fluxo de trabalho, foi também desenvolvido um serviço disponivel na *internet*, que permite aos investigadores fazer upload de sequências de proteínas e modelos MEG, obtendo previsões de cofatores e atualizações nas reconstruções dos modelos.

O fluxo de trabalho oferece uma ferramenta valiosa para o avanço da investigação metabólica, permitindo previsões de cofatores mais precisas, e para melhorar as reconstruções de modelos MEG. Trabalhos futuros poderão focar-se na expansão do conjunto de dados, integrando novos tipos de dados, como estruturas proteicas e atividade enzimática, bem como na exploração de técnicas avançadas de aprendizagem profunda, como *transformers*, para melhorar o desempenho e a generalização entre espécies.

Palavras-chave Aprendizagem de máquina, Aprendizagem profunda, Classificação multi-categoria, Redes Neuronais Convolucionais, Cofatores

Contents

1 Introduction **1**

1.1 Context and Motivation 1

1.2 Research Objectives 1

1.3 Document Organization 2

2 State of the Art **3**

2.1 Systems Biology concepts 3

2.1.1 Genome-Scale Metabolic Model 5

2.2 Enzymes and Cofactors 8

2.2.1 Enzymes 8

2.2.2 Cofactors 10

2.2.3 Databases for Enzymatic and Cofactor Data 13

2.2.4 Methods for prediction of cofactor specificity 14

2.3 Machine Learning 15

2.3.1 Unsupervised Learning 16

2.3.2 Supervised Learning: Models and Algorithms 17

2.3.3 Supervised Learning: ML workflow 21

2.3.4 Python libraries for machine and deep learning 28

2.3.5 Multi-Label Classification Models 29

2.3.6 Multi-Label Performance Metrics 31

2.3.7 Challenges in Multi-Label Classification 34

2.4 Deep Learning 35

2.5 State-of-the-Art Tools for Workflow Automation 42

2.5.1 Docker 42

2.5.2 NextFlow 43

2.5.3 Flask . . . 43

3 Methodology **45**

3.1 Data Extration . . . 45
3.2 Dataset Construction . . . 46
3.3 Data Preprocessing . . . 47
3.4 Machine Learning Models Construction . . . 48
3.5 Deep Learning Models Construction . . . 49
3.6 Cofactor-Predicton Workflow . . . 52
3.7 Web Service for Cofactor Prediction . . . 53
3.8 Case Studies . . . 54

4 Results and Discussion **55**

4.1 Dataset Overview and Characterization . . . 55
4.2 Performance Evaluation of Machine Learning Models . . . 57
4.2.1 ESM2 Embeddings . . . 60
4.2.2 SeqVec Embeddings . . . 60
4.2.3 FlasProt Embeddings . . . 61
4.2.4 Discussion . . . 62
4.3 Performance Evaluation of Deep Learning Models . . . 63
4.4 Case Studies . . . 65
4.5 Web Service Functionality . . . 70

5 Conclusions and future work **71**

5.1 Conclusions . . . 71
5.2 Prospect for future work . . . 72

I Appendices **90**

A Details of results **91**

List of Figures

1 Random Forest Structure. 20
2 Artificial Neural Network Architecture. 21
3 Machine Learning workflow. 21
4 Performance metrics for Multi-Label Classification (MLC). 31
5 Deep Neural Network Structure. 36
6 Convolutional Neural Network Architecture. 37
7 Recurrent Neural Network Architecture. 38
8 Autoencoder Architecture. 39
9 Transformer Architecture. 41
10 Convolutional Neural Network (CNN) Architecture. 51
11 Cofactor-Prediction Workflow Pipeline 52
12 Cofactor Distribution in the Dataset 55
13 Cofactor Relationships in Protein Sequences 57
14 F1 Score, Precision, and Recall at Varying Thresholds for ESM2 embeddings. 64
15 F1 Score, Precision, and Recall at Varying Thresholds 91
16 Principal Component Analysis (PCA) of Protein Embeddings for Different Cofactors . . . 92

List of Tables

1 List of genome-scale metabolic reconstruction tools 6

2 Summary of the universally essential cofactors. 12

4 Classification Metrics. 26

5 Overview of Python libraries for ML/DL. 28

6 ChEBI and Rhea Reaction IDs for Cofactor Retrieval 47

7 Summary of Machine Learning (ML) models employed for MLC and the specific hyperparameters that were fine-tuned during model optimization. 49

8 Label Counts in Train, Validation, and Test Sets 58

9 Execution Times for Generating Embeddings 59

10 Performance Evaluation of Machine Learning Models Using ESM2 Embeddings. 60

11 Performance Evaluation of Machine Learning Models Using SeqVec Embeddings. 61

12 Performance Evaluation of Machine Learning Models Using FlasProt Embeddings. . . . 61

13 Performance Evaluation of Convolutional Neural Networks (CNN) Models (Before Hyperparameter Optimization) 63

14 Performance Evaluation of CNN Models (After Hyperparameter Optimizations) 63

15 Best trial results and hyperparameters for each model 64

16 Detailed Study of Three GSM Models Using Literature and Databases 66

17 Summary of case studies showing the total number of reactions, the number of reactions involving cofactors, reactions with sequence data, and the number of missing sequences. 93

18 GSM Model Distribution of Predicted Genes Matching and Mismatching Reaction Cofactor Usage 98

Acronyms

R^2 Coefficient of Determination.

E. coli *Escherichia coli*.

S. cerevisiae *Saccharomyces cerevisiae*.

ACP Acyl Carrier Protein.

AI Artificial Intelligence.

AIC Akaike Information Criterion.

ANN Artificial Neural Network.

ANOVA Analysis of variance.

ATP adenosine triphosphate.

AUC Area Under the Curve.

BERT Bidirectional Encoder Representations from Transformers.

BIC Bayesian Information Criterion.

BR BinaryRelevance.

BRENDA Braunschweig Enzyme Database.

CC ClassifierChain.

ChEBI Chemical Entities of Biological Interest.

CNN Convolutional Neural Networks.

CoA Coenzyme A.

COBRA COnstraint-Based Reconstruction and Analysis.

DA Denoising Autoencoder.

DL Deep Learning.

DNA Deoxyribonucleic Acid.

DNN Deep Neural Network.

DT Decision Tree.

EC Enzyme Commission.

FAD Flavin adenine dinucleotide.

FBA Flux Balance Analysis.

FMN Flavin Mononucleotide.

FN False Negative.

FP False Positive.

GPT Generative Pre-trained Transformer.

GPUs Graphics Processing Units.

GRU Gated Recurrent Unit.

GSH Glutatione.

GSM Genome-Scale Metabolic.

GUI Graphical User Interface.

JSON JavaScript Object Notation.

KEGG Kyoto Encyclopedia of Genes and Genomes.

KNN K-Nearest Neighbors.

Lasso Least absolute shrinkage and selection operator.

LDA Linear Discriminant Analysis.

LgR Logistic Regression.

LOOCV Leave-One-Out Cross-Validation.

LP LabelPowerset.

LR Linear Regression.

LSTM Long Short-Term Memory.

MAE Mean Absolute Error.

ML Machine Learning.

MLC Multi-Label Classification.

MLkNN Multi-label k-Nearest Neighbour.

MSE Mean Squared Error.

MT-DT Multi-label decision tree algorithm.

NAD(H) Nicotinamide Adenine Dinucleotide.

NADP(H) Nicotinamide Adenine Dinucleotide Phosphate.

NLP Natural Language Processing.

NMR Nuclear Magnetic Resonance.

PCA Principal Component Analysis.

PDB Protein Data Bank.

pFBA Parsimonious Flux Balance Analysis.

PLP Pyridoxal 5' phosphate.

Rank-SVM Ranking Support Vector Machine.

RF Random Forest.

RMSE Root Mean Squared Error.

RNA Ribonucleic Acid.

RNN Recurrent Neural Networks.

ROC Receiver Operating Characteristic.

SAE Stacked Autoencoder.

SAM S-adenosyl-methionine.

SBML Systems Biology Markup Language.

SVD Singular Value Decomposition.

SVM Support Vector Machines.

THF Tetrahydrofolate.

TN True Negative.

TP True Positive.

t-SNE t-distributed Stochastic Neighbor Embedding.

THMPP Thiamine diphosphate.

TSV Tab-Separated Values.

UMAP Uniform Manifold Approximation and Projection.

UniProt Universal Protein Resource Knowledgebase.

VAE Variational Autoencoder.

Chapter 1

Introduction

1.1 Context and Motivation

Enzymes are vital biological catalysts that facilitate the biochemical processes in living organisms, driving species-specific metabolic networks and significantly accelerating reactions necessary for cellular function (1). Many enzymes depend on cofactors like Nicotinamide Adenine Dinucleotide (NAD(H)) and Nicotinamide Adenine Dinucleotide Phosphate (NADP(H)), which are essential in cellular metabolism and biochemical pathways (2). However, predicting enzyme cofactor specificity is challenging, as traditional methods such as homology-based searches often fail to account for variations across different species or subcellular environments.

Inaccuracies in predicting cofactor usage can seriously affect the development of Genome-Scale Metabolic (GSM) models, potentially leading to errors in metabolic engineering and strain design by misidentifying key metabolic pathways and reactions (3; 4). While some GSM tools address cofactor specificity indirectly through homology-based techniques, these approaches often fall short in capturing the complex interactions between enzymes and cofactors across diverse biological contexts.

ML and Deep Learning (DL) offer promising alternatives, providing more accurate predictive capabilities than traditional methods (5). Recent advancements in these fields have shown great potential in improving enzyme cofactor specificity predictions (6; 7; 8; 9; 10). By utilizing the power of ML and DL, we can achieve more precise functional annotations, leading to enhanced GSM model reconstructions and opening new possibilities in metabolic engineering.

1.2 Research Objectives

The main objective of this project is to develop accurate machine learning (ML) and deep learning (DL) models for predicting enzyme cofactor specificity. These models will use supervised learning techniques to

identify the most effective approach. The scientific and technical objectives of this research are as follows:

- Review relevant literature on machine learning and deep learning methods for enzyme cofactor prediction and metabolic modeling.
- Design and compare data preprocessing workflows, using features to evaluate their impact on model performance.
- Develop and evaluate machine learning and deep learning models, including 1D CNN architectures, using metrics like accuracy, F1-score, and hamming loss.
- Applying the developed models to reactions involving cofactors within genome-scale metabolic (GSM) models for different organisms, such as microalgae and cyanobacteria, to test the models' ability to predict enzyme-cofactor interactions across species.
- Integrate the best models into the merlin tool to improve GSM model reconstructions, improving cofactor specificity predictions.

1.3 Document Organization

- **Chapter 1: Introduction** – Introduces the context, motivation, and challenges of predicting enzyme cofactor specificity, along with the research objectives and the role of machine learning and deep learning models in addressing these challenges.
- **Chapter 2: State of the Art** – Reviews key literature on systems biology, enzyme-cofactor interactions, and genome-scale metabolic models, and covers machine learning and deep learning basics with a focus on multi-label classification.
- **Chapter 3: Methodology** – Details the development of the cofactor prediction pipeline, including data extraction, preprocessing, model design, hyperparameter optimization, and the development of a web service for pipeline use.
- **Chapter 4: Results and Discussion** – Presents the performance results of the machine learning and deep learning models, evaluated through metrics like F1-score, Hamming loss, and accuracy, and discusses case studies across different organisms.
- **Chapter 5: Conclusions and Future Work** – Summarizes the findings, contributions, limitations, and proposes future research directions.

Chapter 2

State of the Art

2.1 Systems Biology concepts

For centuries, humans have inadvertently employed microorganisms in processes like fermentation, unlocking their potential to alter raw materials. With Pasteur's groundbreaking work, nearly two centuries ago, a link between microbes, food decay, and fermentation was established, unveiling the immense potential inherent to microbial metabolism (11).

This newfound understanding of microbial metabolism stimulates the development of various applications, ranging from food preservation to the synthesis of antibiotics and synthetic chemicals. However, a significant breakthrough in modifying microbial metabolism emerged with the discovery and function comprehension of Deoxyribonucleic Acid (DNA), birthing the field of metabolic engineering (12).

Metabolic engineering is the process of changing cellular activities by manipulating enzymatic, transport, and regulatory functions using recombinant DNA technology (13). At its core, metabolic engineering seeks to optimize the biosynthesis of high-value compounds by modifying genes to alter metabolic pathways. These genetic modifications prompt cells to redirect metabolic processes, enabling the creation of microorganisms acting as biocatalysts, yielding biochemicals, biofuels, and biopharmaceuticals (14; 15).

Initially relying on random mutagenesis, the field transitioned to a more systematic approach propelled by advancements in mathematical tools, computational tools, and sequencing techniques. This methodical approach integrated gene manipulations with predicted phenotypes, allowing for a comprehensive view of metabolic landscapes (16).

Systems biology, a term first used by Mihajlo Mesarovic in 1968 (17), is about fully understanding complex biological systems. It brings together biology, math, computer science, physics, and engineering to study how different parts of a biological system interact. This approach helps us see the bigger picture, from tiny molecules to whole organisms.

This field is important for advancing medicine, biotechnology, and environmental studies. It helps

us to understand diseases better, develop new medicines, and create advanced biological systems. Systems biology is unique because it combines different sciences to fully understand how life works, from small details to the big picture. This understanding leads to discoveries that can improve health and the environment (18).

The development of systems biology can be delineated into three distinct phases. The initial phase traces back to 1953 when molecular biologists focused on understanding gene structure and functionality, leading to genetic engineering and the decoding of the human genome at the onset of the 21st century (12).

The post-genomic era signified a paradigm shift in molecular biology's objective: transitioning from a focus on single molecules to elucidating how complex molecular pathways and networks underpin biological structure and function. This shift marked the dawn of systems molecular biology. The second phase encompassed the evolution of systems-mathematical biology, rooted in the principles of general systems theory and the exploration of nonlinear dynamics within living organisms. This phase delved into the broader systemic aspects of biological entities, emphasizing holistic approaches and dynamic interactions.

Subsequently, the third phase emerged from the convergence of molecular systems biology and systems-mathematical biology, leading to advanced systems-based medicine, biotechnology, and drug discovery (19; 18).

Around twenty years ago, systems biology was defined as the utilization of systematic genomics, proteomics, and metabolomics technologies to generate data for developing models of biological systems (20).

Omics Technologies (Genomics, Transcriptomics, Proteomics, Metabolomics) provide comprehensive insights into an organism's genetic material, gene expression, proteins, and metabolites, respectively. These technologies aid in decoding biological structures, functions, and interactions across different biological scales, serving as the foundational data for modeling complex biological systems (21).

Traditionally, experimental methods for determining gene function were laborious. However, high-throughput sequencing technologies in the post-genomic era have led to a surge in sequenced genomes. This deluge of sequences has made it impractical to determine the function of every gene or protein experimentally. Thus, computational approaches and bioinformatics tools have become indispensable in predicting or inferring gene activities. While computational predictions expedite gene function annotation, experimental validation remains essential to ensure accuracy and reliability (22).

Integrating computational predictions with experimental evidence maintains the quality and reliability of gene annotations, providing a robust foundation for further research in systems biology.

2.1.1 Genome-Scale Metabolic Model

The reconstruction of GSM models is one of the most significant modeling approaches for studying biological systems. The first GSM model, developed for *Haemophilus influenzae* (23) in 1999, marked the beginning of using these models as essential tools in biological research.

GSM models are advanced computer models that represent an organism's entire metabolic network. These models are carefully created based on an organism's DNA, providing a detailed view of the complex biochemical processes that happen in a cell. GSM models focus on the organism's genetic sequence, which is key to understanding how various metabolic reactions occur and how different substances move across cell membranes. By capturing the entire range of a cell's metabolism, GSM models provide valuable insights into cellular functions at a molecular level (24).

Currently, over 100 detailed GSM models are covering a wide range of organisms. These include simple ones like the bacteria *Escherichia coli* (*E. coli*) (25) and the yeast *Saccharomyces cerevisiae* (*S. cerevisiae*) (26), as well as more complex ones like human (27) and plant cells (28). High-quality GSM models are crucial for simulating how different organisms might react under various conditions, making them very useful in areas like metabolic engineering and drug discovery.

However, the accuracy of GSM models depends on the quality of the data used to make them. So, it is very important to carefully choose and put together experimental data from different sources to create models that are both precise and reliable. Also, it is necessary to keep checking and updating these models as new metabolic information becomes available.

The reconstruction of a genome-scale metabolic network follows four well-delineated steps (29). It starts with a deep dive into the genome. Here, every gene and Ribonucleic Acid (RNA) type are identified, a step known as a structural annotation. Then, each gene is given a role, especially those crucial for enzymes and transporters, in what's called functional annotation.

After comprehensively understanding the genome, the subsequent step involves the development of the metabolic network. This step requires the meticulous construction of a network encompassing enzymatic, spontaneous, and transport reactions. It is imperative to meticulously refine this network to maintain the equilibrium of reaction masses, validate reaction reversibility, and address any discrepancies. Following its construction, the network is subjected to an exhaustive validation process. It is transformed into a computational model and its precision is assessed through simulation methodologies such as Flux Balance Analysis (FBA). This testing compares the model's predictions with real-world results to confirm its reliability.

FBA is one of the methods used to simulate GSM models; it allows predicting the behavior of these

networks by calculating the fluxes of biochemical reactions that lead to a pre-defined objective, considering metabolite's stoichiometry and thermodynamic constraints. The FBA approach is based on the assumption of a (pseudo) steady-state growth. This means that the studied system is assumed to be in a state of constant growth, where the rate of mass entering the system is equal to the rate of mass leaving the system (30).

This method, based on linear programming, involves a stoichiometric matrix (S) and a flux vector (v) to optimize an objective function (Z) and can be formulated as:

$$\begin{aligned} &\text{Maximize } Z \\ &\text{subject to } S \cdot v = 0 \\ &\alpha_j \leq v_j \leq \beta_j \quad j = 1, \ldots, N \end{aligned} \tag{2.1}$$

The objective function can be selected based on the specific optimization goal, such as maximizing cell growth, adenosine triphosphate (ATP) production, the production of a target metabolite, or minimizing substrate uptake. However, FBA provides only one solution, although other optimal flux distributions may exist. To overcome this, Parsimonious Flux Balance Analysis (pFBA) can be used, which optimizes the objective function Z and minimizes the total sum of the flux through the model. This approach provides a unique optimal solution, making it a more reliable tool for studying metabolic networks (31).

Acknowledging the labor-intensive nature of GSM model reconstruction, a range of specialized tools has emerged to simplify this intricate process. These tools are specifically designed to facilitate GSM model reconstruction, and a concise overview of them is presented in Table 1.

Table 1: List of genome-scale metabolic reconstruction tools

Software	**Language**	**Databases**	**Interface**	**Reference**
COBRA Toolbox	Matlab	KEGG, BiGG	Command line	(32; 33)
CarveMe	Python	BIGG	Command line	(34)
merlin	Java	KEGG, MetaCyc, UniProtKB, TCDB	GUI	(35)
RAVEN 2.0	Matlab	KEGG, Published models	Command line	(36)
AuReMe	Python	BIGG, MetaCyc, Model SEED	Command line	(37)
AutoKEGGRec	Matlab	KEGG	Command line	(38)
MetaDraft	Python	BiGG Models	GUI	(39)
FAME	Python	KEGG	GUI	(40)
MEMOSys	Java	ChEBI, KEGG	Web	(41)
GEMSiRV	Java	KEGG, BiGG, MetaCyc, Model SEED	Web	(42)
Pathway Tools	Python	PGDB, MetaCyc	GUI, Web	(43)
CoReCO	Python, R, Perl	KEGG	Command line	(44)
rBioNet	Matlab	KEGG	GUI	(45)
SuBliMinal Toolbox	Java	KEGG, MetaCyc	Command line	(46)
Raven	Matlab	KEGG, MetaCyc	Command line	(36)

COBRA Toolbox and RAVEN 2.0 are software tools developed in MATLAB. Both tools use databases such as Kyoto Encyclopedia of Genes and Genomes (KEGG) and BiGG, and are primarily operated through command-line interaction. These tools are best suited for users who prefer script-based interactions.

Web-based tools like MEMOSys and GEMSiRV offer remote and collaborative capabilities. MEMOSys is Java-based and utilizes databases such as Chemical Entities of Biological Interest (ChEBI) and KEGG, whereas GEMSiRV, also Java-based, extends its database usage to BiGG, MetaCyc, and Model SEED.

CarveMe and AuReMe offer command-line operations, ensuring a hands-on coding experience. CarveMe uses the BIGG database for data interaction, while AuReMe integrates with BIGG, MetaCyc, and Model SEED, providing a comprehensive suite of resources for users who appreciate the versatility of Python.

MetaDraft and FAME stand out as Graphical User Interface (GUI)-based interaction tools. MetaDraft is Python-based and linked with BiGG Models, while FAME is also Python-based but uses KEGG. Both cater to users who prioritize a graphical interface.

On the other hand, *merlin* is a software tool that developed in a Java-based environment. *merlin* offers a GUI and draws from a range of databases such as KEGG, MetaCyc, UniProtKB, and TCDB. This makes it ideal for users without coding expertise, as this user-friendly Java application provides an interface for GSM reconstruction. It uses algorithms and data sources to build high-quality metabolic models, retrieving data from biochemical databases and BiGG Models. *merlin* integrates compartments information, generates transport reactions, and includes plugins for visualizing, analyzing, and validating resulting models. After building a model, it was refined with real-world data and tested by comparing its predicted growth against actual experimental results. This process ensures the model's reliability for biological research.

The most commonly used methods for simulating metabolic models are the COnstraint-Based Reconstruction and Analysis (COBRA) methods. Although the COBRA Toolbox for MATLAB is a well-established tool in this domain, COBRApy, a Python package, provides an attractive alternative for model reconstruction and analysis. It allows users to access a wider range of COBRA methods, many of which are also available in the COBRA Toolbox for MATLAB (47).

Besides the tools mentioned earlier, OptFlux software can also analyze and simulate GSM models. OptFlux is known for its user-friendly interface, making it ideal for experimentalists who want to use their experimental data for innovative applications (48).

Most genome-scale metabolic models are focused on the analysis of steady-state intracellular metabolic fluxes, where the metabolite concentrations within cells are considered constant over time (49). Although constrained by design, constraint-based models, another term for these genome-scale metabolic models, have found wide application across diverse fields. These include advancing biomedical research, partic-

ularly in identifying novel targets for cancer drugs and antibiotics, optimizing microbial strains for biofuel production in industrial settings, and evaluating the environmental roles and bioremediation capabilities of microorganisms (50).

Despite the progress made in developing tools for GSM models reconstruction, these models still require manual curation to increase their reliability and consistency. This is crucial for accurately predicting experimental data through *in silico* methods.

A key obstacle in GSM model reconstruction is the precise functional annotation of enzymes, particularly regarding cofactor specificity. Currently, tools like *merlin* cannot differentiate cofactor binding when an enzyme can accept multiple cofactors, especially cofactors with similar structures like NAD(H) and NADP(H). This limitation is exemplified by the shortcomings of the Enzyme Commission (EC) numbering system, which can misrepresent the actual biochemical functions of enzymes, leading to potential inaccuracies in GSMs and impacting their utility in applications such as metabolic engineering.

2.2 Enzymes and Cofactors

2.2.1 Enzymes

Enzymes are indispensable proteins ubiquitous in all living organisms that catalyze biochemical reactions *in vivo* and play a crucial role in regulating biological processes. These proteins are composed of 20 naturally occurring amino acids. By utilizing these basic building blocks, enzymes enable chemical events within cells, including metabolism, DNA replication, and protein synthesis. Their exceptional stereoselectivity and regioselectivity, along with phenomenal yields, allow these reactions to occur under physiological conditions (around pH 7, 1 atm, and in aqueous solution). This is a highly desired objective for synthetic organic chemical processes, which frequently necessitate harsh conditions to execute the same chemistry (51; 52).

A striking example of the abundance and diversity of enzymes can be seen in the data from Swiss-Prot, which reports that 282,966 of the 570,157 manually annotated proteins are enzymes retrieved on February 12, 2024. This significant number underscores the crucial role enzymes play in biological systems. To manage this vast array of enzymes, the EC system offers a structured classification scheme. Each enzyme is assigned a unique EC number, composed of four digits, which serves as an identifier of its specific function.

The classification of enzymes begins with the first digit of the EC number, which indicates the broad enzyme class based on the chemical reaction it catalyzes. In 1961, the International Union of Biochem-

istry recognized six enzyme classes: EC 1 for Oxidoreductases, which are involved in oxidation-reduction reactions; EC 2 for Transferases, responsible for transferring functional groups between molecules; EC 3 for Hydrolases, which cleave bonds through hydrolysis; EC 4 for Lyases, known for breaking various chemical bonds by means other than hydrolysis and oxidation; EC 5 for Isomerases, which catalyze structural rearrangements within a molecule; and EC 6 for Ligases, involved in the joining of two molecules. In 2018, a seventh EC number was added for translocases (53).

The classification is further refined with the second digit, which specifies the subclass and provides additional insights into the enzyme's function within its main class by designing the type of chemical bond the enzyme acts on. The third digit takes this a step further, indicating a sub-subclass, offering a more detailed understanding of the enzyme's substrate(s). The fourth digit is a serial number, attributed sequentially to discovered enzymes.

The EC number system is the primary method used to classify enzymes' functions, but it has several limitations. One of the issues with this system is that it often attributes multiple reactions to a single EC number. This can lead to a misconception that an enzyme is capable of catalyzing all associated reactions in all organisms. Another limitation is that enzymes from different organisms that share the same EC number may have varying substrate specificities. This variability is due to differences in their molecular recognition sets, which are not always accurately captured by the EC classification. These limitations highlight the challenges in accurately determining enzyme functionality, especially when dealing with diverse biological systems (54).

In 1894, Emil Fischer proposed an explanation for the way enzymes work by introducing a model called the lock-and-key model. This model suggests that the substrate, also known as the key, fits perfectly into the enzyme, also known as the lock. According to this model, the active site of the enzyme, which is a specialized area on its surface, binds to the substrate in a precise manner, producing an enzyme-substrate complex. This binding is highly specific, as the active site matches the substrate's shape and chemical properties perfectly. Once the enzyme and substrate are bound together, the enzyme undergoes changes that help convert the substrate into a product. After the reaction, the product is released, allowing the enzyme to begin another catalytic cycle (51).

The problem with this hypothesis was addressed by focusing on the specificity of enzymes rather than their efficiency. It did not account for the ability of enzymes to adapt to different shapes through induced fit, the potential interference of molecules like water, or the possibility of some substances blocking enzyme function without fitting into the active site, known as noncompetitive inhibition. Additionally, it cannot explain why some substrate analogs, which appear to be a perfect match, do not react (55).

The Induced Fit Theory, introduced by Daniel Koshland in 1958 (56), addresses the limitations of the lock-and-key model by proposing that the enzyme's active site is not a rigid structure but is flexible and can adjust its shape to accommodate the substrate. According to this theory, the binding of a substrate leads to conformational changes in the enzyme, allowing for a more precise fit and optimal alignment of catalytic groups necessary for the chemical reaction. This dynamic adaptation, much like a hand adjusting to fit a glove, ensures that enzymes are not only highly specific but also efficient in their catalytic activity.

The Induced Fit Theory better explains the observed enzyme behavior, including the phenomenon of noncompetitive inhibition and the inability of some substrate analogs to react despite their structural similarity to the actual substrate. It underscores the complexity and dynamic nature of enzyme-substrate interactions, providing a more accurate and comprehensive understanding of enzyme catalysis (55).

2.2.2 Cofactors

Cofactors are crucial in enzymatic reactions by acting as small molecules that transfer electrons, protons, or functional groups between substrates. Their functions are diverse, ranging from group and electron transfer to structural stabilization in enzymes. The interaction between an enzyme and its cofactor at the enzyme's active site, where catalysis occurs, is crucial for the specificity and efficiency of the reaction.

Cofactors are metal ions, which typically contribute to the enzyme's active site by providing structural stability or directly participating in catalytic activities, as well as tiny organic molecules that bind to enzymes and facilitate the transfer of chemical groups during reactions (57).

Universally Essential Cofactors

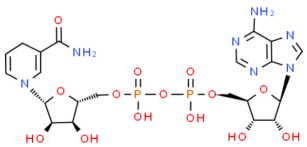
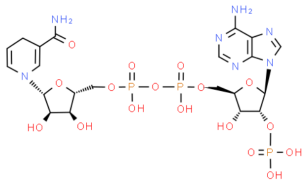
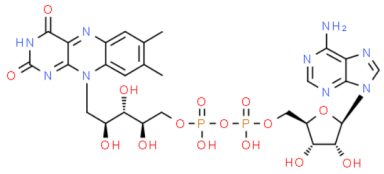
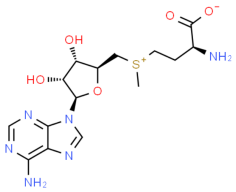
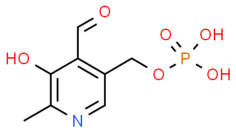
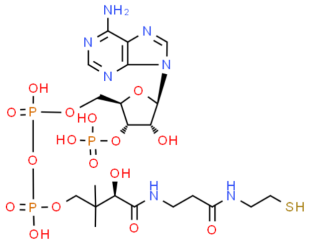
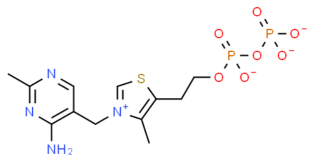
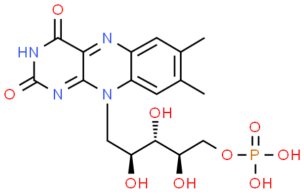
Table 2 presents the universally essential cofactors for correct cell metabolism: NAD(H), NADP(H), S-adenosyl-methionine (SAM), Flavin adenine dinucleotide (FAD), Pyridoxal 5' phosphate (PLP), Coenzyme A (CoA), Thiamine diphosphate (THMPP) and Flavin Mononucleotide (FMN) (4).

The nicotinamide cofactors, NAD(H) and NADP(H), play a crucial role in cellular metabolism. Although they share a similar structure, they differ by a phosphate group on the adenosyl moiety. Despite their structural resemblance, NAD(H) and NADP(H) have distinct roles in metabolic processes. NAD(H) mainly drives catabolic processes and ATP production, while NADP(H) is essential in anabolic processes such as lipid biosynthesis (58). Their involvement in oxidoreductase-mediated electron transfer reactions further emphasizes their significance in various metabolic functions, highlighting the necessity of these cofactors in maintaining cellular metabolic balance and efficiency (59).

SAM is a crucial molecule that contributes significantly to methylation reactions, which help to regulate

gene expression and cellular signaling (60). FAD and FMN are coenzymes that play integral roles in cellular respiration as components of the electron transport chain, which is important for ATP production. FMN operates in Complex I, while FAD is part of Complex II in the Krebs cycle. These coenzymes also play roles in detoxification and the metabolism of fatty and amino acids (61).

Table 2: Summary of the universally essential cofactors for correct cell metabolism.

Cofactor	Full name	Functional role	Chemical structure
NAD(H)	Nicotinamide Adenine Dinucleotide	Transport and transfer of hydride groups	
NADP(H)	Nicotinamide Adenine Dinucleotide Phosphate	Transport and transfer of hydride groups	
FAD(H)	Flavin adenine dinucleotide	Electron transfer, radical and photoreceptor-induced reactions	
SAM	S-adenosyl-methionine	Universal methyl donor; generator of deoxyadenosyl radicals	
P5P	Pyridoxal 5' phosphate	Electrophilic catalyst	
CoA	Coenzyme A	Transport and transfer of acyl groups	
THMPP	Thiamine diphosphate	Making and breaking bonds between C and S, O, H, and N atoms, notably C-C bonds	
FMN	Flavin Mononucleotide	Electron transfer, radical and photoreceptor-induced reactions	

PLP is the active form of vitamin B6 and is essential for amino acid metabolism, impacting protein synthesis and degradation. It also participates in the biosynthesis of nucleic acids and neurotransmitters,

which affects cell replication and response to environmental signals. The CoA, especially in its acetyl-CoA form, is central to the citric acid cycle and plays a pivotal role in energy production and lipid metabolism (62).

THMPP is the biologically active form of vitamin B1. It is a crucial coenzyme synthesized de novo by various organisms, including bacteria and plants. THMPP—dependent enzymes facilitate the formation and cleavage of key chemical bonds in metabolic reactions. THMPP is necessary for the Pyruvate Dehydrogenase and α-Ketoglutarate Dehydrogenase complexes in the Krebs cycle. It is also essential for transketolase in the pentose phosphate pathway, which is vital for nucleotide biosynthesis and cellular redox balance (63).

The proper functioning of each cofactor is a critical aspect of maintaining the cell's homeostasis and ensuring optimal metabolic processes. Microorganisms are known to regulate the levels of these cofactors with high precision to facilitate efficient metabolism. Therefore, a comprehensive understanding of the interaction between enzymes and cofactors is essential to accurately reconstructing metabolic pathways within an organism.

2.2.3 Databases for Enzymatic and Cofactor Data

Enzyme functional annotations can be found in various databases containing a wide range of information. For example, Braunschweig Enzyme Database (BRENDA) (64) categorizes enzymes by EC numbers and also provides insights into enzyme structure, stability, ligand interactions, literature references, organism-specific data, and molecular and functional properties.

Another resource, Universal Protein Resource Knowledgebase (UniProt) (The UniProt Consortium, 2007), consists of two sections: UniProtKB/SwissProt, containing expert-curated protein information, and UniProtKB/TrEMBL, providing information about enzyme function, sequence, domain structure, subcellular location, post-translational modifications, functionally characterized variants, cofactors, and ligands.

The KEGG database offers extensive details on genomic sequences and biological pathways, while MetaCyc (65) provides enzyme properties specific to various species, including substrate specificity, cofactor requirements, activators, inhibitors, and references to primary literature sources.

The SABIO-RK database (66) focuses on the complex kinetics of biochemical reactions and their kinetic properties. It contains a detailed, structured compilation of data, including information about the reaction participants (substrates, products), modifiers (inhibitors, activators, cofactors), as well as detailed information about the proteins catalyzing the reactions (e.g., EC enzyme classification, UniProtKB accession numbers, protein complex composition of the active enzyme, isoenzymes, wildtype/mutant information)

and their biological source (organism, tissue/cell type, cell location). SABIO-RK can be accessed by web interface or automated web services.

Additionally, the Protein Data Bank (PDB) is a valuable resource for protein structural information. Recently, a curated list of small molecules from the CoFactor database was used to add cofactor and cofactor-like molecules to the ligands and environments section of the PDBe (67).

2.2.4 Methods for prediction of cofactor specificity

Researchers initially set about addressing cofactor specificity concerns by focusing on the nucleotide-binding domain of enzymes that bind NAD(P)(H). This focus was driven by the widespread use and significant importance of NAD(P)(H)-binding enzymes in various chemical processes. Scientists conducted extensive examinations of these enzymes, aiming to identify differentiating structural motifs that could shed light on their specificities.

One of the most frequently observed structural motifs in NAD(P)H binding enzymes is the Rossmann fold, which was first identified by Rao and Rossmann in 1973 (68). However, identifying the residues responsible for NAD(H) and NADP(H) specificity based on these motifs has been challenging due to the variability of the present residues (69).

Numerous studies in the field of metabolic engineering have focused on altering the cofactor specificity of enzymes that interact with NAD(P)(H). One common approach is site-directed mutagenesis to modify the enzymes, aiming to switch their cofactor preference. However, these experiments have revealed that changing cofactor specificity effectively typically requires multiple simultaneous mutations, indicating the complexity of enzyme-cofactor interactions and the sophisticated level of intervention needed for such modifications (70; 71; 72). Unfortunately, these techniques are often labor-intensive, costly, and not always efficient.

Molecular docking is a method that uses molecular simulations to create 3D models for analyzing cofactor binding modes and specificity. A study used molecular docking to investigate interactions between RF and the active site in FAD synthetase from *Streptococcus pneumoniae*, providing insights into its structural preferences during FMN production (73). Similarly, docking studies have shed light on the specific interactions within various SAM domain complexes (74).

Another method to determine cofactor specificity and molecular recognition of binding is studying the structure of enzymes. The three-dimensional structure and atom distribution of the active site enables the enzyme to recognize and bind specific molecules, leading to catalytic activity. X-ray crystallography and Nuclear Magnetic Resonance (NMR) are used to elucidate structures recorded in the PDB database.

These methods provide insights into enzyme functions and interactions.

In recent years, the field of enzyme cofactor specificity prediction has seen significant progress with the introduction of ML and DL methods like Cofactory (7) and the Rossmann-toolbox (9), respectively. Cofactory uses ML, specifically hidden Markov models and Artificial Neural Network (ANN), to predict enzyme cofactor specificity based on primary amino acid sequences. It is capable of identifying potential cofactor-binding Rossmann folds and offers predictions for FAD(H2), NAD(H), and NADP(H) with Matthews correlation coefficients of 0.94, 0.79, and 0.65, respectively. Despite its contributions, Cofactory still faces challenges in accurately differentiating between the specificities of these cofactors. On the other hand, the Rossmann-toolbox uses DL to analyze the beta-alpha-beta motif of Rossmann fold proteins, providing accurate predictions for interactions with key cofactors such as NAD(H),NADP(H), FAD, and SAM.

A universal method is needed to accurately predict essential cofactors' specificity in characterized and uncharacterized enzymes. With the increasing number of protein structures available in the PDB, new approaches can be developed to improve the accuracy of cofactor prediction and identify the molecular factors responsible for cofactor specificity. However, determining the specificity of cofactors remains challenging due to the unique properties and roles of different types of cofactors in various enzymes. ML has emerged as an effective approach to handling this issue, as it can mine large amounts of data, identify useful correlations, and create models capable of predicting molecular descriptors and biological features.

2.3 Machine Learning

With the advance of computational methodologies and online resources, a huge amount of data has become available. This opens doors for detailed investigations into molecular recognition processes, like how different substrates and cofactors bind, unveiling specificity mechanisms. Yet, handling massive amounts of protein structures manually is impractical. Protein structural data, rich with thousands of atom coordinates per protein, demands specialized big data analysis methods. Additionally, given the overwhelming volume of available information, crucial data characteristics and patterns may inadvertently go unnoticed. This emphasizes the need for methodologies capable of learning from and interpreting such vast amounts of data. Thus, multiple ML algorithms have been applied to enzyme engineering (5; 6; 7; 8; 9; 10).

ML is a branch of Artificial Intelligence (AI) focused on creating algorithms and models that learn from data to make predictions, identify patterns, or perform tasks without being explicitly programmed for each action.

ML algorithms can be divided into 4 categories: supervised learning, which involves learning from labeled data; unsupervised learning, used for uncovering patterns in unlabeled data; semi-supervised learning, leveraging both labeled and unlabeled datasets for training; and reinforcement learning, where models learn from feedback based on their actions. Each category plays a distinct role in enabling machines to learn autonomously and adapt through diverse data-driven methodologies (75).

2.3.1 Unsupervised Learning

Unsupervised learning occurs when the learning system is tasked with recognizing patterns in data without the presence of pre-existing labels. In this scenario, only the input data is available, and the algorithm operates without knowledge of any corresponding output data. The absence of labeled data presents a significant challenge in unsupervised learning, as it becomes difficult to assess whether the algorithm has obtained meaningful insights. Consequently, measuring the success of unsupervised learning algorithms poses a considerable challenge. Due to the inherent difficulty in evaluating their effectiveness, unsupervised methods are often utilized for data exploration or preprocessing tasks, particularly in preparation for supervised learning. The two most used unsupervised categories are clustering and dimension reduction

Clustering

Clustering entails the categorization of data into distinct groups based on the similarities among data points' features and characteristics. Over time, numerous techniques for data clustering have been devised and applied to address clustering challenges. The two primary methods for clustering are hierarchical and partitional approaches. One of the most widely utilized algorithms in clustering analysis is K-Means Clustering, a partitional method. This technique partitions a dataset into k distinct clusters by assessing the similarities among data points (76).

Dimensional Reduction

Dimensionality reduction helps in visualizing high-dimensional data, compressing data for storage purposes, and speeding up subsequent ML algorithms by eliminating redundant or less informative features. One of the most commonly used approaches for the reduction of dimensionality is PCA.

PCA is a linear technique that identifies patterns and correlations in the data, projecting it onto a new set of axes (called principal components) that are orthogonal to each other. These components are ordered by how much variance they explain, with the first capturing the most. The goal of PCA is to reduce the number of features while keeping as much important information as possible. This makes it well-suited

for tasks like feature reduction before applying machine learning models (77).

On the other hand, non-linear methods like t-distributed Stochastic Neighbor Embedding (t-SNE) and Uniform Manifold Approximation and Projection (UMAP) focus more on visualization. These methods are designed to map complex relationships in high-dimensional data into a lower-dimensional space, helping to reveal clusters or patterns that may not be obvious with linear methods.

t-SNE, introduced by Hinton and Roweis (78), works by comparing the similarity of points in high-dimensional space and then mapping them to a lower-dimensional space while preserving those similarities as much as possible. This makes t-SNE especially useful for visualizing clusters in the data, though it is less suited for feature reduction in machine learning models due to the difficulty in interpreting the resulting dimensions. Linderman et al. (79) have also improved the computational efficiency of t-SNE, making it faster and easier to apply to larger datasets.

Another popular method is UMAP, which offers some advantages over t-SNE. UMAP is designed to capture both local and global relationships in the data by approximating its structure and then optimizing the layout in a lower-dimensional space. It is often praised for its ability to preserve more of the overall structure of the data compared to t-SNE, and it is generally faster and scales better to larger datasets. Like t-SNE, UMAP is mainly used for visualization rather than feature selection (80).

2.3.2 Supervised Learning: Models and Algorithms

In supervised learning, an algorithm is trained on labeled data to predict outcomes for new data. There are two main types of predictive tasks within supervised learning: regression, which predicts continuous numerical outcomes, and classification, which predicts categorical outcomes. Further, classification can be separated into binary, multiclass, or multilabel classification. Binary classification is a supervised ML algorithm that is used to predict one of two classes for an item, while multiclass and multilabel classification is used to predict one or more classes for an item. A multiclass classifier must assign one and only one class or label to each data sample, whereas a multilabel classifier can assign zero or more classes or labels to the same data sample (81). Below, a brief description of the Ml workflow and most popular ML algorithms is addressed.

In supervised ML, a range of algorithms is available for performing tasks like classification and regression. The most widely used algorithms are: Linear Regression (LR), Logistic Regression (LgR), K-Nearest Neighbors (KNN), Support Vector Machines (SVM), Decision Tree (DT), Random Forest (RF), and ANN (82).

Linear Regression LR is the simplest algorithm for regression analysis, which involves establishing a linear relationship between the dependent variable (target variable) and the independent variables (predictor variables) (83). The LR formula is represented by equation 5.

$$\hat{y} = \theta_0 + \sum_{j=1}^{p} x_j \theta_j \tag{5}$$

where $\hat{y}$ is the predicted output, θ_0 represents the intercept term commonly called bias, x_j are the input features, θ_j are the respective weights associated with the input features, and p signifies the number of features.

LR aims to determine the optimal best-fitting line that minimizes the difference between predicted values and the actual values of the target variable (83). It achieves this by minimizing a specific cost function, often represented by the Mean Squared Error (MSE) as shown in equation 6:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta \left(x^{(i)} \right) - y^{(i)} \right)^2 \tag{6}$$

where $J(\theta)$, evaluates the accuracy of a linear regression model by averaging the squared differences between the model's predictions, $h_\theta(x^{(i)})$, and the actual values, $y^{(i)}$.

Logistc Regression LgR is commonly used for binary classification problems. Equation 7 characterizes the logistic function, commonly referred to as the sigmoid function, employed to ascertain the probability distribution of a sample belonging to a specific class:

$$\varphi(z) = \frac{1}{1 + e^{-z}} \tag{7}$$

This function $\varphi(z)$, tend towards 1 as z approaches to infinity $(z \to +\infty)$, and converges towards 0 as z tends towards negative infinity $(z \to -\infty)$. When z equals 0, the probability predicted by the sigmoid function is 0.5. Hence, the sigmoid function effectively transforms input values into a range spanning from 0 to 1 (82). Assuming the parameterization of input data by the weights θ_0, equation 8 emerges:

$$h_\theta(s) = \frac{1}{1 + e^{-\theta^T x}} \tag{8}$$

The estimation of weights θ_0 involves minimizing the cost function $J(\theta)$ using gradient descent methods, an aspect elaborated upon in subsequent sections:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(h_\theta\left(x^{(i)}\right)\right) + \left(1-y^{(i)}\right)\log\left(1-h_\theta\left(x^{(i)}\right)\right)\right] \quad (9)$$

k-Nearest Neighbors KNN is a non-parametric, instance-based learning algorithm used for both classification and regression analysis. The algorithm works on a simple similarity principle: it compares the similarity of a new data point to old data points to anticipate the target value. KNN does not build a model from the training data; instead, it memorizes the dataset. KNN classifies the data point based on the majority voting of the labels of the *K* closest neighbors in classification tasks or computes the average of their values in regression by measuring the distance between the new data point and its neighbors, which is determined by a user-defined '*k*' value (the number of nearest neighbors) (84).

Support vector machine SVM are commonly used for classification and regression tasks. They are particularly effective in high-dimensional spaces and perform well when the data exhibits clear margin separations. The main objective of SVM is to find a hyperplane that best separates different classes within the dataset. This hyperplane, also known as the decision boundary, ensures the maximum possible separation between data points of different classifications. SVM works by transforming input data into a higher-dimensional space and selecting the optimal hyperplane for class separation. The ability of the technique to handle non-linear data is enhanced by kernel functions, such as linear, polynomial, or sigmoid, which transform the original input space into a higher-dimensional space, allowing SVM to perform well on complex datasets (82).

Decision trees DT are used for both classification and regression tasks. These models learn simple decision rules from the data features to predict the target variable's value. A DT structure consists of nodes that represent features or attributes, edges that denote the decision rules, and leaf nodes representing the class label or predicted value. DT are intuitive, easy to interpret, and suitable for handling both numerical and categorical data. However, they are prone to overfitting when they become overly complex. Techniques like pruning and using ensemble methods such as RF mitigate this issue while enhancing their predictive performance (85).

Random forest RF, as illustrated in Figure 1, is an ensemble learning method used for classification tasks. Ensemble learning refers to a technique that combines multiple individual models to create a more robust and accurate predictive model, which reduces the risk of overfitting. By aggregating diverse models, ensembles can capture complex patterns and relationships in the data that might not be

discernible by a single model. RF method is the aggregation of multiple DT, each tree in the ensemble is built using a subset of the training data and a random subset of features, aiming to introduce diversity among trees. The final prediction in a RF is determined by aggregating the predictions of all individual trees. As a result, RF is highly effective, versatile, and less prone to overfitting compared to single DT (86).

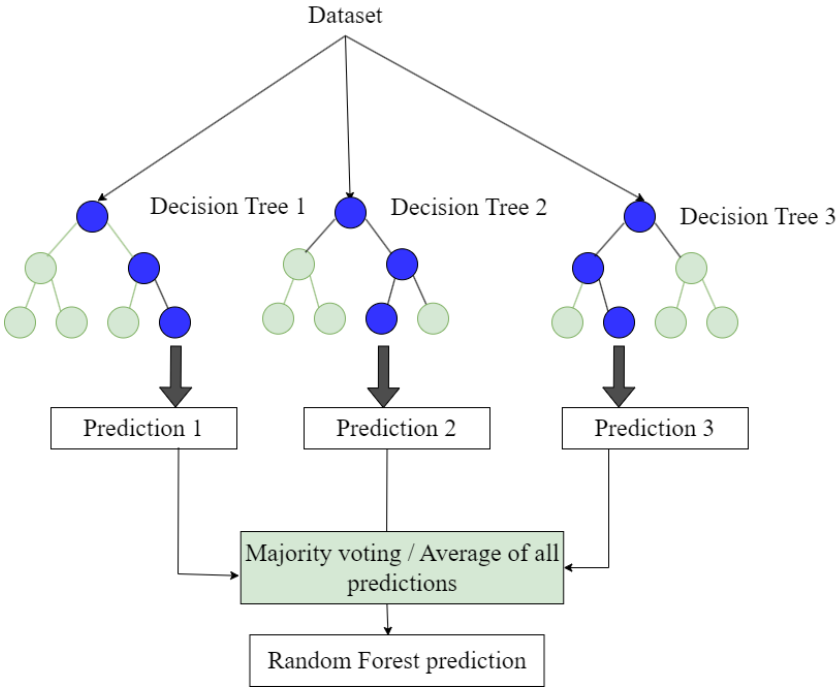


Figure 1: **Random Forest Structure.** The input dataset feeds multiple decision trees, each making an independent prediction ('Prediction 1', 'Prediction 2', 'Prediction 3'). The final output is determined by majority voting (for classification) or averaging (for regression), resulting in the 'Random Forest prediction'.

Artificial Neural Networks ANN are ML models inspired by the human brain's structure and function. Just like the brain uses neurons to process information, ANN use simple processing units called neurons that are connected to each other.

In an ANN, knowledge is stored within the connections between these neurons. The network learns by being exposed to data, which represents an external environment. Through a training algorithm, the network adjusts the strengths (or weights) of the connections between neurons. This adjustment allows the network to recognize patterns, make predictions, and derive insights from the data (87).

The ANNs consist of interconnected nodes or neurons arranged in layers: an input layer receives data, one or more hidden layers perform computations and an output layer produces the final result. The network learns by adjusting its internal parameters called weights, initially set randomly and refined during training. Forward propagation involves feeding input data through the network, producing output predictions compared against actual values using a loss function. Backpropagation then fine-tunes the weights by minimizing this loss, enhancing prediction accuracy (88).

ANNs employ a layered architecture to transform inputs into meaningful outputs. Neurons in the input layer represent raw data features, while hidden layers serve as abstraction layers, extracting and learning patterns from data. The output layer synthesizes the learned information to generate the final output, with the number of neurons in this layer tailored to the problem at hand: single neurons for regression or binary classification tasks and multiple neurons for multiclass classification aligning with the number of distinct classes. Figure 2 shows the ANNs architecture (89).

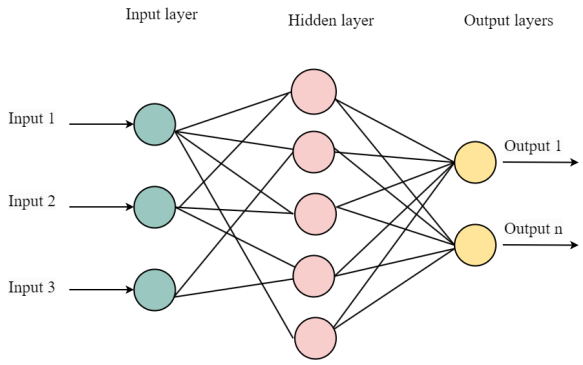


Figure 2: **Artificial Neural Network Architecture.** The input layer has three nodes (Input 1, Input 2, Input 3), fully connected to four neurons in the hidden layer. These connections, representing weights, transform signals passed to the output layer, which contains two nodes (Output 1, Output n) for the network's final predictions.

2.3.3 Supervised Learning: ML workflow

A ML workflow generally involves several key steps or stages that collectively enable the development, training, evaluation, and deployment of ML models. Figure 3 shows an outline of the typical ML workflow.

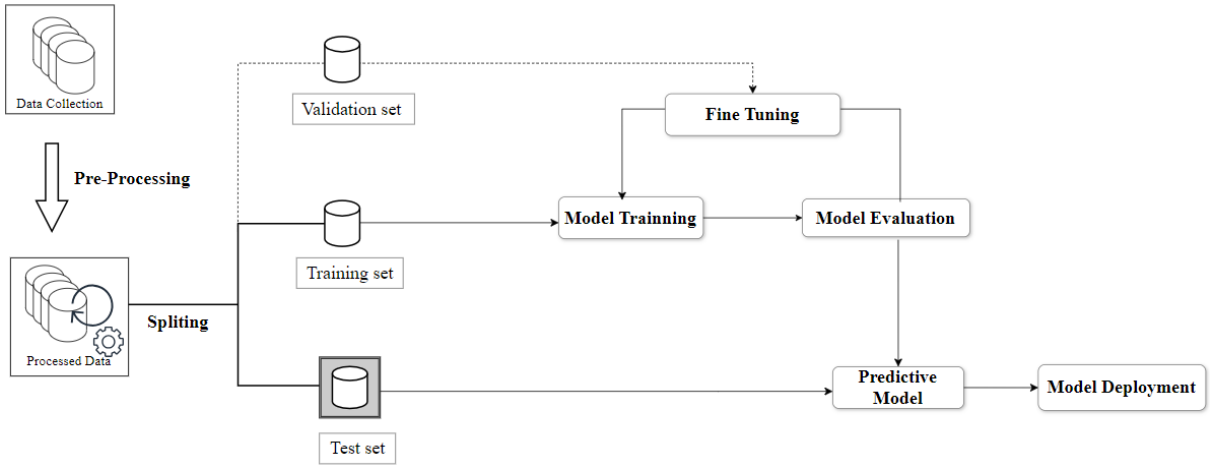


Figure 3: **Machine Learning workflow.** This workflow diagram shows the steps in a ML project, starting with data collection, followed by pre-processing, splitting into train and test data, training with hyperparameter tuning, evaluation, and model deployment.

Data Collection

Data collection is the first step in the ML workflow, which involves gathering, analyzing, and structuring data from different sources. This phase is critical because the quality and relevance of the data significantly affect the accuracy and efficiency of subsequent ML methods. Typically, data is sourced from various repositories, databases, or through data scraping techniques from web platforms, research papers, or domain-specific sources (90).

Data Pre-Processing

Once the data is collected, it goes through a pre-processing phase. This step involves cleaning, transforming, and organizing the data into a format that is ready for analysis. Tasks like handling missing values, scaling, and dealing with outliers are crucial here, as they ensure the data is reliable and suitable for model training (91).

Feature Extraction

Data comes in multiple forms, such as numerical, categorical, and ordinal. However, for ML classification tasks, it is essential to convert string data into numerical values to create an input matrix for the model. Dealing with categorical or string data involves using methods like One-Hot-Encoding, embedding techniques, and sequence descriptor extraction.

Label Encoding and One-Hot Encoding are two fundamental techniques used for transforming categorical data into numerical form that ML algorithms can effectively employ.

In One-Hot Encoding (92), a new binary column is created for each unique category in the original feature. A '1' indicates the presence of the category, while '0' represents its absence.

When dealing with biological data such as protein sequences, advanced embedding techniques are necessary to capture the complex relationships within the data. Embeddings generate dense vectors where each dimension represents specific information about the sequence, making them invaluable for protein function analysis.

SeqVec, a bi-directional Long Short-Term Memory (LSTM) model, applies methods originally developed for natural language processing to biological sequences. It treats protein sequences as sentences, with each amino acid acting like a word. By training on large datasets, SeqVec captures both local and global dependencies within the sequence, identifying relationships between amino acids that may be close together or far apart. This enables SeqVec to provide deep insights into the structural and functional properties of proteins, making it especially useful for tasks like predicting enzyme-cofactor binding (93).

ESM2, a transformer model developed by Facebook AI, takes a different approach by leveraging self-attention mechanisms. This allows ESM2 to focus on multiple parts of the sequence simultaneously, efficiently capturing long-range dependencies that may be difficult for models like SeqVec to handle. ESM2 is particularly effective at modeling complex relationships between amino acids and was used in this study to generate embeddings that capture both sequence-level and structural information, which are critical for predicting enzyme functions and cofactor binding (94).

FlashProt, another transformer-based model, is designed for high-efficiency applications. While it shares ESM2's self-attention mechanisms, FlashProt is optimized for faster processing and inference on large datasets, making it ideal for scenarios requiring real-time or large-scale analysis. FlashProt excels at extracting the most critical features from protein sequences while maintaining computational efficiency, which is key for handling the vast amount of data involved in enzyme-cofactor interaction studies (95).

Beyond embeddings, sequence descriptors are used to transform molecular sequences into formats understandable by ML algorithms. These descriptors provide detailed information about the structure, function, and evolution of biological sequences. Physicochemical descriptors capture properties like molecular weight, hydrophobicity, and charge distribution, offering insights into how these physical traits influence protein behavior (96). Statistical descriptors focus on patterns in amino acid or nucleotide composition, transition, and distribution (96), while structural descriptors quantify 3D characteristics like shape and flexibility, crucial for protein structure prediction and drug design (97). Together, these techniques allow machine learning models to work with rich, detailed representations of biological data, driving more accurate predictions in bioinformatics applications.

Feature Selection

Feature selection is the process of identifying a subset of features that accurately represent the target concept. It aims to deal with high dimensionality, choosing the most relevant features while discarding redundant or irrelevant ones, improving separability between categories, and comparability of features within the same category avoiding overfitting. There are three basic approaches: filter, wrapper, and embedded methods (98).

Filter methods select the most relevant features through the data itself, the features selected are evaluated based on intrinsic properties of the data, guiding the search of relevant features. Compared to the wrapper methods, filter methods are faster to compute. One example of a filter method is univariate feature selection, which selects features based on the statistically significance relationship between each feature and the target. This filter method can include Analysis of variance (ANOVA), Linear Discriminant

Analysis (LDA), Pearson's Correlation, Chi-square, and mutual information (99).

Wrapper models use classification error or accuracy rate as their criteria for evaluating features. Unlike filter models, wrapper models integrate the learning algorithm into the feature selection process, producing the selected features alongside the learning model itself. This approach typically results in higher classification accuracy and a smaller subset of features. However, wrapper models are often criticized for their poor generalization capabilities and high computational complexity compared to the filter methods. One of the most studied wrapper methods is the SVM-RFE uses support vector machines to measure the performance of features and constructs a classifier with high-performance scores (100).

The embedded methods select features during the training process of the learning model, and the feature selection result is automatically outputted when the training process is finished. The most widely used methods are Least absolute shrinkage and selection operator (Lasso) and Ridge regression.

Lasso, also known as l1 regularization, minimizes the residual sum of squares while constraining the sum of absolute values of regression coefficients. This results in some coefficients being exactly zero, effectively performing variable selection. Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC) further refine the model by truncating variables for dimensionality reduction. Ridge regression, on the other hand, incorporates a regularization term into the cost function to prevent overfitting. It adds an L2 regularization term to the linear equation, penalizing the sum of squared coefficients to stabilize the model, especially in cases of multicollinearity or many predictors (101).

Data Splitting

Data splitting is the process that divides a dataset into two parts: training and testing. A widely used and straightforward method for splitting a dataset involves random sampling. In this approach, a certain percentage of the dataset is randomly chosen. In some cases, it might be useful to employ stratified sampling, which involves splitting the dataset in such a way that the original distribution of the dataset is maintained within each subset.

The training set is used to teach the ML model how to recognize patterns and correlations in data. Meanwhile, the testing set is kept hidden during training and is used to evaluate the model's performance and generalization on fresh, unknown data. This distinction prevents overfitting and guarantees that the model can generate accurate predictions on new data that it hasn't seen previously.

Sometimes, a dataset is divided into three parts: a training set, a validation set, and a testing set. The validation set is used during training to fine-tune the model's parameters and optimize its performance (102).

Model Selection and Training

The model selection is a crucial step for the model´s training phase. It involves assessing various models based on the problem's nature, dataset size, complexity, and desired outcomes. Experimentation with different models allows for the selection of the most suitable one, ensuring it aligns with the problem statement and data characteristics. During training, the model learns complex patterns in the data and adjusts its settings to make accurate predictions. We then test these predictions on new, unseen data to see how well the model performs. The goal here is to use the carefully prepared data to train a selected model that has a high potential for good performance. After training, we thoroughly tested the model to see how effective it was. There are different methods and metrics to validate the model during this process.

Performance Metrics The performance metrics are useful for understanding the model's effectiveness and performance. For binary classification problems, the most widely used metrics include recall, f1-score, accuracy, and precision (103). Moreover, a confusion matrix is often built to allow an intuitive evaluation of the model.

Table 3 shows a confusion matrix. It is a tabular representation that cross-references the actual and predicted classes, showcasing True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN) values, which are the four possible prediction outcomes for a classification problem. TP instances signify accurate positive predictions, while TN denotes accurate negative predictions. FP represents incorrect positive predictions, where the model classifies a negative instance as positive. Conversely, FN represents incorrect negative predictions, marking instances where the model labels a positive observation as negative.

Table 3: Confusion Matrix.

		Predicted Values	
		Positive	Negative
Real values	Positive	True Positive (TP)	False Negative (FN)
	Negative	False Positive (FP)	True Negative (TN)

Upon establishing the values for TP, TN, FP, and FN derived from the confusion matrix, determining further classification metrics becomes a straightforward process. Table 4 provides a comprehensive breakdown of each metric, detailing the computational methodologies behind them.

The Area Under the Curve (AUC), is a widely used performance metric in classification tasks, particularly for evaluating the quality of binary classification models. It represents the overall measure of a model's ability to discriminate between positive and negative classes across various threshold values.

AUC calculates the area under the Receiver Operating Characteristic (ROC) curve, plotting the true

Table 4: Classification Metrics.

Metric	Description	Equation
Accuracy	Fraction of correct predictions	$\frac{TP+TN}{TP+TN+FP+FN}$
Recall	Fraction of actual positives correctly identified	$\frac{TP}{TP+FN}$
Precision	Fraction of the number of true positives divided by the number of predicted positives	$\frac{TP}{TP+FP}$
F1-score	Harmonic mean of precision and recall	$2 \times \frac{Precision \times Recall}{Precision+Recall}$
MCC	Balanced measure, specifically when the classes have significantly different sizes.	$\frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$

positive rate (sensitivity) against the false positive rate (1-specificity) at different classification thresholds. An AUC value closer to 1 signifies a superior model with better discrimination capabilities, while a value around 0.5 indicates a classifier with performance akin to random guessing (91).

On the other hand, for regression problems, the assessment of the model performance often involves the utilization of metrics such as the MSE, Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and Coefficient of Determination (R^2). The MSE computes the average of the squared differences between the actual and predicted values, providing a comprehensive overview of the model's overall predictive accuracy. Similarly, the RMSE, derived from the MSE, calculates the square root of the average squared differences, representing a more interpretable metric. The coefficient of determination, R^2, quantifies the proportion of variance in the dependent variable that is predictable from the independent variable(s) in the model, indicating the goodness of fit. Conversely, the MAE determines the mean of the absolute differences between the predicted and actual values, offering a more intuitive understanding of the error magnitude (104).

Model Evaluation and Improvement

Hyperparameter tuning

Hyperparameter tuning is a crucial step in optimizing ML models. Unlike model parameters, which are learned during the training process, hyperparameters are set before the learning process begins and significantly influence a model's performance. Examples of hyperparameters include the learning rate in gradient descent-based algorithms, the depth of DT in random forests, or the number of hidden layers in a neural network. The aim is to find the best hyperparameter combination that maximizes evaluation metrics like accuracy or precision. This process often involves testing various values to enhance performance without overfitting during training. Grid search and random search are common approaches to hyperparameter tuning. Grid search systematically tests predefined hyperparameter combinations across a grid, while random search randomly samples hyperparameters from specified distributions. More advanced techniques

like Bayesian optimization or evolutionary algorithms can efficiently explore the hyperparameter space, reducing the computational burden of exhaustive search methods. (91).

In addition to hyperparameter tuning, ensemble methods stand out as a rich strategy for enhancing model performance (105). These methods combine multiple models to produce improved predictions. Bagging, boosting, and stacking are the most common types of ensemble techniques. Bagging involves training multiple models in parallel on different subsets of the data and then combining their predictions through voting or averaging, reducing variance and enhancing reliability; RF (86) is a well-known example of bagging. Boosting employs a sequential approach where models are trained on the full dataset with a focus on instances that were previously misclassified, gradually refining accuracy; AdaBoost (106) and Gradient Boosting (107) are noted examples. Lastly, stacking takes a more intricate approach by training diverse base models and employing a meta-learner to learn how to best combine their strengths, resulting in a final prediction that leverages the expertise of each base model, ultimately achieving superior predictive performance through collaboration (108).

Cross Validation

Cross-validation is a statistical technique for assessing the generalization performance of a model, offering greater stability and comprehensiveness than simply splitting the data into a training set and a test set. Instead of a single split, cross-validation repeatedly partitions the data and trains multiple models. The most popular form of cross-validation is k-fold cross-validation, where the data is partitioned into k subsets, training the model on k − 1 subsets and validating it on the remaining subset in each iteration. At the most extreme, Leave-One-Out Cross-Validation (LOOCV) is sometimes used, which is a special case of k-fold cross-validation, where k equals the number of samples in the dataset. The model is trained n times, each time leaving one sample out for validation. Usually, this technique is used when we deal with a small amount of data (109).

Cross-validation offers several benefits over a single training-test split. It reduces the risk of skewed performance evaluations caused by "lucky" or "unlucky" data splits. Each data point is used in both training and testing, which helps the model to generalize well. This method also provides insights into the model's sensitivity to the data selection and uses the data more efficiently, often resulting in more accurate models. However, it increases computational cost as multiple models are trained instead of just one (99).

Model Deployment

After completing the training, evaluation, and fine-tuning of the ML model, the final phase entails its deployment into a production setting. This critical step makes the model operational, enabling it to execute predictions or classifications on novel data that it hasn't encountered before. (91).

2.3.4 Python libraries for machine and deep learning

When working with ML and DL, several Python libraries serve different roles, which can be grouped into three categories: auxiliary tools, model training libraries and visualization tools.

To better illustrate the distinctions between these libraries, Table 5 provides a structured comparison, categorizing each library by its core functions—whether in model training, optimization, or visualization.

Table 5: Overview of Python libraries for ML/DL.

Category	Libraries
Auxiliary Libraries	NumPy, pandas, SciPy, ProPythia, Optuna
Model Training Libraries	Scikit-learn, Keras, PyTorch, TensorFlow
Visualization Libraries	Matplotlib, Seaborn

Auxiliary Libraries

These libraries handle essential tasks like data manipulation and preparation. NumPy (110) is a Python library specialized in multidimensional array and matrix processing, while pandas (111) is responsible for preparing datasets for ML and training, relying on one-dimensional (series) and two-dimensional (dataframe) data structures. SciPy (112) builds on Numpy for scientific computing and Propythia (113), developed at CEB/U. Minho, is an example of a specialized tool used in bioinformatics for protein sequence analysis, providing a user-friendly platform for ML and DL applications.

Model Training Libraries

These libraries are at the core of building, training, and optimizing machine learning ML and DL models, each with its unique strengths.

Scikit-Learn is an open-source ML library built on NumPy and SciPy, providing a diverse range of algorithms that support both supervised and unsupervised learning. It also comes with tools for data mining, feature selection, model evaluation, and cross-validation. What makes Scikit-learn particularly useful is how easy it is to integrate with other libraries, making it a foundational tool in any ML workflow.

Keras is a user-friendly, high-level framework for building neural networks. It's designed to make DL development simple and accessible, allowing you to quickly prototype models while still offering flexibility for more complex architectures. Keras runs on top of either TensorFlow or PyTorch (114), which gives it the versatility to train models on CPUs or GPUs, making the development process faster and smoother. Whether you're building CNNs, Recurrent Neural Networks (RNN)s, or other types of DL models, Keras helps streamline the process.

PyTorch (114) is highly valued for its dynamic nature, allowing developers to make changes to the model architecture on the go. This flexibility, combined with strong Graphics Processing Units (GPUs) acceleration, makes PyTorch a favorite for applications like Natural Language Processing (NLP) and Computer Vision. With its easy debugging and a rich set of pre-built layers and optimization tools, PyTorch has become a leading choice for both research and production-level DL projects.

TensorFlow (115), developed by Google, is another powerhouse in the DL world. It is built for scalability and can efficiently handle large-scale data across multiple CPUs or GPUs. TensorFlow is highly versatile, supporting everything from simple feed-forward networks to complex models like transformers. It also comes with TensorBoard, a visualization tool that helps track model performance, and TensorFlow Extended (TFX), which makes deploying models in production easier.

Optuna (116) is an open-source optimization framework specifically designed for hyperparameter tuning. It is particularly useful for improving the performance of ML and DL models by automating the search for the best hyperparameters. Optuna uses techniques like Bayesian optimization to efficiently explore large parameter spaces, helping you find the optimal configurations for your models. In practice, it integrates seamlessly with libraries like PyTorch and Scikit-learn, allowing users to tune parameters for a wide range of algorithms.

Visualization Libraries

Matplotlib and Seaborn are essential for data and model result analysis. Matplotlib (117) creates a broad range of static plots, offering full control for publication-ready figures, while Seaborn (118) builds on it with more complex plots, like heatmaps and violin plots. With its high-level functions, Seaborn enables easier data trend visualization and works smoothly with pandas dataframes for a seamless workflow.

2.3.5 Multi-Label Classification Models

In MLC, two primary strategies address the complexity of predicting multiple labels: problem transformation and algorithm adaptation. Each strategy takes a unique approach to handling multi-label data by either

breaking down the problem into simpler tasks or adapting models directly for multi-label requirements.

Problem Transformation Methods

Problem transformation reformulates the MLC task into a series of single-label or ranking problems, allowing standard ML methods to be applied.

BinaryRelevance (BR) (119) treats each label as an independent binary classification problem, constructing a separate classifier for each. This "one-versus-all" method allows flexibility across labels but may miss the dependencies among them, reducing accuracy when label correlations are significant.

LabelPowerset (LP) (120) considers each unique label set as a single class, transforming the MLC task into a multi-class classification problem. This approach accounts for label correlations but may lead to a large number of classes, particularly in datasets with many unique label combinations, increasing the model's complexity and potential for overfitting.

ClassifierChain (CC) (121) extends BR by arranging the binary classifiers in a chain where each classifier considers the previous classifiers' predictions. This method captures label dependencies more effectively than BR but can be computationally demanding, and finding the optimal order for the chain of labels remains challenging.

Algorithm Adaptation Approaches

Algorithm adaptation directly extends machine learning models to handle multi-label data. This approach avoids transforming the problem, aiming instead to improve existing algorithms' performance on MLC tasks by incorporating label dependencies and correlations.

Multi-label k-Nearest Neighbour (MLkNN) (122) is an adaptation of the KNN algorithm tailored for MLC. Using Bayesian inference, MLkNN estimates the probability of each label's relevance based on the k-nearest neighbors in the training set. This method performs well with sufficient training data, effectively capturing local label dependencies, but its reliance on the choice of k and distance metrics, along with high computational demands, limits its scalability to large datasets.

Multi-label decision tree algorithm (MT-DT) adapts traditional decision trees for MLC by modifying splitting criteria to simultaneously predict multiple labels. Each node evaluates label distributions to select optimal splits that consider label dependencies. MT-DT provides a clear and interpretable model structure, effectively handling multi-label interactions; however, it may become large and less accurate with imbalanced datasets or when handling many labels.

Ranking Support Vector Machine (Rank-SVM) (123) adapts the SVM algorithm for MLC by learning

a ranking function to prioritize relevant labels for each instance. By focusing on the ordering of label relevance, Rank-SVM considers dependencies between relevant and irrelevant labels. It captures label relationships effectively, although it is computationally intensive and sensitive to ranking loss and hyper-parameters, limiting its scalability.

Each method provides unique advantages and limitations in MLC, depending on dataset size, label dependencies, and computational resources. The choice of model strategy thus relies heavily on the specific requirements of the multi-label problem and the data's structure.

2.3.6 Multi-Label Performance Metrics

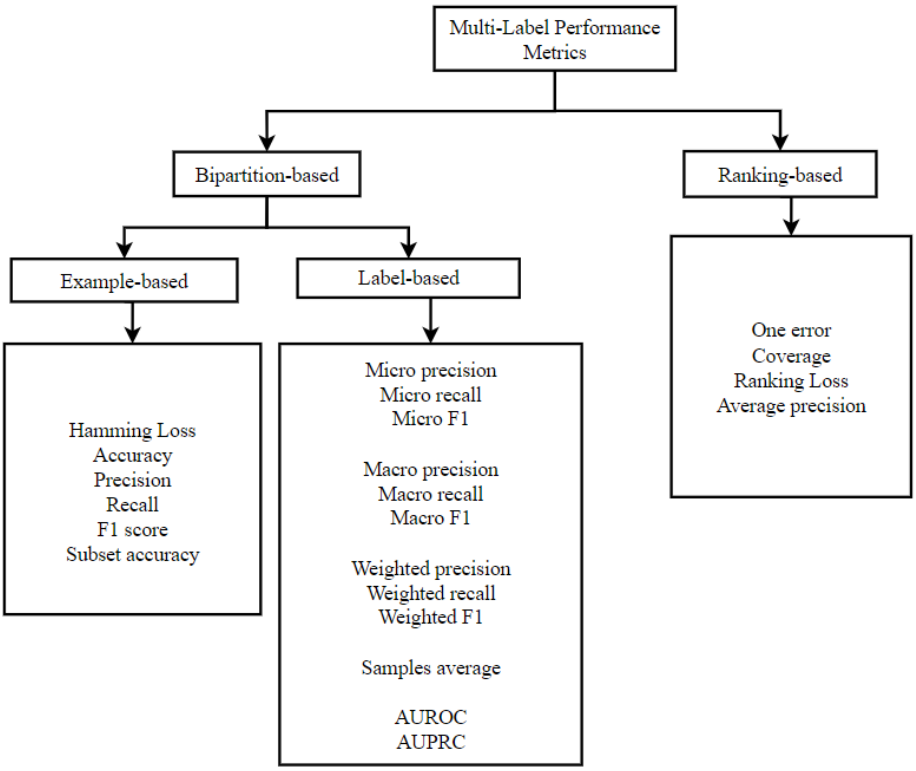


Figure 4: **Performance metrics for MLC.** There are two types of measures: one is used to evaluate the performance of methods, and the other is used to evaluate the efficiency of methods. Performance evaluation can be done by evaluating bipartitions or by evaluating the relevance of a label for the sample.

The evaluation of the performance of a MLC method depends on the type of predictions it makes. There are two main types of metrics: bipartition-based metrics, which can be divided into label-based and example-based categories. Label-based metrics assess the performance of the learning system on each label individually, without considering any relationship between the labels. On the other hand, example-based metrics evaluate the learning system's performance on each test example separately and then calculate the average value across the test set, considering the complete set of labels predicted for each

instance and capturing the correlations among different labels.

Example-based Metrics

Hamming Loss measures the fraction of labels that are incorrectly predicted. It quantifies the frequency of misclassification.

$$\text{Hamming Loss} = \frac{1}{N \times L} \sum_{i=1}^{N} \sum_{j=1}^{L} \mathbb{I}(y_{ij} \neq \hat{y}_{ij})$$

where N is the number of instances, L is the number of labels, y_{ij} is the true label, $\hat{y}_{ij}$ is the predicted label, and $\mathbb{I}$ is an indicator function.

Subset Accuracy , also known as Exact Match Ratio, evaluates the proportion of instances where the entire set of predicted labels exactly matches the true set of labels.

$$\text{Subset Accuracy} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}(\mathbf{y}_i = \hat{\mathbf{y}}_i)$$

where N is the number of instances, $\mathbf{y}_i$ is the true label set, and $\hat{\mathbf{y}}_i$ is the predicted label set for instance i.

Ranking-based Metrics

Coverage Assesses the average number of steps required to cover all relevant labels in the ranked label list of an instance.

$$\text{Coverage} = \frac{1}{N} \sum_{i=1}^{N} \max_{j:y_{ij}=1} \text{rank}_{ij}$$

where rank_{ij} is the position of label j in the ranked list of labels for instance i.

Ranking Loss evaluates the average fraction of label pairs that are incorrectly ordered. It measures the extent to which the predicted ranking of labels for example disagrees with the true ranking, i.e., an irrelevant label is ranked higher than a relevant label.

$$\text{Ranking Loss} = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{|Y_i^+||Y_i^-|} \sum_{(j,k) \in Y_i^+ \times Y_i^-} \mathbb{I}(\text{rank}_{ij} > \text{rank}_{ik})$$

where Y_i^+ is the set of relevant labels and Y_i^- is the set of irrelevant labels for instance i.

One-Error estimates the fraction of instances where the top-ranked label is not in the set of relevant labels. It focuses on the accuracy of the highest-ranked prediction.

$$\text{One-Error} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}(\text{rank}_{i1} \notin Y_i^+)$$

where rank_{i1} is the top-ranked label for instance i.

For coverage, ranking loss, and one-error, lower values indicate better performance, while higher values are desirable for metrics like subset accuracy

Label-based Metrics

Macro-average will compute the metric independently for each class and then take the average (hence treating all classes equally).

$$\text{Macro Precision} = \frac{1}{L} \sum_{j=1}^{L} \frac{\text{TP}_j}{\text{TP}_j + \text{FP}_j}$$

$$\text{Macro Recall} = \frac{1}{L} \sum_{j=1}^{L} \frac{\text{TP}_j}{\text{TP}_j + \text{FN}_j}$$

$$\text{Macro F1} = \frac{1}{L} \sum_{j=1}^{L} \frac{2 \times \text{Precision}_j \times \text{Recall}_j}{\text{Precision}_j + \text{Recall}_j}$$

Micro-average will aggregate the contributions of all classes to compute the average metric. This approach gives equal weight to each instance, treating all classes as a single combined binary classification.

$$\text{Micro Precision} = \frac{\sum_{j=1}^{L} \text{TP}_j}{\sum_{j=1}^{L} (\text{TP}_j + \text{FP}_j)}$$

$$\text{Micro Recall} = \frac{\sum_{j=1}^{L} \text{TP}_j}{\sum_{j=1}^{L} (\text{TP}_j + \text{FN}_j)}$$

$$\text{Micro F1} = \frac{2 \times \text{Micro Precision} \times \text{Micro Recall}}{\text{Micro Precision} + \text{Micro Recall}}$$

Weighted-average Weighted-average accounts for the number of true instances for each class (i.e., class prevalence) by computing the metric for each class and then taking the weighted average according to the number of instances of each class.

$$\text{Weighted Precision} = \frac{\sum_{j=1}^{L} w_j \times \frac{\text{TP}_j}{\text{TP}_j + \text{FP}_j}}{\sum_{j=1}^{L} w_j}$$

$$\text{Weighted Recall} = \frac{\sum_{j=1}^{L} w_j \times \frac{\text{TP}_j}{\text{TP}_j + \text{FN}_j}}{\sum_{j=1}^{L} w_j}$$

$$\text{Weighted F1} = \frac{\sum_{j=1}^{L} w_j \times \frac{2 \times \text{Precision}_j \times \text{Recall}_j}{\text{Precision}_j + \text{Recall}_j}}{\sum_{j=1}^{L} w_j}$$

Where w_j is the number of true instances for class j.

Sample-average Sample-average computes metrics at the instance level and then takes the average over all instances. This approach is used when each instance may belong to multiple classes (MLC).

$$\text{Sample Precision} = \frac{1}{N} \sum_{i=1}^{N} \text{Precision}_i$$

$$\text{Sample Recall} = \frac{1}{N} \sum_{i=1}^{N} \text{Recall}_i$$

$$\text{Sample F1} = \frac{1}{N} \sum_{i=1}^{N} \text{F1}_i$$

Where N is the total number of samples, and Precision_i, Recall_i, and F1_i are the precision, recall, and F1 score calculated for each sample.

AUROC and **AUPRC** measure the area under ROC and Precision-Recall curves, respectively, to assess ranking quality, particularly for imbalanced data.

2.3.7 Challenges in Multi-Label Classification

Multi-label classification (MLC) is a complex field with unique challenges. While various solutions have been proposed, their effectiveness often depends on specific dataset characteristics. Key challenges in MLC include the following:

Data Cleaning

Data cleaning is a foundational step in preparing multi-label datasets, focusing on removing noise, handling missing values, and resolving inconsistencies. In highly varied or unstructured data (e.g., candidate skills in

recruitment), techniques like tokenization and stop word removal are essential. However, these methods can inadvertently remove valuable information, requiring a balance between cleaning and information retention. Malik and Bhardwaj (124) proposed a method to detect label errors, addressing this challenge directly.

High Dimensionality

High dimensionality, a prominent challenge in MLC, leads to increased risks of overfitting and higher computational demands. Dimensionality reduction methods such as PCA, LDA, Singular Value Decomposition (SVD), and t-SNE aim to emphasize relevant features. However, these unsupervised techniques overlook label information, which can be vital for enhancing performance in multi-label contexts (125).

Label Dependency

Label dependencies, whether conditional (dependent on instance features) or unconditional (independent of instance features), can significantly simplify and improve MLC performance. Efficient multi-label classifiers should capture these dependencies, yet managing them simultaneously often complicates the process. While some approaches attempt to leverage label dependencies, finding an optimal strategy remains challenging (126).

Data Imbalance

Data imbalance arises when some labels are underrepresented, potentially diminishing classifier performance. Lesser-represented labels might carry higher importance. Methods to reduce hamming loss (127) address imbalance to some extent, but comprehensive solutions are still needed. Current research focuses on improving approaches to handle this imbalance effectively (128; 129).

2.4 Deep Learning

DL stands as a subset of ML techniques rooted in the concept of ANNs, mirroring the intricate functionality of the human brain. It distinguishes itself through multi-layered architectures, enabling systems to autonomously derive crucial representations required for pattern recognition and feature detection from raw data. This ability significantly reduces dependency on handcrafted features, granting machines the power to learn and discern complex patterns. One of its fundamental advantages lies in its capacity to automate feature extraction from data, leading to breakthroughs in diverse fields such as image and speech

recognition, natural language processing, and autonomous decision-making systems (130; 131).

Figure 5 shows a Deep Neural Network (DNN), which serves as a foundational model within DL, featuring multiple interconnected layers of neurons. The DNNs have better performance compared to ANNs; however, DNNs incorporate more layers, so they face a higher risk of overfitting, where the model excessively learns specific patterns from the training data, hindering its ability to generalize to new data.

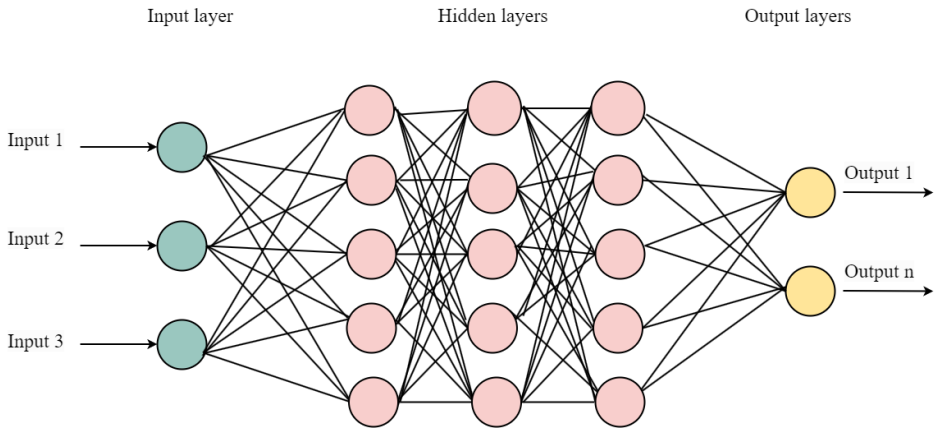


Figure 5: **Deep Neural Network Structure.** A DNN with one input layer, multiple hidden layers, and an output layer. The input layer (Input 1 to Input 3) feeds features into the network, where hidden layers of interconnected neurons extract patterns through learned transformations. The output layer (Output 1 to Output n) produces the final prediction or classification.

Deep learning architectures

Convolutional Neural Networks: CNNs are a class of DL models that are inspired by the structure of the animal visual cortex (132), designed to handle data with a grid pattern, such as images. CNNs represent a mathematical architecture consisting of three primary layers: convolutional, pooling, and fully connected layers. The initial layers, namely convolutional and pooling, undertake the crucial tasks of feature extraction and dimensionality reduction, respectively. These layers process input data, discerning essential patterns and reducing complexity. Subsequently, the fully connected layers assimilate the extracted features and translate them into the final output, facilitating classification or regression processes. Figure 6 illustrates a CNN.

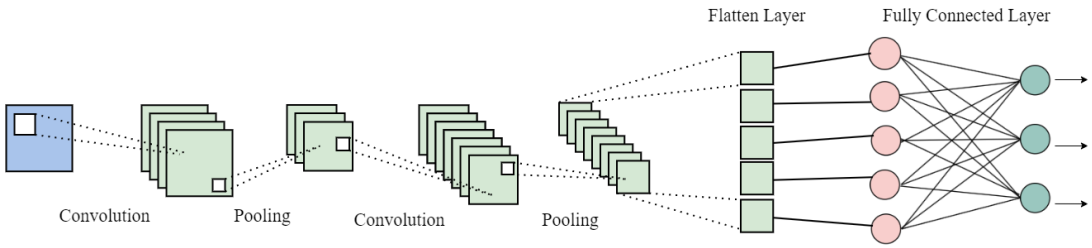


Figure 6: **Convolutional Neural Network Architecture.** The input image passes through convolutional layers that apply filters to extract features, followed by pooling layers to reduce dimensionality. The feature map is then flattened and fed into fully connected layers, which perform the final classification or regression task.

The CNN operates by applying a convolution operation to input data, typically a tensor representing an image with height, width, and channel dimensions. This operation employs a set of filters or kernels, which are small arrays of numerical values. These kernels move across the input tensor, systematically scanning and performing localized feature detection. At each position, the kernel conducts an element-wise multiplication with the corresponding region of the input tensor, producing a series of dot products. These dot products are then aggregated to generate an output value placed in the resulting tensor, termed a feature map or an activation map. The convolution operation is performed repeatedly with multiple kernels, each producing a distinct feature map, effectively extracting diverse features from the input data.

Furthermore, the convolution operation involves hyperparameters that define its behavior, such as the size of the kernels, typically 3×3 or 5×5, and the number of kernels, determining the depth or number of feature maps produced. The padding ensures that the center of each kernel aligns with the outermost elements of the input tensor, while the stride dictates the step size at which the kernel traverses the input. These mechanisms together enable the Convolutional layer to systematically detect and extract various local features across the input data (133; 134).

Following the convolutional layer, the pooling layer often reduces the number of network connections by downsampling and dimensionality reduction on the input data (135). They achieve this by sliding a filter over the feature maps generated by previous layers and either selecting the maximum value (Max Pooling) or computing the average value (Average Pooling) within localized regions. These techniques condense the information, maintaining essential features while reducing computational load and minimizing overfitting risks (136).

The final layer in a CNN is the fully connected layer, responsible for producing the network's output. Here, the extracted features from earlier layers are flattened into a single vector and fed into a densely connected neural network, enabling classification or regression. This layer's activation function, commonly a softmax for classification tasks or linear function for regression, computes the probabilities for different

classes or the numeric value, respectively, making the ultimate decision or prediction based on the learned features (134).

Recurrent Neural Networks: RNNs represent a distinct class of neural networks tailored to handle sequential data (137). Unlike conventional feedforward networks, RNNs possess a unique loop-like architecture, as illustrated in figure 7, to retain memory and process sequences. This attribute makes them particularly adept at tasks involving sequential information, such as natural language processing, speech recognition, and time series analysis (138).

In RNNs, the information flow is not confined to a unidirectional path from input to output. Instead, it moves in a loop, enabling the network to preserve an internal state representing the context of previously seen inputs. This ability to retain historical information allows RNN to model dependencies within sequences and adapt their processing based on past elements, enhancing their understanding of sequential patterns (131).

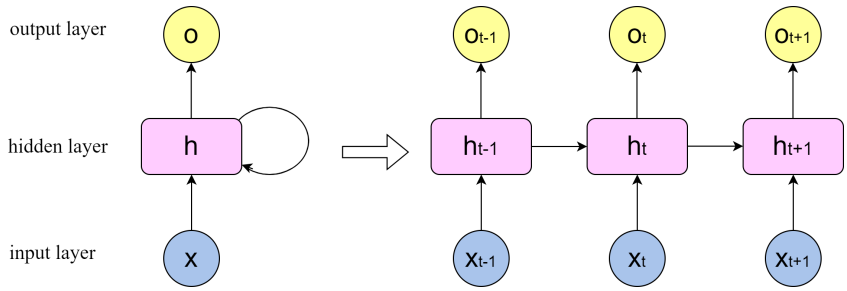


Figure 7: **Recurrent Neural Network Architecture.** The RNN shows how a single cell processes inputs (X) across time steps, passing information to subsequent cells. Each step influences both the output (O) and the hidden state (h), enabling the network to remember past information for tasks like language processing.

However, simple RNNs have a short-term memory, which limits their ability to remember information across long sequences (139). To overcome this, specialized architectures like LSTM (140) and Gated Recurrent Unit (GRU) (141) have been introduced. LSTM is more complex and adept at modeling long-term dependencies, while GRU, being simpler and computationally more efficient, sacrifices a bit of representational power for faster processing.

Autoencoders: Autoencoders were first introduced in 1986 (142) as a neural network designed to reconstruct its input. An autoencoder uses unsupervised learning to learn a representation (encoding) for a set of data, often to lower the data's dimensionality. The network is taught to compress the input into a lower-dimensional code and then reconstruct the output from this representation to resemble the original input as closely as possible (143).

As shown in Figure 8 the encoder has three layers: input, hidden, and output layers. Initially, an encoding function compresses the data from the input layer and stores it in the hidden layer. Notably, the hidden layer, which contains the condensed representation of the initial input, has a significantly lower number of nodes than the input layer. Following that, the output layer attempts to rebuild the compressed data using a decoder function, intending to produce an output that is similar to the original input (142).

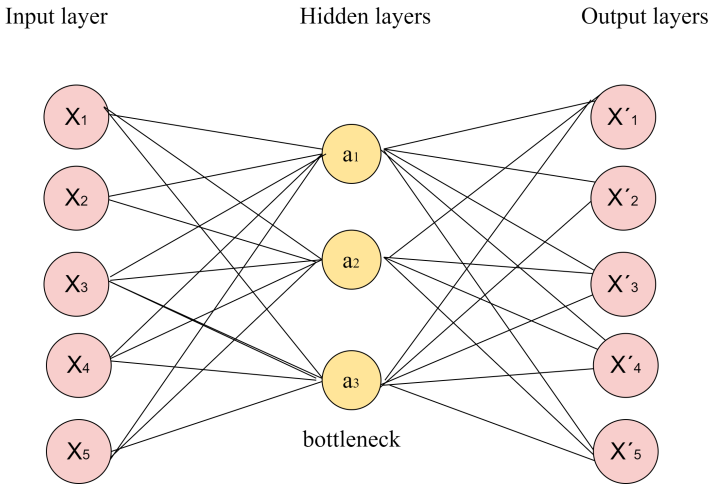


Figure 8: **Autoencoder Architecture.** The network includes an input layer (X1 to X5), a bottleneck (a1 to a3) that encodes key features, and an output layer (X'1 to X'5) that reconstructs the original input. Autoencoders are used for dimensionality reduction and feature learning.

The encoder network has four main parts:

1. **Encoder:** Condenses input data by extracting essential features and mapping them to a lower-dimensional space.

2. **Bottleneck (latent space):** Stores the compressed representation of the input data, containing fewer nodes to capture crucial features.

3. **Decoder:** Reconstructs the encoded data back to a representation resembling the original input by mapping it from the reduced dimensionality back to the initial space.

4. **Reconstruction Loss:** Measures how well a decoder works and how close output is to the original input, guiding the network to minimize this difference during training through backpropagation.

There are many types of autoencoder such as Denoising Autoencoder (DA), Stacked Autoencoder (SAE), and Variational Autoencoder (VAE).

DAs receive input data corrupted by noise, then they are trained to reconstruct a clean version of an input signal, through a process called denoising. This is commonly accomplished by employing a

reconstruction loss function, which computes the difference between the clean input and the rebuilt output. The DAs are trained to minimize this loss by backpropagating the encoder and decoder weights.

SAEs address challenges in datasets with complicated feature interactions. While a single Autoencoder may struggle to adequately reduce dimensionality, SAEs employ multiple layers to learn hierarchical representations of input data, facilitating more efficient feature extraction and reducing overall dimensionality (144).

VAEs function as a generative model employing probabilistic distributions, such as mean and variance parameters in a Gaussian distribution, to create new data instances (145). VAEs introduce probabilistic distributions that enable more flexible and interpretable latent representations. Unlike conventional autoencoders, VAEs provide a more systematic approach to generating data instances by modeling the underlying probability distribution of the data (146).

Transformers: Transformers, first introduced by Vaswani et al. in 2017 (147), represent a groundbreaking shift in DL, especially for handling sequential data like text, speech, or even biological sequences. Unlike traditional models like RNN and LSTM, which rely on sequential processing, transformers use a mechanism called self-attention. This allows the model to process input sequences in parallel, making it faster and more efficient, especially for long sequences where traditional models often struggle with maintaining relationships between distant elements.

At the core of transformers is the self-attention mechanism, which allows the model to focus on different parts of an input sequence simultaneously. In traditional models like RNNs data is processed one token at a time, which makes it difficult for the model to capture long-range dependencies. On the other hand, transformers apply self-attention to the entire sequence all at once, giving them the ability to weigh the importance of each word or token in relation to the others. This means they can capture each word's context, no matter how far apart those words are in the sequence. For example, in a sentence, certain words may have strong connections with others several words away, and transformers are particularly good at recognizing and capturing these connections (147).

A key feature of transformers is multi-head attention. Instead of using just one attention mechanism, transformers apply several simultaneously. Each "head" focuses on a different aspect of the sequence, allowing the model to capture various kinds of relationships within the data. This multi-head attention gives transformers a richer understanding of the input compared to older models (147).

One challenge transformers address is the lack of basic sequence understanding. Unlike RNNs which process data sequentially and thus know the order of inputs, transformers process all inputs simultaneously. To retain the order of the data, transformers use positional encoding. This adds information about

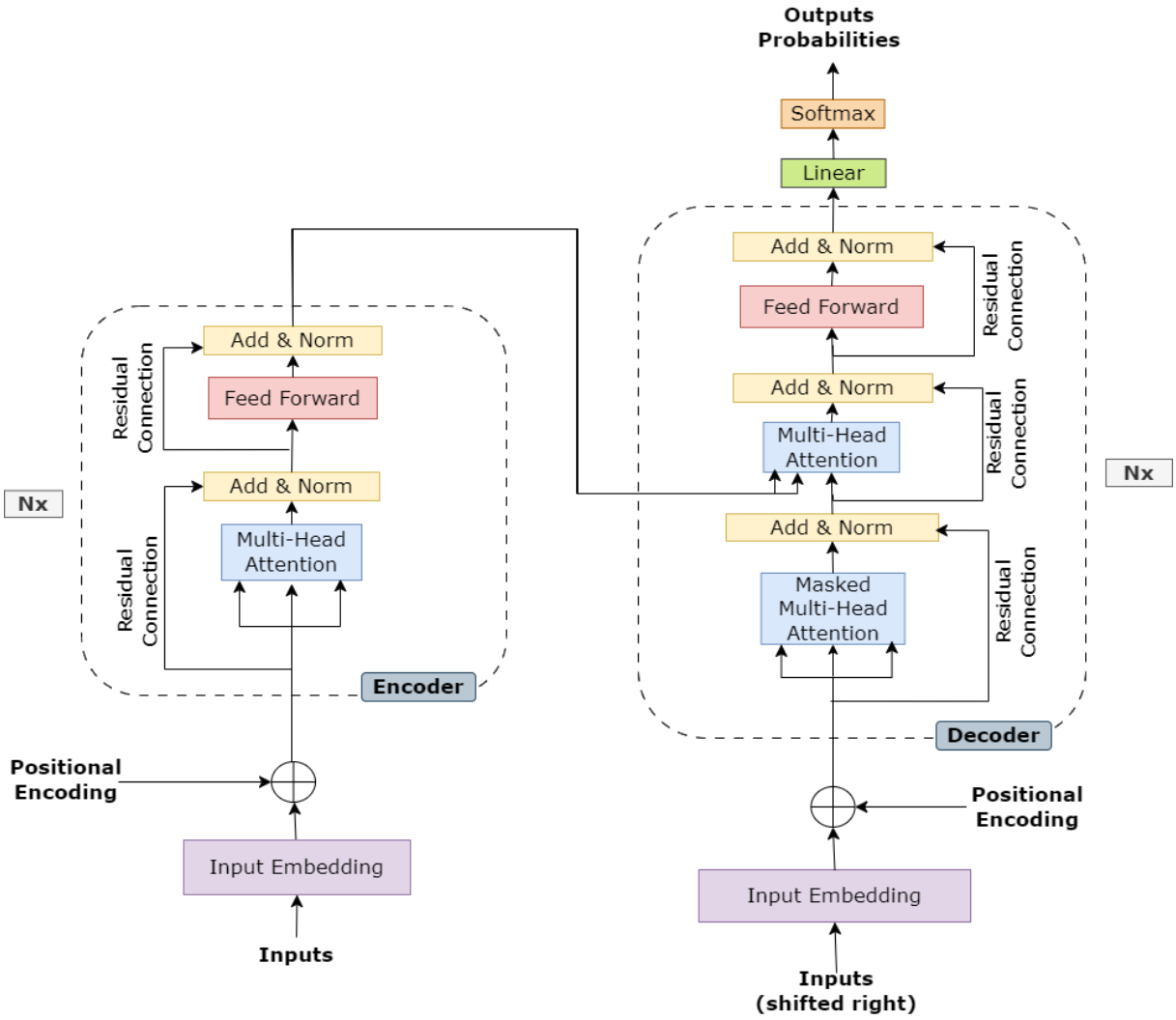


Figure 9: **Transformer Architecture.** The model consists of an encoder and a decoder. The input embeddings are combined with positional encoding, and the encoder applies multi-head attention to capture token relationships. The decoder generates predictions by attending to the encoder's output and previously generated tokens.

the position of each token in the sequences, ensuring the model can still understand the structure and order of the sequences even while processing everything in parallel (147).

Originally, transformers were designed with an encoder-decoder architecture (see Figure 9). In this setup, the encoder processes the input sequence to create a representation that captures its essential features, while the decoder uses this representation to generate the output sequence. However, over time, variations of this architecture have been created to suit different tasks. For example, Bidirectional Encoder Representations from Transformers (BERT) (148) uses only the encoder and excels at tasks like question answering and text classification, while Generative Pre-trained Transformer (GPT) (149) uses only the decoder and is strong in text generation tasks.

One of the major advantages of transformers is their ability to process data in parallel. In RNN and

LSTMs, data must be processed one step at a time, making training slow and limiting how well the model can capture long-term dependencies. Transformers, by processing all tokens at once, dramatically speed up training and are better at capturing relationships between distant elements in a sequence. This efficiency has made transformers the model of choice for large-scale tasks like language translation and text generation.

Looking ahead, research is focused on making transformers more efficient. Approaches like sparse attention, which reduces the number of attention operations, and linear transformers, which lower the memory requirements, are helping scale transformers for even larger tasks and datasets. Additionally, transformers are being applied to more domains, such as protein folding, where their ability to handle long-range dependencies is proving invaluable.

2.5 State-of-the-Art Tools for Workflow Automation

In bioinformatics and computational science, efficiently managing and reproducing data workflows is crucial. Tools such as Nextflow, Docker, and Flask support these needs by streamlining data pipeline management, application deployment, and reproducibility across different environments. Each of these tools addresses specific challenges in workflow automation.

2.5.1 Docker

Docker (150) is an open-source platform designed for packaging applications into containers, which include all necessary components, such as libraries, system tools, and the application code itself. Unlike virtual machines, Docker containers share the host's OS kernel, making them lightweight and efficient, requiring fewer resources and allowing for faster startup times.

One of Docker's main benefits is its portability. A containerized application can run consistently across different environments—whether on a local machine, HPC cluster, or cloud service—without issues related to software version mismatches or configuration differences. Docker operates through a client-server architecture where users interact with Docker via a client, and the Docker daemon manages container creation and execution.

Docker containers are created from images, which define the application environment and its dependencies. The containerization process ensures the application functions independently from the host system, and multiple containers can run concurrently on the same machine without conflicts. Docker also supports networking and persistent storage, allowing containers to communicate and store data outside

their isolated environment.

By integrating with orchestration tools like Kubernetes, Docker enables efficient management and scaling of applications, making it suitable for scenarios requiring the deployment of numerous containers.

2.5.2 NextFlow

Nextflow (151) is a workflow management system designed to handle the complexity of data workflows, particularly in bioinformatics. It allows users to create workflows by connecting independent processes written in different scripting languages like Bash, Python, or R. These processes communicate through channels, enabling parallel execution and efficient data processing.

A key feature of Nextflow is its ability to run the same pipeline across different environments, from local machines to HPC clusters or cloud platforms like AWS and Google Cloud. This is made possible through its execution abstraction, which separates the pipeline's design from the underlying execution system.

Nextflow supports reproducibility in workflows by integrating with tools like Docker and Conda. Docker, in particular, allows each process to be executed in a container, ensuring that dependencies and software versions remain consistent across environments. This capability is especially useful in bioinformatics, where workflows often need to be replicated in different systems.

Nextflow is built on Groovy, a language that extends Java, making it accessible to developers familiar with Java or Python. It allows configuration through a *nextflow.config* file, where users can specify environment settings and pipeline parameters, making it highly customizable for various computational environments.

2.5.3 Flask

Flask (152) is a Python-based micro-framework designed for building web applications. It provides the essential tools needed to create web applications, offering flexibility for developers to extend its functionality as required. Flask follows a minimalistic approach, allowing developers to add only the components necessary for their projects.

Flask routes web requests to specific Python functions, with URL patterns mapped to view functions. When an HTTP request is made, Flask executes the corresponding function and returns a response to the user. This makes Flask well-suited for building web pages, APIs, and interactive applications.

Flask uses the Jinja2 templating engine to generate dynamic HTML content. Templates can include variables and logic, enabling the creation of personalized web pages based on user input or data from a database. Developers can extend Flask with libraries such as Flask-SQLAlchemy for database management

and Flask-Login for user authentication, providing additional functionality for web development projects.

Flask's support for different HTTP methods allows it to handle various interactions, such as form submissions and data retrieval. It is commonly used for creating RESTful APIs, making it a popular choice for backend development in web and mobile applications.

Flask also includes a built-in development server for testing and debugging during development. Its modular nature makes it scalable, from small applications to more complex projects that may require additional features or future expansion.

Nextflow, Docker, and Flask are three essential tools that address different aspects of workflow automation, from building and deploying software to managing complex data pipelines and creating web applications.

Chapter 3

Methodology

This chapter discusses the development of the dissertation pipeline and presents the methods designed for the application of ML and DL to the classification of enzyme cofactor specificity applied to GSM models. The different modules are available at github.com/joanag05/CofactorPredictionTool.

3.1 Data Extration

To develop models for predicting enzyme-cofactor specificity, it was necessary to extract detailed biological data, focusing on enzymes, their sequences, and associated cofactors. This data forms the foundations for training ML and DL models capable of making predictions.

The data extraction process utilized several key biological databases: BRENDA, SABIO-RK, PDB, UniProt, ChEBI, and Rhea. Access to these databases was facilitated through their respective APIs or Python packages.

Initially, various APIs were evaluated for their suitability in retrieving information on enzymes and cofactors. The SABIO-RK REST API (153) provided EC numbers, organisms, and cofactor information; however, it listed cofactors as substrates or products rather than explicitly as cofactors. Additionally, it did not support the retrieval of protein sequences by protein ID, which was a crucial requirement for building the dataset.

The "BRENDApyrser" Python package (154) was used to parse BRENDA's raw files for protein identifiers and cofactor information. However, BRENDA only allowed the extraction of one cofactor per protein ID from text files, requiring additional text mining for comprehensive data collection. Like SABIO-RK, it could not also retrieve protein sequences based on protein IDs.

The RCSB PDB Search API was tested to extract protein sequences and cofactors, but it returned limited results for proteins with cofactors, reducing its applicability for this task.

Given these limitations, the UniProt API emerged as the most suitable for the project. Its well-structured

data format enabled the efficient extraction of detailed information, including protein sequences and respective associated cofactors, which were essential for the later stages of model development. In conclusion, the UniProt API was chosen for its reliability and comprehensive coverage, while the other APIs were deemed less suitable due to their limitations in extracting the necessary information.

3.2 Dataset Construction

The dataset was constructed using the *BuildDataset* class. The cofactors selected for inclusion in the data set are those known to be essential for cell growth: SAM, NAD(H), NADP(H), FAD, Glutatione (GSH), CoA, Tetrahydrofolate (THF), FMN, Menaquinone, Ubiquinone, Plastoquinone, Ferredoxin, and Ferricytochrome (4; 155; 156; 157; 158).

Initially, only UniprotKB reviewed entries were included in the dataset, using ChEBI IDs of the cofactor and the respective redox pair. For example, for NAD, the search included NAD (ChEBI:57540) and NADH (ChEBI:57945), while for NADP, it included NADP (ChEBI:58349) and NADPH (ChEBI:57783). These searches were constructed using the format *chebi:CofactorID AND chebi:RedoxPairID*, ensuring that both oxidized and reduced forms of each cofactor were included. Similar queries were applied for the other cofactors, such as FAD (ChEBI:58307 and ChEBI:57692), and CoA (ChEBI:57288 and ChEBI:57287).

To incorporate unreviewed data and expand the dataset, Rhea reaction IDs were utilized. These Rhea IDs allowed for the retrieval of relevant enzymatic reactions linked to each cofactor, which was particularly useful for cofactors that were underrepresented in the reviewed data. For instance, FAD-related reactions were retrieved using Rhea IDs like 24004, 24052, and others. The Rhea queries were formatted as *rhea:RheaID* to retrieve corresponding UniProt entries, supplementing the dataset with additional sequences that included unreviewed data.

The inclusion of unreviewed data aimed to ensure a more comprehensive representation of underrepresented cofactors, such as FAD, GSH, and Menaquinone. To validate whether adding unreviewed sequences would impact model performance, PCA was performed on the protein embeddings for each cofactor. This analysis visualized the distribution of unreviewed sequences alongside reviewed data, demonstrating that the unreviewed sequences overlapped sufficiently with the reviewed ones, maintaining dataset quality.

The details of specific ChEBI IDs and Rhea reaction IDs used for cofactor retrieval are presented in Table 6, which provides a comprehensive overview of how each cofactor was searched and integrated into the dataset.

The dataset also includes proteins that do not require cofactors, acting as negative examples. This

approach ensured a balanced dataset, allowing the models to distinguish between proteins with and without cofactor requirements. Specific proteins with E.C. numbers 2.1.1.23 and 3.6.1.22 were excluded, as they participate in reactions involving NAD(H) and NADP(H) but do not use them as cofactors.

Table 6: ChEBI and Rhea Reaction IDs for Cofactor Retrieval

Cofactor	ChEBI IDs	Rhea Reaction IDs
SAM	59789, 57856	-
NAD	57540, 57945	-
NADP	58349, 57783	-
FAD	58307, 57692	24004, 24052, 42800, 43448, 43456, 43464, 43780
CoA	57288, 57287	-
THF	67016, 15636	-
FMN	57618, 58210	21620, 17181, 31599
Menaquinone	16374, 18151	27834, 27409, 74079
GSH	57925, 58297	24424, 19057, 48620, 48888, 50708
Ubiquinone	16389, 17976	27405, 63936
Plastoquinone	17757, 62192	30287, 22148
Ferredoxin	33738, 33737	-
Ferricytochrome	29033, 29034	24236, 43108, 45032, 51224, 77903

3.3 Data Preprocessing

Data preprocessing is a critical step in ensuring the quality of the dataset. A dedicated *Preprocessing* class was developed to handle common data issues. This class uses the *ReadSequence* class from the Propythia package to detect and handle sequences with unusual characters, which could potentially disrupt downstream analysis. Additionally, the class filtered out sequences that were either excessively long (greater than 5000 characters) or too short (less than 45 characters), as these could lead to inefficiencies or inaccuracies during processing (159).

After preprocessing, feature extraction was performed to enhance the representation of the protein sequences. This involved using three embedding techniques previous discussed in the Section 2.3.3: ESM2 (94), SeqVec (93), and ProtFlash (95).

Using each embedding method, a unique dataset was generated, offering diverse representations of the protein sequences. This strategy provided the flexibility required to select the optimal dataset for further analysis and model training based on the specific needs of the prediction task.

3.4 Machine Learning Models Construction

To study cofactor-enzyme binding through a MLC approach, a series of ML models were tested. The *skmultilearn* library, which specializes in handling multilabel classification problems, was used for this purpose, while the evaluation metrics were derived from the *scikit-learnlibrary* . The core components of the pipeline included data loading, train-test splitting with iterative stratification, classifier training and hyperparameter tuning, prediction, and results storage.

The ML framework was designed to accommodate various MLC algorithms, each fine-tuned using a specified parameter grid. To ensure effective data splitting into training, testing, and validating sets, the iterative stratification method from the *iterstrat* package was employed as *scikit-learn* does not natively support stratification for multi-label datasets. This method follows the approach described by Sechidis et al. (160), ensuring a balanced representation of labels across splits.

For algorithmic implementation, transformation methods such as LP, BR, and CC, as previously described in 2.3.5, were employed from the *skmultilearn.problem_transform* module, which required a base single-label classification algorithm. In this case, the ensemble method RF was utilized. Additionally, adapted algorithms were implemented using the *MultiOutputClassifier* module from *scikit-learn*, with ExtraTrees, AdaBoost, and XGBoost from the *xgboost* package serving as base classifiers.

Hyperparameter optimization was performed using *GridSearchCV* from *scikit-learn*, with the F1 score used to select the best model. Although other metrics like accuracy, precision, and recall were calculated, the model selection focused on optimizing the F1 score.

The ML models used for MLC, along with their base estimators and the range of hyperparameters tested, are summarized in Table 7.

After evaluating the models, three files were saved for each ML model: 1) a joblib file containing the trained model; 2) a text file documenting the evaluation metrics and confusion matrix for each label; and 3) a Tab-Separated Values (TSV) file containing the true and predicted labels for the test dataset. These files were saved to keep track of the model's performance and predictions for easy access.

Table 7: Summary of ML models employed for MLC and the specific hyperparameters that were fine-tuned during model optimization.

Methods	Models	Hyperparameters
LabelPowerSet, BinaryRelevance, ClassifierChain	RandomForestClassifier	criterion: {'gini', 'entropy'} n_estimators: {10, 20, 30, 50} max_depth: {10, 20, 50} min_samples_split: {2, 5, 10} min_samples_leaf: {1, 2, 4} bootstrap: {True, False}
Multioutput	ExtraTreeClassifier	n_estimators: {50, 100, 200} bootstrap: {True, False} criterion: {'gini', 'entropy'} max_depth: {10, 50, 100, None} min_samples_split: {2, 5, 10} min_samples_leaf: {1, 2, 4} max_features: {'auto', 'sqrt', 'log2'} max_leaf_nodes: {None, 10, 50, 100} min_impurity_decrease: {0.0, 0.1, 0.2}
	XCBClassifier	learning_rate: {0.01, 0.1} gamma: {0.5, 2} subsample: {0.6, 1.0} colsample_bytree: {0.6, 1.0} max_depth: {3, 9} n_estimators: {100, 500}
	AdaBoostClassifier	n_estimators: {50, 100, 200} learning_rate: {0.1, 0.5, 1.0} algorithm: {'SAMME', 'SAMME.R'}

3.5 Deep Learning Models Construction

This methodology section describes the construction and training of a CNN model for cofactor prediction. Initially, other DL architectures, such as DNNs, were tested, but the preliminary results showed better

performance with CNNs. Recent approaches have utilized CNNs effectively for protein sequence analysis due to their capability to capture local patterns and motifs in amino acid sequences, which are critical for predicting functional properties, such as enzyme-cofactor interactions (161; 162; 163; 164). CNNs are particularly effective for MLC tasks like enzyme cofactor prediction, as they can model the relationships between multiple cofactors within a single protein sequence. Given their proven success and the superior results observed compared to DNNs, the decision was made to use CNNs for the final model construction.

The model development process began with setting up data pipelines and defining the CNN architecture, followed by implementing training and evaluation procedures. The dataset for cofactor prediction was split using the iterative stratification method provided by the *iterstrat* package, ensuring that training, validation, and test sets had a consistent label distribution. This consistent split allowed for fair and accurate evaluation of the CNN model throughout the training process.

The dataset was preprocessed using the same protein embeddings employed for the ML models: FlashProt, ESM2, and SeqVec. These embeddings converted protein sequences into high-dimensional vectors that capture essential sequence features. The dataset was then loaded into PyTorch, where it was converted into tensors for efficient computation. PyTorch's DataLoader was used to handle batching and shuffling, with a batch size of 32 to optimize training efficiency and ensure balanced learning.

The CNN architecture, shown in Figure 10, consists of three convolutional layers with progressively increasing filter sizes to capture hierarchical features within the sequences. The first convolution layer uses 32 filters with a kernel size of 5, followed by the second and third layers, which use 64 and 128 filters, respectively, also with a kernel size of 5. A padding of 2 is applied to each convolution layer to maintain the input dimensions, ensuring that the feature extraction process retains important information. Batch normalization is applied after each convolution layer to stabilize the training process, improve convergence, and minimize issues like vanishing or exploding gradients. Max-pooling layers, with a pool size of 2, follow each convolutional layer to down-sample the feature maps and reduce dimensionality while retaining essential information.

After the convolution layer, the network includes fully connected layers to integrate the extracted features. The output of the final convolution layers is flattened and passed through a fully connected layer with 64 units, where a dropout rate of 10% is applied to prevent overfitting. ReLU is used as the activation function for non-linearity, helping to prevent the vanishing gradient problem. For the final output layer, a sigmoid function is used for MLC, providing probabilistic outputs for each class in the MLC task.

The training process was handled by the *train_model* function, where the model was trained using Binary Cross-Entropy with Logits as the loss function and the Adam optimizer for efficient learning. The

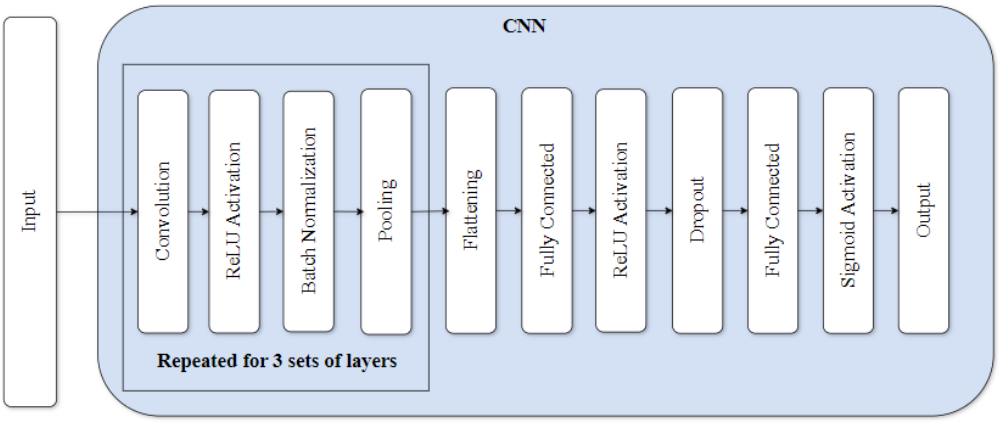


Figure 10: **Convolutional Neural Network (CNN) Architecture**. The model includes convolution, ReLU activation, batch normalization, and pooling layers, followed by fully connected layers and a Sigmoid activation for output.

learning rate was initialized at 1×10^{-5}, with a weight decay of 1×10^{-5} to reduce overfitting. A learning rate scheduler with a step size of 5 and a decay factor (gamma) of 0.5 was employed to adjust the learning rate during training, enhancing convergence as the model progressed.

A test was performed with various numbers of epochs, and it was observed that after 50 epochs, the model's performance no longer showed any significant improvements. Therefore, the training was conducted over 50 epochs, with early stopping implemented to avoid overfitting, ensuring the model did not train on noise after achieving the best performance on the validation set. Each epoch involved both training and validation phases, allowing the model's performance to be continuously monitored and adjusted.

After training, the model was evaluated on the test set using metrics such as F1 score, precision, recall, accuracy, and hamming loss. The *test_model* function was used to generate and analyze predictions, with a focus on assessing the model's performance across all labels. Additionally, threshold optimization was applied using the *select_threshold*s function to maximize the F1 score by determining the optimal threshold for converting predicted probabilities to binary labels.

The process resulted in three key outputs for each model: a ".pth" file containing the trained model, a text file documenting the evaluation metrics and confusion matrix for each label, and a TSV file with the true and predicted labels for the test dataset.

Hyperparameter Optimization

Hyperparameter tuning for the CNN model was performed using the Optuna framework (116). The key hyperparameters optimized included learning rate, weight decay, step size, and gamma.

Optuna executed multiple trials, each testing a different set of hyperparameters. During each trial,

the model was trained and validated over 50 epochs, with early stopping applied if performance ceased to improve. The results from these trials guided the selection of optimal hyperparameters for subsequent trials, ensuring progressive improvement.

The optimization process spanned 200 trials, thoroughly exploring the hyperparameter space. The best trial identified settings that significantly enhanced the F1 score, which was then applied in the final model training to achieve the highest performance.

3.6 Cofactor-Predicton Workflow

This section of the methodology describes the pipeline developed using NextFlow (151) to process sequence data, compute embeddings, make predictions, and update a GSM model, as illustrated in Figure 11.

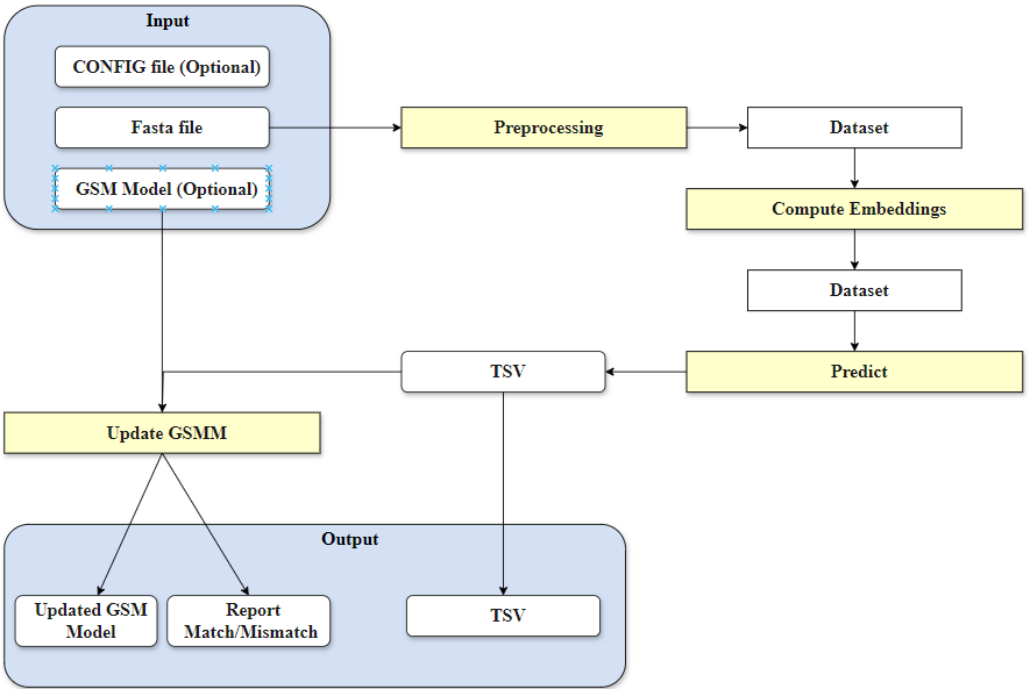


Figure 11: **Cofactor-Prediction Workflow Pipeline** The pipeline starts with optional input files (FASTA, GSM model, CONFIG). After preprocessing, embeddings are computed, and predictions are made. If a GSM model is provided, it is updated with the predictions. The output includes an updated GSM model, a match/mismatch report, and a TSV file with predictions.

The pipeline begins with three input files: 1) a FASTA file containing sequences for the genes present in the model; 2) an optional GSM model in Systems Biology Markup Language (SBML) format; and 3) an optional configuration (CONFIG) file in JavaScript Object Notation (JSON) format specifying the particular settings and parameters for the pipeline. The FASTA file is required as it contains the protein sequences to

be analyzed, while the GSM model and CONFIG file can be omitted depending on the desired functionality.

If provided, the SBML file supplies the structural and functional details of the GSM model, which can be updated based on the ML model's predictions. The optional CONFIG file contains important configuration data, such as paths to input files, model parameters, and mappings of cofactors to specific reactions and their corresponding compartments within the cell. For example, the CONFIG file might map cofactors like NAD(H), NADP(H), and FAD to specific metabolite pairs located in different compartments such as the cytosol, mitochondria, and peroxisomes. This compartmental specificity ensures that the predictions reflect the correct subcellular localization of these biochemical processes. If the CONFIG file is not provided, default settings will be used. Default settings are predefined to use BiGG identifiers in all cellular compartments currently available in this database.

The first stage of the pipeline involves preprocessing the FASTA file using the *Preprocessing* class described in Section 3.3. The sequences are cleaned and formatted before being passed to the next step, where embeddings are computed using the *EmbeddingsESM2* class, also detailed in 3.3. Nextflow organizes these steps so that data flows smoothly from one step to the next, and each task is done in the right order and environment.

Next, the pipeline leverages the selected trained model to predict cofactor binding based on the generated embeddings. These predictions, including binary outcomes and probability scores, are saved in TSV format. If a GSM model is provided, the pipeline updates the model accordingly and generates a JSON report of, detailing the match and mismatch counts as well as their corresponding percentages.

The GSM model update process offers different possibilities: the pipeline can add new reactions to the model to represent the predicted results while retaining the original reactions for reference. Alternatively, it can modify existing reactions to reflect the updated predictions better. In cases where certain reactions no longer align with the predicted metabolic activities, the pipeline can remove these reactions, thereby refining the model to match better the ML model's predictions. These options can be defined in the CONFIG file.

3.7 Web Service for Cofactor Prediction

The web service for cofactor prediction was built to work directly with *merlin*, an open-source platform designed for the reconstruction of GSM models. *merlin* facilitates the annotation of genomes with enzymatic and transport functions, enabling the automated integration of relevant biological data into draft metabolic models.

merlin's remote plugins are executed through web services - *merlin*-web-services- organized in docker containers (workers) controlled by a main manager docker. Such an organization allows managing different user's submissions in queues, avoiding overloading the server.

The cofactor prediction web service, built using Flask, offers a user-friendly platform for predicting enzyme-cofactor specificity. Users can upload protein sequences in FASTA format, with optional input for a GSM model in SBML format and a CONFIG file in JSON format for custom settings. If the optional files are not provided, the system uses default configurations to perform the predictions.

The backend of the web service is driven by a Nextflow pipeline, as described in 3.6, which automates key stages in the prediction workflow.

Upon completion, the system generates a ZIP file containing a TSV file with the predicted cofactors and, if provided, an updated GSM model. This ensures users receive the predicted enzyme-cofactor interactions along with any necessary updates to their metabolic models.

3.8 Case Studies

To validate the pipeline, case studies were retrieved from BiGG Models database using the BiGG Models API. A function, *get_model_info_from_bigg*, sends a request to the BiGG Models API to retrieve a list of available models. Once the model information is retrieved, the *select_model* function is used to select models from different organisms. If multiple models exist for the same organism and strain, the model with more genes is selected, and the respective SBML file is downloaded. Furthermore, the protein sequences for each gene in the model are also obtained from the BiGG Models database and compiled into a FASTA file.

Once case study files are prepared, they are integrated into the pipeline described in 3.6 to update the GSM models based on the predictions. The process begins by matching the FASTA and SBML files for each model. For each matching pair, the number of genes in the FASTA file is compared with the number of genes in the model to ensure sufficient overlap. If the FASTA file contains more than half of the genes present in the model, the pair is considered valid for analysis. Additionally, if the FASTA file does not contain any protein sequences, it is rejected from the analysis.

The Nextflow pipeline is executed for each valid case study using the FASTA file, the corresponding SBML, and a configuration file that specifies the pipeline parameters. After executing the pipeline, a set of output files are generated: an updated SBML model, prediction results in a TSV format, and a JSON report.

Chapter 4

Results and Discussion

4.1 Dataset Overview and Characterization

The dataset used to build the ML and DL models was compiled from the database UniProt. Comprising 73,307 protein sequences, the dataset includes annotations of up to 13 different cofactors, namely NAD(H), NADP(H), FAD, CoA, THF, FMN, Menaquinone, SAM, GSH, Ubiquinone, Plastoquinone, Ferredoxin, and Ferricytochrome. These annotations serve as MLC targets for the models, with the dataset chosen to ensure complete coverage of proteins with relevant cofactor information.

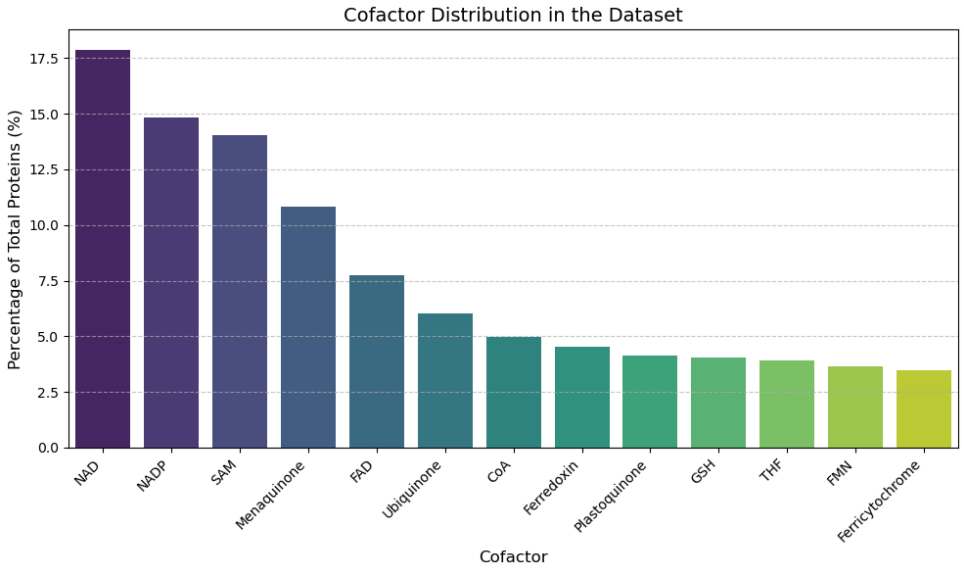


Figure 12: **Cofactor Distribution in the Dataset** This figure shows the percentage of different cofactors across the protein sequences, highlighting the varying distribution, with some cofactors appearing much more frequently than others.

An analysis of the cofactor distribution within the dataset reveals a noticeable imbalance, as illustrated in Figure 12 where the frequency of each dataset was calculated. NAD(H) is the cofactor that occurs most frequently, found in approximately 17.5% of the proteins. This is followed by NADP(H) and SAM, present

in about 15% and 13% of the proteins, respectively. However, cofactors such as Ferricytochrome, FMN, and THF are much less common, each associated with only about 5% or fewer proteins. This imbalance challenges model training, particularly in accurately predicting the less common cofactors.

The multilabel nature of the data set introduces a layer of complexity, as several proteins are associated with multiple cofactors (165; 166; 167; 168). Specifically, 7,521 proteins are annotated with more than one cofactor, while 55,771 proteins have only one associated cofactor.

The multi-label aspect reflects the intricate relationships between different cofactors within the same protein sequences, which the models must learn to predict accurately. This presents a challenge, as certain cofactors are not independently associated with proteins but instead co-occur because of functional or metabolic dependencies.

For instance, NAD(H) and NADP(H) co-occur in 3,863 proteins, representing 29.83% of proteins annotated with NAD(H) and 35.98% of proteins annotated with NADP(H). This significant overlap indicates interdependency between these cofactors. Both NAD(H) and NADP(H) are crucial electron carriers in cellular redox reactions, with NAD(H) predominantly involved in catabolic processes and NADP(H) in anabolic reactions. This simultaneous occurrence points to potential complementary functions or a collaborative regulatory system in specific proteins that require both cofactors for maintaining equilibrium in cellular activities.

To illustrate these relationships, Figure 13 presents the Pearson correlation analysis between different cofactors. The co-occurrence patterns previously observed (not shown here) were further explored using this correlation heatmap to quantify the strength of the linear relationships.

The Pearson correlation analysis (Figure 13) indicates a moderate positive correlation (0.20) between NAD(H) and NADP(H), supporting the observed co-occurrence of these cofactors and implying that they frequently coexist as alternatives in enzyme reactions. The correlation heatmap provides a more concise and statistically meaningful view of cofactor relationships. For instance, SAM shows negative correlations with redox cofactors such as NAD(H) (-0.17) and NADP(H) (-0.16), suggesting that SAM is involved in different processes, like methylation reactions, which are separate from the redox functions of NAD(H) and NADP(H).

These relationships indicate that predictive models must not only identify the presence of a specific cofactor but also recognize when multiple cofactors may bind to the same protein. This distinction is important for accurate predictions regarding cofactor usage, emphasizing the need for models that can account for both single and multiple cofactor scenarios.

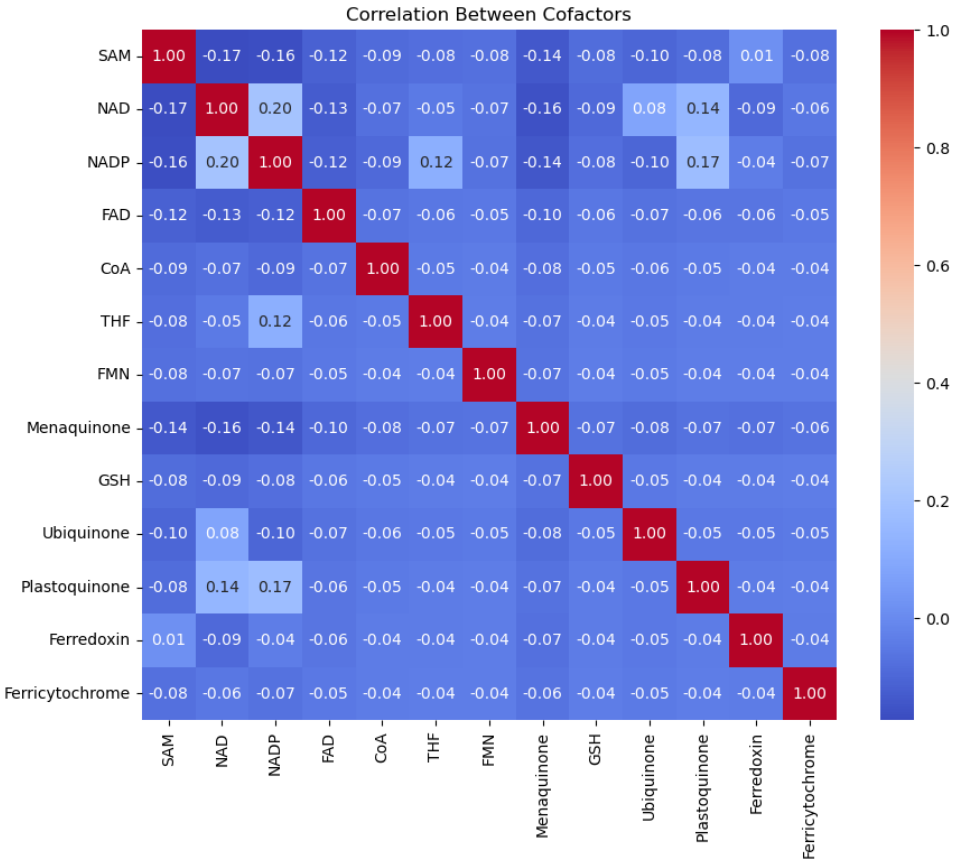


Figure 13: **Cofactor Relationships in Protein Sequences.** The heatmap displays the Pearson correlation between different cofactors. Positive correlations (red) suggest cofactors often appear together in proteins, while negative correlations (blue) indicate they rarely co-occur.

4.2 Performance Evaluation of Machine Learning Models

The dataset was split into training, validation, and test sets, ensuring a distribution of cofactor labels across all three subsets. Table 8 shows the counts of each cofactor in the train, validation, and test sets. It highlights that certain cofactors, such as NAD(H) and NADP(H), are highly represented in both sets, while others, like CoA and Ferricytochrome, are less frequent. This distribution is important when evaluating the models' performance, as it can affect recall and precision, especially for underrepresented cofactors.

The validation set was used during model training to tune hyperparameters and assess model generalization. By monitoring performance on the validation set, we aimed to ensure that the models were not overfitting to the training data and were capable of generalizing well to unseen data. This step is crucial, as a model that performs well on the training set but poorly on the validation set would likely underperform when applied to the test set.

In the initial phase of model training, only UniProtKB-reviewed data was used to ensure a high-quality dataset. Reviewed entries are manually curated, providing reliable annotations for enzyme-cofactor inter-

Table 8: Label Counts in Train, Validation, and Test Sets

Labels	Train Set Counts	Validation Set Counts	Test Set Counts
NAD	8791	1601	2559
NADP	7316	1258	2163
FAD	3797	674	1136
SAM	6914	1233	2017
CoA	2451	443	718
THF	1978	324	536
FMN	1778	316	533
Menaquinone	5370	916	1539
GSH	2033	333	576
Ubiquinone	2933	559	860
Plastoquinone	2084	345	551
Ferredoxin	2201	392	676
Ferricytochrome	1726	279	502

actions, which was expected to yield high precision, particularly for common cofactors.

However, this approach revealed some limitations, especially with respect to recall for less frequent cofactors.

When evaluating the model's performance using only reviewed data, it became clear that while precision was high, recall dropped significantly for certain cofactors. For example, the F1 score for NADH was 0.88, but it fell to 0.73 for CoA and 0.81 for FMN. Additionally, cofactors like FAD, GSH, and Menaquinone had no true positives, highlighting their underrepresentation in the dataset when only reviewed data was used. Full details of the model's performance with reviewed data are available in the "reviewed" folder on the GitHub repository.

To address this imbalance, unreviewed data for underrepresented cofactors—particularly FAD, GSH, and Menaquinone—was incorporated, significantly improving the model's ability to predict these cofactors. A PCA analysis was performed to check if the unreviewed sequences overlapped with the reviewed ones, ensuring that adding this data wouldn't negatively affect the model's performance.

With the inclusion of unreviewed data, the dataset became more diverse and better balanced, particularly for the previously underrepresented cofactors. This resulted in improved recall for cofactors like FAD and GSH. The PCA analysis, which can be found in the Appendix A or in the PCA folder on the GitHub repository, confirmed that the unreviewed data did not introduce unnecessary noise, further validating this approach.

Additionally, in this section, the performance of different state-of-the-art multi-label ML models will be

evaluated using three distinct protein sequence embeddings: ESM2, SeqVec, and ProtFlash. Besides evaluating their predictive ability, I also assessed the execution time necessary to generate each embedding, as this factor is critical in large-scale applications.

The performance of each model was assessed using three key metrics: Weighted F1 Average, F1 samples average, and Hamming loss. These metrics provide insights into the models' ability to handle MLC tasks and their robustness in predicting enzyme cofactor binding.

Although key performance metrics are provided here, detailed results, such as confusion matrices, classification reports, and other evaluation metrics, can be found in the resultsML folder on GitHub. Such supplementary information offers a more complete knowledge of each model's performance across all labels.

Execution Time Comparison and Efficiency

Beyond the predictive accuracy of the models, computational efficiency plays a significant role, especially for large-scale datasets like the 73,307 protein sequences used in this study. The execution time required to generate embeddings differs substantially across the three methods—ProtFlash, SeqVec, and ESM2.

ProtFlash, the most compact of the embeddings, was completed in approximately one day, offering a balance between speed and acceptable accuracy. SeqVec took roughly three days to generate, providing better accuracy at a longer computational cost. ESM2, which produced the most detailed embeddings and led to the best model performance, required around one week to process the full dataset.

To quantify this difference, a "time per sequence" metric was calculated, as shown in Table 9, which highlights the trade-off between accuracy and computational efficiency. While ESM2 yielded the highest F1 Scores and lowest Hamming Loss, its significant time cost per sequence (0.229 hours) may not always be feasible for time-sensitive projects. ProtFlash, with a processing time of only 0.0328 hours per sequence, is a more practical option when speed is a priority and small reductions in accuracy can be tolerated.

Table 9: Execution Times for Generating Embeddings

Embedding Model	Total Execution Time	Time per Sequence (hours)
ProtFlash	1 day	0.0328
SeqVec	3 days	0.0984
ESM2	7 days	0.229

4.2.1 ESM2 Embeddings

Table 10: **Performance Evaluation of Machine Learning Models Using ESM2 Embeddings.** This table compares the performance of machine learning models applied to MLC tasks based on key metrics: Hamming Loss, Weighted F1 Average, and F1 Samples Average.

ML Model	Hamming Loss	Weighted F1 Average	F1 Samples Average
Label PowerSet	0.00420	0.97	0.83
BinaryRelevance	0.00650	0.95	0.80
ClassifierChain	0.00653	0.95	0.80
AdaBoost	0.00940	0.94	0.80
XGBClassifier	**0.00390**	**0.97**	**0.84**
ExtraTreesClassifier	0.00687	0.95	0.79

The ESM2 embeddings encapsulate extensive evolutionary data derived from protein sequences. These embeddings are presented in a high-dimensional format with 73,307 rows and 1,294 columns, providing rich context but requiring significant computational resources.

The Label PowerSet model demonstrated strong performance, as indicated in Table 10, with the top Weighted F1 Average score of 0.97. An F1 samples average of 0.83 signifies that it also proficiently managed MLC, accurately predicting several cofactors for each protein.

The MultiOutputClassifier (XGBClassifier) exhibited the smallest hamming loss at 0.00390, indicating it produced minimal errors. It also achieved high F1 scores, with a weighted average of 0.97 and a samples average of 0.84, indicating that it was precise and able to minimize incorrect label predictions. Other models like BinaryRelevance and ClassifierChain performed well but had slightly higher error rates and lower F1 scores.

4.2.2 SeqVec Embeddings

SeqVec embeddings capture sequence-level information and are somewhat less complex than ESM2 (73,307 rows and 1,038 columns). They are faster to generate and offer a good balance between detail and computational efficiency.

As seen in Table 11, the Label PowerSet model also exhibits a good performance with SeqVec embeddings, achieving a Weighted F1 Average of 0.97. Its F1 samples average of 0.83 showed that it handled MLC well, similar to its performance with ESM2 embeddings. Notably, it achieved the lowest hamming

Table 11: **Performance Evaluation of Machine Learning Models Using SeqVec Embeddings.** This table compares the performance of machine learning models applied to MLC tasks based on key metrics: Hamming Loss, Weighted F1 Average, and F1 Samples Average.

ML Model	Hamming Loss	Weighted F1 Average	F1 Samples Average
Label PowerSet	**0.00433**	**0.97**	**0.83**
BinaryRelevance	0.01486	0.89	0.71
ClassifierChain	0.01535	0.88	0.71
AdaBoost	0.01089	0.93	0.79
XGBClassifier	0.00459	0.97	0.83
ExtraTreesClassifier	0.00668	0.95	0.80

loss at 0.00433, indicating a high accuracy in label prediction.

The MultiOutputClassifier (XGBClassifier) also performed had good results, with a hamming loss of 0.00459 and matching the Label PowerSet model´s weighted f1 average of 0.97 and F1 samples average of 0.83. This shows that both models are highly effective when using SeqVec embeddings.

Other models like BinaryRelevance and ClassifierChain showed comparatively lower performance, with higher Hamming losses and lower F1 scores. suggesting that simpler embeddings were more challenging for these models. This suggests that these models may find simpler embeddings more challenging, highlighting the advantage of using models like LabelPowerSet and XGBClassifier for this type of data.

4.2.3 FlasProt Embeddings

Table 12: **Performance Evaluation of Machine Learning Models Using FlasProt Embeddings.** This table compares the performance of machine learning models applied to MLC tasks based on key metrics: Hamming Loss, Weighted F1 Average, and F1 Samples Average.

ML Model	Hamming Loss	Weighted F1 Average	F1 Samples Average
Label PowerSet	**0.00675**	**0.95**	**0.82**
BinaryRelevance	0.01687	0.87	0.70
ClassifierChain	0.01737	0.87	0.69
AdaBoost	0.01412	0.90	0.76
XGBClassifier	0.00727	0.95	0.81
ExtraTreesClassifier	0.00922	0.94	0.78

ProtFlash embeddings capture key protein motifs and domains and are the most compact of the three

(73,307 rows and 782 columns). Due to their smaller size, they can be processed more quickly, which can be useful in situations where faster analysis is needed or when computational resources are limited. While the speed of processing ultimately depends on the hardware available to the user, these embeddings are more efficient, allowing for quicker handling of large datasets.

Table 12 shows that the Label PowerSet model achieved both the highest Weighted F1 Average (0.95) and the lowest Hamming loss (0.00675), indicating it performed exceptionally well with the ProtFlash embeddings. The MultiOutputClassifier (XGBClassifier) also performed strongly, with a Weighted F1 Average of 0.95 and a Hamming loss of 0.00727, slightly higher than Label PowerSet. Other models like ClassifierChain and BinaryRelevance struggled more with ProtFlash embeddings, showing lower F1 scores and higher error rates.

4.2.4 Discussion

ESM2 Embeddings

ESM2 embeddings provided the most detailed representation of the protein sequences, leading to the best overall performance. Both the Label PowerSet and XGBClassifier models showed high F1 scores and low Hamming loss, making ESM2 the most accurate choice for predicting cofactor binding. While these embeddings take longer to generate, the focus here is on achieving the best results rather than speed.

SeqVec Embeddings

SeqVec embeddings performed well, offering a good balance between detail and efficiency. The Label PowerSet and XGBClassifier models still achieved strong results, though slightly below ESM2. These embeddings are faster to generate, making them a viable option when time is a factor, though accuracy remains the main priority.

ProtFlash Embeddings

ProtFlash embeddings were the quickest to generate, though they performed slightly below ESM2 and SeqVec. Despite the small drop in accuracy, they still provided reliable predictions. ProtFlash can be useful when faster results are needed, but for this study, precision is key, making it a secondary option.

The choice of embedding depends on the balance between accuracy and processing time. For this study, where the goal is to make the best predictions, ESM2 stands out as the preferred option. SeqVec and ProtFlash are faster but come with slight compromises in accuracy. Ultimately, achieving the best

results matters most, and for that, ESM2 is the most suitable.

4.3 Performance Evaluation of Deep Learning Models

This section presents the performance of the CNN model before and after hyperparameter optimization, using three types of protein sequence embeddings: ESM2, SeqVec, and ProtFlash. The dataset was split into training, validation, and test sets following the same strategy used in the evaluation of the machine learning models (Section 4.2), ensuring a distribution of cofactor labels across all subsets. The model's performance was evaluated based on key metrics such as Hamming Loss, Weighted F1 Average, and F1 Samples Average, as shown in Tables 13 and 14.

Before hyperparameter optimization, the model achieved moderate results (Table 13). ESM2 embeddings produced the best performance, with an F1 Samples Average of 0.74, while ProtFlash showed the lowest performance, with a score of 0.64. This suggests that the richer embeddings provided by ESM2 captured the relationships between protein sequences and cofactors more effectively. ProtFlash, being simpler, struggled to deliver the same level of accuracy.

Table 13: Performance Evaluation of CNN Models (Before Hyperparameter Optimization)

Embedding	Hamming Loss	Weighted F1 Average	F1 Samples Average
ESM2	0.0135	0.90	0.74
SeqVec	0.0160	0.88	0.72
ProtFlash	0.0236	0.81	0.64

After hyperparameter optimization, the model's performance improved significantly, as shown in Table 14. Hamming Loss decreased across all embeddings, meaning fewer errors, and the F1 Samples Average increased across the board. ESM2 continued to perform best, now with an F1 Samples Average of 0.85, while ProtFlash also saw a noticeable improvement, raising its score to 0.83. These results highlight the value of optimization in fine-tuning model performance, even for simpler embeddings like ProtFlash.

Table 14: Performance Evaluation of CNN Models (After Hyperparameter Optimizations)

Embedding	Hamming Loss	Weighted F1 Average	F1 Samples Average
ESM2	**0.0025**	**0.98**	**0.85**
SeqVec	0.0031	0.97	0.84
ProtFlash	0.0044	0.98	0.83

The improvement was driven by optimized hyperparameters, as shown in Table 15. For instance, ESM2 benefited from a low learning rate (0.0011706) and minimal weight decay (6.13e-09), which helped the model generalize well. ProtFlash, although initially the lowest performing, showed considerable improvement with tuning, making it a viable option for scenarios that need faster computations without sacrificing too much accuracy.

Table 15: Best trial results and hyperparameters for each model

Embedding	F1 samples average	Learning Rate (lr)	Other Hyperparameters
ESM2	0.85	0.0011706	Weight Decay: 6.13e-09 Step Size: 5 Gamma: 0.85
SeqVec	0.84	0.0006278	Weight Decay: 1.25e-08 Step Size: 10 Gamma: 0.70
ProtFlash	0.83	0.0021557	Weight Decay: 4.58e-08 Step Size: 7 Gamma: 0.85

Threshold Selection for Each Embedding Model

To determine the best threshold for each embedding model, the relationship between F1 Score, Precision, and Recall was evaluated across thresholds ranging from 0.1 to 0.9. Figure 14 shows the results for the ESM2 model, while the analysis for SeqVec and ProtFlash can be found in the Supplementary Material (Appendix A Figures A and B).

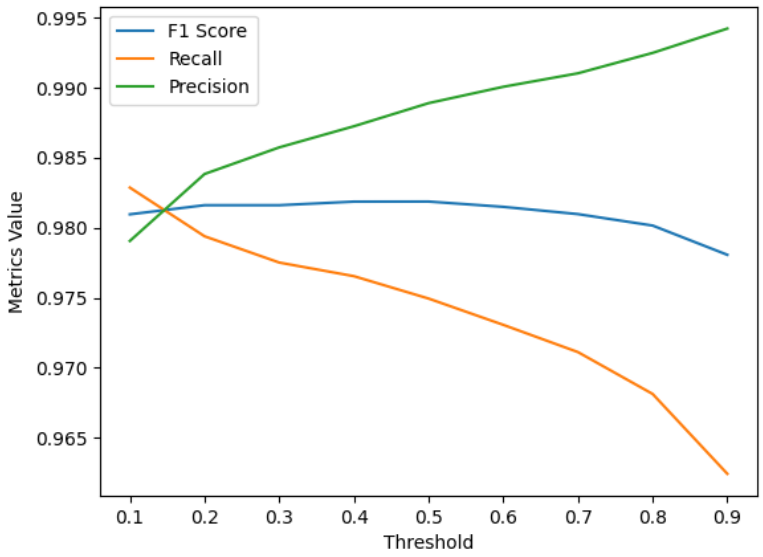


Figure 14: F1 Score, Precision, and Recall at Varying Thresholds for ESM2 embeddings.

For all models, the F1 Score reached its highest point around a threshold of 0.5, with precision increasing at higher thresholds and recall decreasing. After weighing these factors, a threshold of 0.5 was chosen as it provides a good balance between Precision and Recall, ensuring a strong F1 Score. This is particularly important for MLC tasks like cofactor prediction, where accuracy in both positive and negative predictions is critical.

Selecting 0.5 as the threshold helps avoid overly conservative (which would reduce recall) or permissive (which would lower precision) predictions, making it the most suitable choice for consistent and reliable results across different embedding models.

Given the strong performance of the ESM2-based model, it was selected for the case studies due to its low error rate and high accuracy in predicting enzyme cofactor binding across various organisms. ESM2 consistently outperformed the other embeddings, achieving the highest F1 Samples Average and lowest Hamming Loss after optimization, making it the ideal choice for in-depth analysis.

4.4 Case Studies

The cofactor prediction pipeline, developed as part of this work, was evaluated across 84 case studies, each representing a different organism or strain. Nine models, including iAM_Pk459 and Recon3D, were excluded due to missing protein sequences. For the remaining 75 case studies, Table 17 provides a summary of reactions and sequence data, while Appendix A presents more detailed results, including match and mismatch percentages for cofactor predictions.

Table 17 summarizes the cofactor prediction results for the remaining 75 case studies. It includes details such as the total number of reactions, the number of reactions involving cofactors, the number of reactions with available sequence data, and the number of missing sequences in reactions involving cofactors for each GSM model.

Additionally, Table 18, provided in Appendix A, presents a more detailed view of the cofactor binding predictions for each model. This table includes both the counts and percentages of reactions that either matched or mismatched the predicted cofactors, offering a clearer perspective on the pipeline's performance.

Table 18 shows the diverse range of organisms that we have used, including 59 bacterial models, 5 eukaryotic models, 2 archaeal models, and 1 cyanobacterial model. Among the bacterial models, Gram-negative bacteria dominate, with 55 models, while only 4 represent Gram-positive species, such as *Bacillus subtilis*, *Clostridium ljungdahlii*, and two strains of Staphylococcus aureus. This distribution

provided a broad basis for testing the pipeline across various species and metabolic pathways.

The case studies cover a broad range of organisms, primarily Gram-negative bacteria, with notable representation from *E. coli* and *Acinetobacter baumannii*. These bacterial models consistently showed high match rates, with match percentages between 83.2% and 91.3%. The model's strong performance for these species is likely due to the abundance of well-annotated sequences in databases like UniProt, which were used for training. For instance, *Yersinia pestis* CO92 (iPC815) achieved a match percentage of 91.0%, further confirming the pipeline's effectiveness in bacterial models.

Challenges arose with organisms having more complex or unique metabolisms, such as *H. pylori* (iLT341) and *Mycobacterium tuberculosis* (iEK1008). The iLT341 model for *H. pylori*, with a match rate of 62.5%, struggled with predictions, likely due to the distinct cofactor needs of its specialized metabolism. Imbalances in the training dataset, where NAD(H) and NADP(H) were overrepresented, may have skewed predictions for these underrepresented cofactors in more specialized organisms.

To further explore some model mismatches, three GSM model present in table 16 were studied in detail using literature and databases such as BRENDA and UniProt: the two worst-performing models iSynCJ816, iIT341 and one randomly selected *E. coli* model, the iML1515. The detailed results are available at case studies analysis folder.

Table 16: Detailed Study of Three GSM Models Using Literature and Databases

GSM model	Organism	Match	%	Mismatch	%
iSynCJ816	*Synechocystis* sp. PCC 6803	105	66.9	52	33.1
iIT341	*Helicobacter pylori* ATCC 26695	45	62.5	27	37.5
iML1515	*Escherichia coli* str. K-12 substr. MG1655	213	84.9	38	15.1

Discussion of iLT341 Model (Helicobacter pylori ATCC 26695)

The iLT341 model for *H. pylori* presented significant challenges, with a higher mismatch percentage compared to other models. One key reason for this is the complex and unique metabolic pathways of *H. pylori*. Unlike well-characterized organisms like *E. coli*, *H. pylori* relies on specialized enzymes and cofactors suited to its niche within the human stomach. These less common cofactors likely led to incorrect predictions based on the model's training data, which favored more frequently observed bacterial patterns.

For example, the gene *HP0056* was predicted to use both NADP(H)(56%) and NAD(H)(47%), with UniProt identifying NAD(H) as the correct cofactor. This close confidence split suggests that the enzyme might interact with either cofactor depending on conditions, a flexibility acknowledged by databases like

BRENDA. The model's threshold favored NADP(H), though the enzyme likely uses both, depending on species-specific factors. Similar dual predictions were made for *HP0510* and *HP0822*, further illustrating the enzyme-cofactor variability.

The genes *HP0195* and *HP0558*, involved in fatty acid biosynthesis, interact with different cofactors during various stages of the process. According to UniProt, *HP0195* plays a crucial role in reducing a carbon-carbon double bond in an enoyl moiety linked to an Acyl Carrier Protein (ACP), which is essential in fatty acid elongation. UniProt identifies NAD(H) as the primary cofactor for *HP0195*, but the model predicted NADP(H) with high confidence, possibly reflecting the enzyme's ability to bind NADP(H) in specific species or conditions.

For *HP0558*, the model predicted CoA as a cofactor, which aligns with its role in transferring acyl groups, a process that typically requires CoA. The different cofactor predictions for these proteins—NADP(H) for *HP0195* and CoA for *HP0558*—illustrate the model's recognition of their flexibility in cofactor usage, depending on the reaction they are facilitating at different stages of fatty acid biosynthesis.

The genes *HP1273*, *HP1266*, *HP1269*, and others are critical components of NDH-1 complex, a membrane-bound oxidoreductase involved in electron transport. In *H. pylori*, this complex primarily transfers electrons from NAD(H), via FMN and iron-sulfur (Fe-S) centers, to plastoquinone in the respiratory chain. This electron transfer generates a proton gradient across the cytoplasmic membrane, which is essential for ATP synthesis, with NAD(H) serving as the main electron donor, as confirmed by both UniProt and BRENDA databases.

While the GSM model attributed NADP(H) as the cofactor for some of these genes, the correct cofactor in *H.pylori* is NAD(H), reflecting its core role in energy conservation through electron transport. The model's prediction of NADP(H) may indicate flexibility in some organisms to use different cofactors under specific conditions, but for the NDH-1 complex, NAD(H) is the biologically relevant cofactor. This highlights the model's sensitivity to species variability, although NAD(H) remains the primary cofactor for the NDH-1 complex in most cases.

In conclusion, the iLT341 model for *H. pylori* illustrates challenges in predicting cofactors for organisms with unique metabolic pathways. While the model performed well overall, discrepancies in predictions for genes like *HP0056* and *HP0195* highlight the complexity of enzyme-cofactor interactions. Additionally, predictions for the NDH-1 complex reveal challenges when alternative cofactors are possible. Refining models with more organism-specific data can improve prediction accuracy, especially for less-characterized species like *H. pylori*.

Discussion of iML1515 Model (Escherichia coli str. K-12 substr. MG1655)

The iML1515 model for *E. coli* K-12 substr. MG1655 showed mixed results in predicting enzyme cofactor usage. It performed well for genes present in the training set, such as *b2582* and *b3940*, which encode multifunctional enzymes like Bifunctional aspartokinase/homoserine dehydrogenase 1 and 2, known to use both NAD(H) and NADP(H). The model accurately predicted these dual cofactor interactions, likely due to the inclusion of UniProt data indicating their flexibility.

However, some predictions were inaccurate. For instance, the model predicted NADP(H) for gene b3588, but UniProt specifies NAD(H) as the correct cofactor for *E. coli*, suggesting the model failed to account for organism-specific differences. Similar issues occurred with *b3011*, where the model predicted NAD(H) instead of the correct NADP(H). The model also struggled with less common cofactors, such as menaquinone, which was underrepresented in the training data. This led to errors, such as predicting NAD(H) for *b4072*, a gene that uses quinones

Further, the genes *b2723*, *b2724*, *b2719*, and *b2722* are part of a proton-translocating formate hydrogenlyase (FHL) system, which is crucial for energy metabolism in *E. coli*. The model's prediction of NAD(H) as the cofactor for these genes is reasonable, given their connection to NAD(H) oxidoreductases (complex I and Nuo). NAD(H) binding domains are expected in this system, as NAD(H) plays a significant role in electron transport processes (169).

The model predicted that genes like *b1004*, *b3713*, *b0046*, *b3575*, *b0386*, and *b0031* can use both NAD(H) and NADP(H) as cofactors, which aligns with UniProt data showing these enzymes have flexibility in cofactor binding. Given that the model was trained on UniProt protein sequence data, it accurately reflected this dual cofactor potential, especially for genes in the training set. These predictions demonstrate the ability of enzymes to interact with either NAD(H) or NADP(H) depending on cellular conditions.

Overall, the model performed well for genes with cofactor flexibility but struggled with organism-specific variations and uncommon cofactors. This highlights the need for more balanced datasets to improve predictions for specialized enzymes and less frequent cofactors.

Discussion of iSynCJ816 Model (Synechocystis sp. PCC 6803)

The iSynCJ816 model, representing *Synechocystis* sp. PCC 6803, showed good performance overall, with a 33% mismatch rate in cofactor predictions. A BLAST analysis highlighted that many sequences matched species other than *Synechocystis*, likely due to a lack of specific annotations, which contributed to the mismatches.

Most of the mismatches involved genes associated with the NDH-1 and NDH-2 shuttles, which are

crucial for electron transport in both photosynthesis and respiration in *Synechocystis*. For NDH-1 genes like *SGL_RS10730*, *SGL_RS16995*, and *SGL_RS10735*, the model predicted the use of NAD(H), NADP(H), and Plastoquinone, aligning with UniProt data. However, recent literature (170; 171) suggests that under photosynthetic conditions, NDH-1 may primarily utilize NADP(H) instead of NAD(H), which reveals the model's limitation in capturing these nuanced, species-specific differences.

In the case of the NDH-2-related gene *SGL_RS08115,* the model predicted NADP(H), as indicated by UniProt, but the literature (172) suggests that NDH-2 proteins in *Synechocystis* likely bind NAD(H), again emphasizing the need for species-specific refinements in the model.

Additionally, genes like *SGL_RS13580*, *SGL_RS02595*, and *SGL_RS02995* were annotated in UniProt as interacting with both NAD(H) and NADP(H), but the GSM model considered only one cofactor, underscoring the need for more precise, organism-specific predictions.

Overall, while the iSynCJ816 model performed well, it struggled to distinguish between NAD(H) and NADP(H) for some genes, suggesting that more *Synechocystis*-specific data could improve future predictions.

In conclusion, while the cofactor prediction pipeline demonstrates strong potential across a range of organisms, its accuracy hinges on the quality and availability of species-specific data. The model excels for well-characterized organisms like *E. coli*, where metabolic pathways and cofactor usage are well-documented. However, for less-studied or metabolically unique organisms like *H. pylori* and *Synechocystis* sp. PCC 6803, challenges arise, particularly in distinguishing between closely related cofactors like NAD(H) and NADP(H). The model's current approach, based on simple binding predictions, struggles to account for the subtle differences in cofactor preferences and organism-specific contexts.

Incorporating taxonomic or metabolic context into the model could address these limitations, as different organisms exhibit distinct cofactor preferences and metabolic environments. For example, an enzyme that binds ubiquinone in one organism may function differently in another, depending on the availability of cofactors. Additionally, the adoption of a regression model that predicts cofactor affinity (e.g., Kcat/Km), rather than just binding, could enhance the pipeline's predictive power for complex and less-characterized organisms. These improvements would better capture the nuances of enzyme-cofactor interactions across diverse biological systems.

4.5 Web Service Functionality

The web service for cofactor prediction allows users to make predictions by submitting various input files. The service processes predictions effectively when provided with a required FASTA file, along with optional files such as the CONFIG and GSM models.

When a valid FASTA file is uploaded, the service processes the predictions and returns the expected results. Including both the FASTA and CONFIG files enables more personalized predictions, tailored to user-defined configurations. Similarly, adding a GSM model alongside the FASTA file results in the predicted cofactors and an updated version of the GSM model being returned, confirming the integration of cofactor predictions into the metabolic model.

The service requires a FASTA file to function properly, as protein sequence data is essential for generating embeddings and proceeding with predictions. Therefore, submitting only a GSM or CONFIG file without a FASTA file will not produce results, highlighting the need for sequence information in the prediction process.

The web service is accessible via the *merlin* platform (version 4.0.15, to be released soon) or directly through a hosted service, which is currently available exclusively to members of the University of Minho.

Chapter 5

Conclusions and future work

5.1 Conclusions

This dissertation presents the creation and implementation of a pipeline that predicts enzyme cofactor binding, achieving impressive results in organisms that are well-studied, like *Escherichia coli*. Using advanced protein embeddings, especially ESM2, the pipeline showed strong accuracy and reliability, making it a practical tool for researchers interested in enzyme-cofactor interactions.

An additional outcome of this project, beyond the original scope, was the development of a web service. This new feature simplifies access for researchers, allowing them to easily predict cofactor binding and incorporate these insights into their work. This tool not only makes predictions accessible but also proves valuable for metabolic engineering, where accurately assigning cofactors is essential to reconstructing GSM models.

This work also sheds light on some challenges, especially when applied to organisms that are less understood, such as *Helicobacter pylori* and *Synechocystis* sp. PCC 6803. Limited species-specific data made predictions less accurate for these organisms, particularly in differentiating between similar cofactors like NAD(H) and NADP(H). This highlights the need for a broader, more detailed dataset to improve prediction accuracy across various species.

Reflecting on the initial goals, this project successfully established a reliable tool, revealed valuable insights, and demonstrated how deep learning techniques can enhance cofactor binding predictions. This work adds to the field of bioinformatics by offering a structured and accessible way to study enzyme-cofactor interactions, with promising applications in areas like synthetic biology and metabolic engineering.

5.2 Prospect for future work

Future work may focus on improving the model's performance for complex and less-characterized organisms by incorporating additional data types, such as protein structures and enzyme activity data, to better capture the nuances of enzyme-cofactor interactions. Furthermore, exploring new DL methods like transformers could enhance the model's predictive capabilities, particularly for multi-cofactor interactions.

Introducing organism-specific context into the predictions could be another valuable step forward. By factoring in taxonomic and metabolic details, the model could better account for differences in cofactor usage between species, which would be especially useful in reconstructing GSM models. This would enhance its application in metabolic engineering, where accurate cofactor predictions are key to designing effective metabolic pathways.

To make the tool more accessible, it will be released as a PyPI package. This will allow researchers to easily integrate the pipeline into their workflows, particularly for GSM models' reconstructions. Additionally, developing an independent web service outside the limitations of the *merlin* platform would improve the tool's flexibility and scalability, making it easier to use with larger datasets and more complex models.

Bibliography

[1] Donald Hilvert. Design of Protein Catalysts. *Annual Review of Biochemistry*, 82(1):447–470, 2013.

[2] Wenfang Liu and Ping Wang. Cofactor regeneration for sustainable enzymatic biosynthesis. *Biotechnology Advances*, 25(4):369–384, July 2007.

[3] Yipeng Wang, Ka-Yiu San, and George N Bennett. Cofactor engineering for advancing chemical biotechnology. *Current Opinion in Biotechnology*, 24(6):994–999, December 2013.

[4] Joana C. Xavier, Kiran Raosaheb Patil, and Isabel Rocha. Integration of Biomass Formulations of Genome-Scale Metabolic Models with Experimental Data Reveals Universally Essential Cofactors in Prokaryotes. *Metabolic Engineering*, 39:200–208, January 2017.

[5] Stanislav Mazurenko, Zbynek Prokop, and Jiri Damborsky. Machine Learning in Enzyme Engineering. *ACS Catalysis*, 10(2):1210–1223, January 2020.

[6] Sou Sugiki, Teppei Niide, Yoshihiro Toya, and Hiroshi Shimizu. Logistic Regression-Guided Identification of Cofactor Specificity-Contributing Residues in Enzyme with Sequence Datasets Partitioned by Catalytic Properties. *ACS Synthetic Biology*, 11(12):3973–3985, December 2022.

[7] Henrik Marcus Geertz-Hansen, Nikolaj Blom, Adam M. Feist, Søren Brunak, and Thomas Nordahl Petersen. Cofactory: Sequence-based prediction of cofactor specificity of Rossmann folds. *Proteins: Structure, Function, and Bioinformatics*, 82(9):1819–1828, 2014.

[8] Jackson K. B. Cahn, Caroline A. Werlang, Armin Baumschlager, Sabine Brinkmann-Chen, Stephen L. Mayo, and Frances H. Arnold. A General Tool for Engineering the NAD/NADP Cofactor Preference of Oxidoreductases. *ACS Synthetic Biology*, 6(2):326–333, 2017.

[9] Kamil Kamiński, Jan Ludwiczak, Maciej Jasiński, Adriana Bukala, Rafal Madaj, Krzysztof Szczepaniak, and Stanisław Dunin-Horkawicz. Rossmann-toolbox: a deep learning-based protocol for the prediction and design of cofactor specificity in Rossmann fold proteins. *Briefings in Bioinformatics*, 23(1):bbab371, January 2022.

[10] Fei Li, Xiaozhe Wan, Jing Xing, Xiaoqin Tan, Xutong Li, Yulan Wang, Jihui Zhao, Xiaolong Wu, Xiaohong Liu, Zhaojun Li, Xiaomin Luo, Wencong Lu, and Mingyue Zheng. Deep Neural Network Classifier for Virtual Screening Inhibitors of (S)-Adenosyl-L-Methionine (SAM)-Dependent Methyltransferase Family. *Frontiers in Chemistry*, 7:324, May 2019.

[11] Kendall A. Smith. Louis Pasteur, the Father of Immunology? *Frontiers in Immunology*, 3:68, April 2012.

[12] J. D. WATSON and F. H. C. CRICK. Molecular Structure of Nucleic Acids: A Structure for Deoxyribose Nucleic Acid. *Nature*, 171(4356):737–738, April 1953.

[13] John Blazeck and Hal Alper. Systems metabolic engineering: Genome-scale models and beyond. *Biotechnology journal*, 5(7):647–659, July 2010.

[14] Birgit Wiltschi, Tomislav Cernava, Alexander Dennig, Meritxell Galindo Casas, Martina Geier, Steffen Gruber, Marianne Haberbauer, Petra Heidinger, Enrique Herrero Acero, Regina Kratzer, Christiane Luley-Goedl, Christina A. Müller, Julia Pitzer, Doris Ribitsch, Michael Sauer, Katharina Schmölzer, Wolfgang Schnitzhofer, Christoph W. Sensen, Jung Soh, Kerstin Steiner, Christoph K. Winkler, Margit Winkler, and Tamara Wriessnegger. Enzymes revolutionize the bioproduction of value-added compounds: From enzyme discovery to special applications. *Biotechnology Advances*, 40:107520, May 2020.

[15] Lauren B. Pickens, Yi Tang, and Yit-Heng Chooi. Metabolic Engineering for the Production of Natural Products. *Annual review of chemical and biomolecular engineering*, 2:211–236, 2011.

[16] Gregory N. Stephanopoulos, Aristos A. Aristidou, and Jens Nielsen. CHAPTER 6 - Examples of Pathway Manipulations: Metabolic Engineering in Practice. In Gregory N. Stephanopoulos, Aristos A. Aristidou, and Jens Nielsen, editors, *Metabolic Engineering*, pages 203–283. Academic Press, San Diego, 1998.

[17] Mihajlo D. Mesarović. Systems Theory and Biology—View of a Theoretician. In M. D. Mesarović, editor, *Systems Theory and Biology*, pages 59–87, Berlin, Heidelberg, 1968. Springer.

[18] Srdjan Kesić. Systems biology, emergence and antireductionism. *Saudi Journal of Biological Sciences*, 23(5):584–591, September 2016.

[19] Hans V. Westerhoff and Bernhard O. Palsson. The evolution of molecular biology into systems biology. *Nature Biotechnology*, 22(10):1249–1252, October 2004.

[20] Trey Ideker. Systems biology 101—what you need to know. *Nature Biotechnology*, 22(4):473–475, April 2004.

[21] Ana Conesa and Ali Mortazavi. The common ground of genomics and systems biology. *BMC Systems Biology*, 8(2):S1, March 2014.

[22] Hiroaki Kitano. Systems biology: a brief overview. *Science (New York, N.Y.)*, 295(5560):1662–1664, March 2002.

[23] Jeremy S. Edwards and Bernhard O. Palsson. Systems Properties of the Haemophilus influenzaeRd Metabolic Genotype. *Journal of Biological Chemistry*, 274(25):17410–17416, June 1999.

[24] Mustafa Sertbas and Kutlu O. Ulgen. Genome-Scale Metabolic Modeling for Unraveling Molecular Mechanisms of High Threat Pathogens. *Frontiers in Cell and Developmental Biology*, 8:566702, November 2020.

[25] J. S. Edwards and B. O. Palsson. The Escherichia coli MG1655 in silico metabolic genotype: Its definition, characteristics, and capabilities. *Proceedings of the National Academy of Sciences of the United States of America*, 97(10):5528–5533, May 2000.

[26] Jochen Förster, Iman Famili, Patrick Fu, Bernhard Ø Palsson, and Jens Nielsen. Genome-scale reconstruction of the Saccharomyces cerevisiae metabolic network. *Genome Research*, 13(2):244–253, February 2003.

[27] Natalie C. Duarte, Scott A. Becker, Neema Jamshidi, Ines Thiele, Monica L. Mo, Thuy D. Vo, Rohith Srivas, and Bernhard Ø. Palsson. Global reconstruction of the human metabolic network based on genomic and bibliomic data. *Proceedings of the National Academy of Sciences of the United States of America*, 104(6):1777–1782, February 2007.

[28] Thiago Batista Moreira, Rahul Shaw, Xinyu Luo, Oishik Ganguly, Hyung-Seok Kim, Lucas Gabriel Ferreira Coelho, Chun Yue Maurice Cheung, and Thomas Christopher Rhys Williams. A Genome-Scale Metabolic Model of Soybean (Glycine max) Highlights Metabolic Fluxes in Seedlings. *Plant Physiology*, 180(4):1912–1929, August 2019.

[29] Ines Thiele and Bernhard Ø Palsson. A protocol for generating a high-quality genome-scale metabolic reconstruction. *Nature Protocols*, 5(1):93–121, January 2010.

[30] Jeffrey D. Orth, Ines Thiele, and Bernhard Ø Palsson. What is flux balance analysis? *Nature Biotechnology*, 28(3):245–248, March 2010.

[31] Matthew L. Jenior, Thomas J. Moutinho Jr, Bonnie V. Dougherty, and Jason A. Papin. Transcriptome-guided parsimonious flux analysis improves predictions with metabolic networks in complex environments. *PLOS Computational Biology*, 16(4):e1007099, April 2020.

[32] Samuel M D Seaver, Filipe Liu, Qizhi Zhang, James Jeffryes, José P Faria, Janaka N Edirisinghe, Michael Mundy, Nicholas Chia, Elad Noor, Moritz E Beber, Aaron A Best, Matthew DeJongh, Jeffrey A Kimbrel, Patrik D'haeseleer, Sean R McCorkle, Jay R Bolton, Erik Pearson, Shane Canon, Elisha M Wood-Charlson, Robert W Cottingham, Adam P Arkin, and Christopher S Henry. The ModelSEED Biochemistry Database for the integration of metabolic annotations and the reconstruction, comparison and analysis of metabolic models for plants, fungi and microbes. *Nucleic Acids Research*, 49(D1):D575–D588, January 2021.

[33] Adam P. Arkin, Robert W. Cottingham, Christopher S. Henry, Nomi L. Harris, Rick L. Stevens, Sergei Maslov, Paramvir Dehal, Doreen Ware, Fernando Perez, Shane Canon, Michael W. Sneddon, Matthew L. Henderson, William J. Riehl, Dan Murphy-Olson, Stephen Y. Chan, Roy T. Kamimura, Sunita Kumari, Meghan M. Drake, Thomas S. Brettin, Elizabeth M. Glass, Dylan Chivian, Dan Gunter, David J. Weston, Benjamin H. Allen, Jason Baumohl, Aaron A. Best, Ben Bowen, Steven E. Brenner, Christopher C. Bun, John-Marc Chandonia, Jer-Ming Chia, Ric Colasanti, Neal Conrad, James J. Davis, Brian H. Davison, Matthew DeJongh, Scott Devoid, Emily Dietrich, Inna Dubchak, Janaka N. Edirisinghe, Gang Fang, José P. Faria, Paul M. Frybarger, Wolfgang Gerlach, Mark Gerstein, Annette Greiner, James Gurtowski, Holly L. Haun, Fei He, Rashmi Jain, Marcin P. Joachimiak, Kevin P. Keegan, Shinnosuke Kondo, Vivek Kumar, Miriam L. Land, Folker Meyer, Marissa Mills, Pavel S. Novichkov, Taeyun Oh, Gary J. Olsen, Robert Olson, Bruce Parrello, Shiran Pasternak, Erik Pearson, Sarah S. Poon, Gavin A. Price, Srividya Ramakrishnan, Priya Ranjan, Pamela C. Ronald, Michael C. Schatz, Samuel M. D. Seaver, Maulik Shukla, Roman A. Sutormin, Mustafa H. Syed, James Thomason, Nathan L. Tintle, Daifeng Wang, Fangfang Xia, Hyunseung Yoo, Shinjae Yoo, and Dantong Yu. KBase: The United States Department of Energy Systems Biology Knowledgebase. *Nature Biotechnology*, 36(7):566–569, August 2018.

[34] Daniel Machado, Sergej Andrejev, Melanie Tramontano, and Kiran Raosaheb Patil. Fast automated reconstruction of genome-scale metabolic models for microbial species and communities. *Nucleic Acids Research*, 46(15):7542–7553, September 2018.

[35] João Capela, Davide Lagoa, Ruben Rodrigues, Emanuel Cunha, Fernando Cruz, Ana Barbosa, José Bastos, Diogo Lima, Eugénio C Ferreira, Miguel Rocha, and Oscar Dias. *merlin* , an improved framework for the reconstruction of high-quality genome-scale metabolic models. *Nucleic Acids Research*, 50(11):6052–6066, June 2022.

[36] Hao Wang, Simonas Marcišauskas, Benjamín J. Sánchez, Iván Domenzain, Daniel Hermansson, Rasmus Agren, Jens Nielsen, and Eduard J. Kerkhoven. RAVEN 2.0: A versatile toolbox for metabolic network reconstruction and a case study on Streptomyces coelicolor. *PLOS Computational Biology*, 14(10):e1006541, October 2018.

[37] Méziane Aite, Marie Chevallier, Clémence Frioux, Camille Trottier, Jeanne Got, María Paz Cortés, Sebastián N. Mendoza, Grégory Carrier, Olivier Dameron, Nicolas Guillaudeux, Mauricio Latorre, Nicolás Loira, Gabriel V. Markov, Alejandro Maass, and Anne Siegel. Traceability, reproducibility and wiki-exploration for "à-la-carte" reconstructions of genome-scale metabolic models. *PLoS computational biology*, 14(5):e1006146, May 2018.

[38] Emil Karlsen, Christian Schulz, and Eivind Almaas. Automated generation of genome-scale metabolic draft reconstructions based on KEGG. *BMC Bioinformatics*, 19(1):467, December 2018.

[39] Mark Hanemaaijer, Brett G. Olivier, Wilfred F. M. Röling, Frank J. Bruggeman, and Bas Teusink. Model-based quantification of metabolic interactions from dynamic microbial-community data. *PLOS ONE*, 12(3):e0173183, March 2017.

[40] Joost Boele, Brett G. Olivier, and Bas Teusink. FAME, the Flux Analysis and Modeling Environment. *BMC Systems Biology*, 6(1):8, January 2012.

[41] Stephan Pabinger, Robert Rader, Rasmus Agren, Jens Nielsen, and Zlatko Trajanoski. MEMOSys: Bioinformatics platform for genome-scale metabolic models. *BMC Systems Biology*, 5(1):20, January 2011.

[42] Yu-Chieh Liao, Ming-Hsin Tsai, Feng-Chi Chen, and Chao A. Hsiung. GEMSiRV: a software platform for GEnome-scale metabolic model simulation, reconstruction and visualization. *Bioinformatics*, 28(13):1752–1758, July 2012.

[43] Peter D. Karp, Mario Latendresse, Suzanne M. Paley, Markus Krummenacker, Quang D. Ong, Richard Billington, Anamika Kothari, Daniel Weaver, Thomas Lee, Pallavi Subhraveti, Aaron Spaulding, Carol Fulcher, Ingrid M. Keseler, and Ron Caspi. Pathway Tools version 19.0 update: software

for pathway/genome informatics and systems biology. *Briefings in Bioinformatics*, 17(5):877–890, September 2016.

[44] Esa Pitkänen, Paula Jouhten, Jian Hou, Muhammad Fahad Syed, Peter Blomberg, Jana Kludas, Merja Oja, Liisa Holm, Merja Penttilä, Juho Rousu, and Mikko Arvas. Comparative Genome-Scale Reconstruction of Gapless Metabolic Networks for Present and Ancestral Species. *PLOS Computational Biology*, 10(2):e1003465, February 2014.

[45] Stefan Gretar Thorleifsson and Ines Thiele. rBioNet: A COBRA toolbox extension for reconstructing high-quality biochemical networks. *Bioinformatics*, 27(14):2009–2010, July 2011.

[46] Neil Swainston, Kieran Smallbone, Pedro Mendes, Douglas B. Kell, and Norman W. Paton. The SuBliMinaL Toolbox: automating steps in the reconstruction of metabolic networks. *Journal of Integrative Bioinformatics*, 8(2):187–203, June 2011.

[47] Ali Ebrahim, Joshua A. Lerman, Bernhard O. Palsson, and Daniel R. Hyduke. COBRApy: COnstraints-Based Reconstruction and Analysis for Python. *BMC Systems Biology*, 7(1):74, August 2013.

[48] Isabel Rocha, Paulo Maia, Pedro Evangelista, Paulo Vilaça, Simão Soares, José P. Pinto, Jens Nielsen, Kiran R. Patil, Eugénio C. Ferreira, and Miguel Rocha. OptFlux: an open-source software platform for in silico metabolic engineering. *BMC Systems Biology*, 4(1):45, April 2010.

[49] Andreas Karoly Gombert and Jens Nielsen. Mathematical modelling of metabolism. *Current Opinion in Biotechnology*, 11(2):180–186, April 2000.

[50] Aarash Bordbar, Jonathan M. Monk, Zachary A. King, and Bernhard O. Palsson. Constraint-based models predict metabolic and associated cellular functions. *Nature Reviews Genetics*, 15(2):107–120, February 2014.

[51] Richard Wolfenden and Mark J. Snider. The Depth of Chemical Time and the Power of Enzymes as Catalysts. *Accounts of Chemical Research*, 34(12):938–945, December 2001.

[52] Gemma L. Holliday, Julia D. Fischer, John B. O. Mitchell, and Janet M. Thornton. Characterizing the complexity of enzymes on the basis of their mechanisms and structures with a bio-computational analysis. *The FEBS Journal*, 278(20):3835–3845, 2011.

[53] Andrew McDonald and Keith Tipton. Enzyme Nomenclature and Classification: the State of the Art. *The FEBS Journal*, 290, November 2021.

[54] D. T. Lee, J. T. Lee, A. J. Ashworth, M. T. Kidd, A. Mauromoustakos, and S. J. Rochell. Evaluation of a threonine fermentation product as a digestible threonine source in broilers. *Journal of Applied Poultry Research*, 31(3):100252, 2022.

[55] Daniel E. Koshland Jr. The Key–Lock Theory and the Induced Fit Theory. *Angewandte Chemie International Edition in English*, 33(23-24):2375–2378, 1995.

[56] DE Koshland Jr. Application of a theory of enzyme specificity to protein synthesis. *Proceedings of the National Academy of Sciences*, 44(2):98–104, 1958.

[57] Joan B Broderick. Coenzymes and cofactors. In *Encyclopedia of Life Sciences*. John Wiley & Sons, Ltd, 2001.

[58] O. Carugo and P. Argos. NADP-dependent enzymes. I: Conserved stereochemistry of cofactor binding. *Proteins*, 28(1):10–28, May 1997.

[59] Lara Sellés Vidal, Ciarán L. Kelly, Paweł M. Mordaka, and John T. Heap. Review of NAD(P)H-dependent oxidoreductases: Properties, engineering and application. *Biochimica et Biophysica Acta (BBA) - Proteins and Proteomics*, 1866(2):327–347, 2018.

[60] Ivan Gout. Coenzyme A: a protective thiol in bacterial antioxidant defence. *Biochemical Society Transactions*, 47(1):469–476, February 2019.

[61] Steven O. Mansoorabadi, Christopher J. Thibodeaux, and Hung-wen Liu. The Diverse Roles of Flavin Coenzymes - Nature's Most Versatile Thespians. *The Journal of organic chemistry*, 72(17):6329–6342, August 2007.

[62] Marcelina Parra, Seth Stahl, and Hanjo Hellmann. Vitamin B6 and Its Role in Cell Metabolism and Physiology. *Cells*, 7(7):84, July 2018.

[63] Huimin Liu, Shaoming Sang, Yuan Lu, Zhongfeng Wang, Xiang Yu, and Chunjiu Zhong. Thiamine metabolism is critical for regulating correlated growth of dendrite arbors and neuronal somata. *Scientific Reports*, 7(1):5342, July 2017.

[64] Ida Schomburg, Antje Chang, and Dietmar Schomburg. BRENDA, enzyme data and metabolic information. *Nucleic Acids Research*, 30(1):47–49, January 2002.

[65] Ron Caspi, Richard Billington, Ingrid M. Keseler, Anamika Kothari, Markus Krummenacker, Peter E. Midford, Wai Kit Ong, Suzanne Paley, Pallavi Subhraveti, and Peter D. Karp. The MetaCyc database of metabolic pathways and enzymes - a 2019 update. *Nucleic Acids Research*, 48(D1):D445–D453, January 2020.

[66] Ulrike Wittig, Martin Golebiewski, Renate Kania, Olga Krebs, Saqib Mir, Andreas Weidemann, Stefanie Anstein, Jasmin Saric, and Isabel Rojas. SABIO-RK: Integration and Curation of Reaction Kinetics Data. In Ulf Leser, Felix Naumann, and Barbara Eckman, editors, *Data Integration in the Life Sciences*, pages 94–103, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[67] Abhik Mukhopadhyay, Neera Borkakoti, Lukáš Pravda, Jonathan D Tyzack, Janet M Thornton, and Sameer Velankar. Finding enzyme cofactors in protein data bank. *Bioinformatics*, 35(18):3510–3511, 2019.

[68] S. T. Rao and Michael G. Rossmann. Comparison of super-secondary structures in proteins. *Journal of Molecular Biology*, 76(2):241–256, May 1973.

[69] Yun Hao Hua, Chih Yuan Wu, Karen Sargsyan, and Carmay Lim. Sequence-motif Detection of NAD(P)-binding Proteins: Discovery of a Unique Antibacterial Drug Target. *Scientific Reports*, 4(1):6471, September 2014.

[70] George A Khoury, Hossein Fazelinia, Jonathan W Chin, Robert J Pantazes, Patrick C Cirino, and Costas D Maranas. Computational design of Candida boidinii xylose reductase for altered cofactor specificity. *Protein Science : A Publication of the Protein Society*, 18(10):2125–2138, October 2009.

[71] Dongbing Cui, Lujia Zhang, Shuiqin Jiang, Zhiqiang Yao, Bei Gao, Jinping Lin, Y. Adam Yuan, and Dongzhi Wei. A computational strategy for altering an enzyme in its cofactor preference to NAD(H) and/or NADP(H). *The FEBS journal*, 282(12):2339–2351, June 2015.

[72] Christopher C. Marohnic, Maria C. Bewley, and Michael J. Barber. Engineering and Characterization of a NADPH-Utilizing Cytochrome b5 Reductase. *Biochemistry*, 42(38):11170–11182, September 2003.

[73] Sunghark Kwon. Structural Insight into the Working Mechanism of the FAD Synthetase from the Human Pathogen Streptococcus pneumoniae: A Molecular Docking Simulation Study. *International Journal of Molecular Sciences*, 24(4):3121, February 2023.

[74] Flavia A. Mercurio, Concetta Di Natale, Luciano Pirone, Marian Vincenzi, Daniela Marasco, Stefania De Luca, Emilia M. Pedone, and Marilisa Leone. Exploring the Ability of Cyclic Peptides to Target SAM Domains: A Computational and Experimental Study. *ChemBioChem*, 21(5):702–711, 2020.

[75] Mehryar Mohri – Foundations of Machine Learning - Book.

[76] Absalom E Ezugwu, Abiodun M Ikotun, Olaide O Oyelade, Laith Abualigah, Jeffery O Agushaka, Christopher I Eke, and Andronicus A Akinyelu. A comprehensive survey of clustering algorithms: State-of-the-art machine learning applications, taxonomy, challenges, and future research prospects. *Engineering Applications of Artificial Intelligence*, 110:104743, 2022.

[77] Basna Mohammed Salih Hasan and Adnan Mohsin Abdulazeez. A review of principal component analysis algorithm for dimensionality reduction. *Journal of Soft Computing and Data Mining*, 2(1):20–30, 2021.

[78] Geoffrey E Hinton and Sam Roweis. Stochastic neighbor embedding. *Advances in neural information processing systems*, 15, 2002.

[79] George C Linderman, Manas Rachh, Jeremy G Hoskins, Stefan Steinerberger, and Yuval Kluger. Fast interpolation-based t-sne for improved visualization of single-cell rna-seq data. *Nature methods*, 16(3):243–245, 2019.

[80] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction, 2020.

[81] Tammy Jiang, Jaimie L Gradus, and Anthony J Rosellini. Supervised machine learning: a brief primer. *Behavior therapy*, 51(5):675–687, 2020.

[82] Batta Mahesh. *Machine Learning Algorithms -A Review*. January 2019.

[83] Thomas M. H. Hope. Chapter 4 - Linear regression. In Andrea Mechelli and Sandra Vieira, editors, *Machine Learning*, pages 67–81. Academic Press, 2020.

[84] Susmita Ray. A quick review of machine learning algorithms. In *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, pages 35–39, 2019.

[85] Lior Rokach and Oded Maimon. Decision Trees. In Oded Maimon and Lior Rokach, editors, *Data Mining and Knowledge Discovery Handbook*, pages 165–192. Springer US, Boston, MA, 2005.

[86] Jehad Ali, Rehanullah Khan, Nasir Ahmad, and Imran Maqsood. Random Forests and Decision Trees. *International Journal of Computer Science Issues(IJCSI)*, 9(5), 2012.

[87] Peter Zhang, Eddy Patuwo, and Michael Hu. Forecasting With Artificial Neural Networks: The State of the Art. *International Journal of Forecasting*, 14:35–62, March 1998.

[88] Kenji Suzuki. *Artificial Neural Networks - Methodological Advances and Biomedical Applications*. IntechOpen, April 2011.

[89] J.J. Hopfield. Artificial neural networks. *IEEE Circuits and Devices Magazine*, 4(5):3–10, 1988.

[90] Iqbal H. Sarker. Machine Learning: Algorithms, Real-World Applications and Research Directions. *Sn Computer Science*, 2(3):160, 2021.

[91] Sebastian Raschka. *Python Machine Learning*. Packt Publishing Ltd, September 2015.

[92] Dahouda Mwamba and Inwhee Joe. A deep-learned embedding technique for categorical features encoding. *IEEE Access*, PP:1–1, 08 2021.

[93] Hwa Jong Kim, Seong Eun Hong, and Kyung Jin Cha. seq2vec: Analyzing sequential data using multi-rank embedding vectors. *Electronic Commerce Research and Applications*, 43:101003, 2020.

[94] Zeming Lin, Halil Akin, Roshan Rao, Brian Hie, Zhongkai Zhu, Wenting Lu, Nikita Smetanin, Allan dos Santos Costa, Maryam Fazel-Zarandi, Tom Sercu, Sal Candido, et al. Language models of protein sequences at the scale of evolution enable accurate structure prediction. *bioRxiv*, 2022.

[95] Lei Wang, Hui Zhang, Wei Xu, Zhidong Xue, and Yan Wang. Deciphering the protein landscape with protflash, a lightweight language model. *Cell Reports Physical Science*, 4(10), 2023.

[96] Kuo-Chen Chou. Prediction of protein cellular attributes using pseudo-amino acid composition. *Proteins: Structure, Function, and Bioinformatics*, 43(3):246–255, 2001.

[97] Srikanth Patala. Understanding grain boundaries – the role of crystallography, structural descriptors and machine learning. *Computational Materials Science*, 162:281–294, 2019.

[98] Saúl Solorio-Fernández, J Ariel Carrasco-Ochoa, and José Fco Martínez-Trinidad. A review of unsupervised feature selection methods. *Artificial Intelligence Review*, 53(2):907–948, 2020.

[99] Andreas C Müller and Sarah Guido. *Introduction to machine learning with Python: a guide for data scientists*. " O'Reilly Media, Inc.", 2016.

[100] Isabelle Guyon, Jason Weston, Stephen Barnhill, and Vladimir Vapnik. Gene selection for cancer classification using support vector machines. *Machine learning*, 46:389–422, 2002.

[101] MP Rajan. An efficient ridge regression algorithm with parameter estimation for data analysis in machine learning. *SN Computer Science*, 3(2):171, 2022.

[102] V. Roshan Joseph and Akhil Vakayil. SPlit: An Optimal Method for Data Splitting. *Technometrics*, 64(2):166–176, 2022.

[103] Yangguang Liu, Yangming Zhou, Shiting Wen, and Chaogang Tang. A Strategy on Selecting Performance Metrics for Classifier Evaluation. *International Journal of Mobile Computing and Multimedia Communications*, 6:20–35, October 2014.

[104] Alexei Botchkarev. A new typology design of performance metrics to measure errors in machine learning regression algorithms. *Interdisciplinary Journal of Information, Knowledge, and Management*, 14:045–076, 2019.

[105] Suyash Kumar, Prabhjot Kaur, and Anjana Gosain. A comprehensive survey on ensemble methods. In *2022 IEEE 7th International conference for Convergence in Technology (I2CT)*, pages 1–7, 2022.

[106] Yoav Freund and Robert E. Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In Paul Vitányi, editor, *Computational Learning Theory*, pages 23–37, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

[107] Jerome H. Friedman. Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38(4):367–378, 2002. Nonlinear Methods and Data Mining.

[108] Navid Kardani, Annan Zhou, Majidreza Nazem, and Shui-Long Shen. Improved prediction of slope stability using a hybrid stacking ensemble method based on finite element analysis and field data. *Journal of Rock Mechanics and Geotechnical Engineering*, 13(1):188–201, 2021.

[109] Lily-belle Sweet, Christoph Müller, Mohit Anand, and Jakob Zscheischler. Cross-validation strategy impacts the performance and interpretation of machine learning models. *Artificial Intelligence for the Earth Systems*, 2(4):e230026, 2023.

[110] Ivan Idris. *NumPy: Beginner's Guide*. Packt Publishing Ltd, 2015.

[111] Wes McKinney et al. pandas: a foundational python library for data analysis and statistics. *Python for high performance and scientific computing*, 14(9):1–9, 2011.

[112] Eric Jones, Travis Oliphant, Pearu Peterson, et al. Scipy: Open source scientific tools for python. 2001.

[113] Ana Marta Sequeira, Diana Lousa, and Miguel Rocha. ProPythia: A Python Automated Platform for the Classification of Proteins Using Machine Learning. In Gabriella Panuccio, Miguel Rocha, Florentino Fdez-Riverola, Mohd Saberi Mohamad, and Roberto Casado-Vara, editors, *Practical Applications of Computational Biology & Bioinformatics, 14th International Conference (PACBB 2020)*, pages 32–41, Cham, 2021. Springer International Publishing.

[114] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library, December 2019.

[115] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: A system for large-scale machine learning, May 2016.

[116] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631, 2019.

[117] Ekaba Bisong and Ekaba Bisong. Matplotlib and seaborn. *Building machine learning and deep learning models on google cloud platform: A comprehensive guide for beginners*, pages 151–165, 2019.

[118] Michael L. Waskom. seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021, 2021.

[119] Min-Ling Zhang, Yu-Kun Li, Xu-Ying Liu, and Xin Geng. Binary relevance for multi-label learning: an overview. *Frontiers of Computer Science*, 12:191–202, 2018.

[120] Matthew R Boutell, Jiebo Luo, Xipeng Shen, and Christopher M Brown. Learning multi-label scene classification. *Pattern recognition*, 37(9):1757–1771, 2004.

[121] Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Eibe Frank. Classifier chains for multi-label classification. *Machine learning*, 85:333–359, 2011.

[122] Min-Ling Zhang and Zhi-Hua Zhou. A k-nearest neighbor based algorithm for multi-label classification. In *2005 IEEE international conference on granular computing*, volume 2, pages 718–721. IEEE, 2005.

[123] Jiarong Wang, Jun Feng, Xia Sun, Su-Shing Chen, and Bo Chen. Simplified constraints rank-svm for multi-label classification. In *Pattern Recognition: 6th Chinese Conference, CCPR 2014, Changsha, China, November 17-19, 2014. Proceedings, Part I 6*, pages 229–236. Springer, 2014.

[124] Hassan H Malik and Vikas S Bhardwaj. Automatic training data cleaning for text classification. In *2011 IEEE 11th international conference on data mining workshops*, pages 442–449. IEEE, 2011.

[125] Raed Alazaidah and Farzana Kabir Ahmad. Trending challenges in multi label classification. *International Journal of Advanced Computer Science and Applications*, 7(10):127–131, 2016.

[126] Jesse Read, Antti Puurula, and Albert Bifet. Multi-label classification with meta-labels. In *2014 IEEE international conference on data mining*, pages 941–946. IEEE, 2014.

[127] Li Li and Houfeng Wang. Towards label imbalance in multi-label classification with many labels. *arXiv preprint arXiv:1604.01304*, 2016.

[128] Mohammad Reza Rezaei-Dastjerdehei, Amirmohammad Mijani, and Emad Fatemizadeh. Addressing imbalance in multi-label classification using weighted cross entropy loss function. In *2020 27th National and 5th International Iranian Conference on Biomedical Engineering (ICBME)*, pages 333–338. IEEE, 2020.

[129] Tal Ridnik, Emanuel Ben-Baruch, Nadav Zamir, Asaf Noy, Itamar Friedman, Matan Protter, and Lihi Zelnik-Manor. Asymmetric loss for multi-label classification. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 82–91, 2021.

[130] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, January 2015.

[131] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015.

[132] D. H. Hubel and T. N. Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of Physiology*, 195(1):215–243, March 1968.

[133] Rikiya Yamashita, Mizuho Nishio, Richard Kinh Gian Do, and Kaori Togashi. Convolutional neural networks: an overview and application in radiology. *Insights into Imaging*, 9(4):611–629, August 2018.

[134] Laith Alzubaidi, Jinglan Zhang, Amjad J. Humaidi, Ayad Al-Dujaili, Ye Duan, Omran Al-Shamma, J. Santamaría, Mohammed A. Fadhel, Muthana Al-Amidie, and Laith Farhan. Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. *Journal of Big Data*, 8(1):53, March 2021.

[135] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, and Tsuhan Chen. Recent advances in convolutional neural networks. *Pattern Recognition*, 77:354–377, 2018.

[136] Xue Ying. An Overview of Overfitting and its Solutions. *Journal of Physics: Conference Series*, 1168:022022, February 2019.

[137] Saedeh Abbaspour, Faranak Fotouhi, Ali Sedaghatbaf, Hossein Fotouhi, Maryam Vahabi, and Maria Linden. A Comparative Analysis of Hybrid Deep Learning Models for Human Activity Recognition. *Sensors*, 20(19), 2020.

[138] Wei Fang, Yupeng Chen, and Qiongying Xue. Survey on Research of RNN-Based Spatio-Temporal Sequence Prediction Algorithms. *Journal on Big Data*, 3:97–110, January 2021.

[139] Halit Apaydin, Hajar Feizi, Mohammad Taghi Sattari, Muslume Sevba Colak, Shahaboddin Shamshirband, and Kwok-Wing Chau. Comparative Analysis of Recurrent Neural Network Architectures for Reservoir Inflow Forecasting. *Water*, 12(5), 2020.

[140] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-term Memory. *Neural computation*, 9:1735–80, December 1997.

[141] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling, December 2014.

[142] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning Internal Representations by Error Propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*, pages 318–362. MIT Press, Cambridge, MA, USA, 1986.

[143] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders, April 2021.

[144] Pascal Vincent, H. Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion. *J. Mach. Learn. Res.*, 11:3371–3408, 2010.

[145] Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes, December 2022.

[146] Diederik P. Kingma and Max Welling. An Introduction to Variational Autoencoders. *Foundations and Trends® in Machine Learning*, 12(4):307–392, 2019.

[147] A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

[148] Jacob Devlin. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[149] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding with unsupervised learning. 2018.

[150] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2, 2014.

[151] Paolo Di Tommaso, Maria Chatzou, Evan W Floden, Pablo Prieto Barja, Emilio Palumbo, and Cedric Notredame. Nextflow enables reproducible computational workflows. *Nature biotechnology*, 35(4):316–319, 2017.

[152] Miguel Grinberg. *Flask web development: developing web applications with python*. " O'Reilly Media, Inc.", 2018.

[153] Ulrike Wittig, Maja Rey, Andreas Weidemann, Renate Kania, and Wolfgang Müller. Sabio-rk: an updated resource for manually curated biochemical reaction kinetics. *Nucleic acids research*, 46(D1):D656–D660, 2018.

[154] Semidán Robaina Estévez. Robaina/brendapyrser: zenodo, August 2022.

[155] Jannell V. Bazurto, Kristen R. Farley, and Diana M. Downs. An unexpected route to an essential cofactor: Escherichia coli relies on threonine for thiamine biosynthesis. *Mbio*, 7(1), 2016.

[156] Yang Sun, Ting Zhang, Bing qian Lu, Xiangfei LiX. Li, and Ling min Jiang. Application of cofactors in the regulation of microbial metabolism: A state of the art review. *Frontiers in Microbiology*, 14, 2023.

[157] Margaret F. Romine, Dmitry A. Rodionov, Yukari Maezato, Andrei L. Osterman, and William C. Nelson. Underlying mechanisms for syntrophic metabolism of essential enzyme cofactors in microbial communities. *The ISME Journal*, 11(6):1434–1446, 2017.

[158] Nan Xu, Chao Ye, Xiulai Chen, Jia Liu, and Liming Liu. Genome-scale metabolic modelling common cofactors metabolism in microorganisms. *Journal of Biotechnology*, 251:1–13, 2017.

[159] Alexander Rives, Joshua Meier, Tom Sercu, Siddharth Goyal, Zeming Lin, Jason Liu, Demi Guo, Myle Ott, C Lawrence Zitnick, Jerry Ma, et al. Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *Proceedings of the National Academy of Sciences*, 118(15):e2016239118, 2021.

[160] Konstantinos Sechidis, Grigorios Tsoumakas, and Ioannis Vlahavas. On the stratification of multi-label data. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2011, Athens, Greece, September 5-9, 2011, Proceedings, Part III 22*, pages 145–158. Springer, 2011.

[161] Lavkush Sharma, A. Deepak, Ashish Ranjan, and Gopalakrishnan Krishnasamy. A cnn-cbam-bigru model for protein function prediction. *Statistical Applications in Genetics and Molecular Biology*, 23(1), 2024.

[162] Abhishek Pandey and Sanjiban Shekhar Roy. Protein sequence classification using convolutional neural network and natural language processing. In *Handbook of Machine Learning Applications for Genomics*, pages 133–144. Springer, 2022.

[163] Thanh Hai Dang and Tien Anh Vu. xcapt5: protein–protein interaction prediction using deep and wide multi-kernel pooling convolutional neural networks with protein language model. *BMC bioinformatics*, 25(1):106, 2024.

[164] Leonardo V Castorina, Suleyman Mert Ünal, Kartic Subr, and Christopher W Wood. TIMED-Design: flexible and accessible protein sequence design with convolutional neural networks. *Protein Engineering, Design and Selection*, 37:gzae002, 01 2024.

[165] Yapei Tong, Henriette J. Rozeboom, Marnix R. Loonstra, Hein J. Wijma, and Marco W. Fraaije. Characterization of two bacterial multi-flavinylated proteins harboring multiple covalent flavin cofactors. *BBA Advances*, 4:100097, 2023.

[166] Takashi Iyanagi. Molecular mechanism of metabolic nad(p)h-dependent electron-transfer systems: The role of redox cofactors. *Biochimica et Biophysica Acta (BBA) - Bioenergetics*, 1860(3):233–258, 2019.

[167] Stephen J Hearnshaw. *Multicofactor proteins: Structure, prediction, function*. PhD thesis, University of East Anglia, 2011.

[168] Björn Stegemann and Gerhard Klebe. Cofactor-binding sites in proteins of deviating sequence: Comparative analysis and clustering in torsion angle, cavity, and fold space. *Proteins: Structure, Function, and Bioinformatics*, 80(2):626–648, 2012.

[169] Simon C Andrews, Ben C Berks, Joseph McClay, Andrew Ambler, Michael A Quail, Paul Golby, and John R Guest. A 12-cistron escherichia coli operon (hyf) encoding a putative proton-translocating formate hydrogenlyase system. *Microbiology*, 143(11):3633–3647, 1997.

[170] Weimin Ma, Young Deng, Teruo Ogawa, and Hualing Mi. Active ndh-1 complexes from the cyanobacterium synechocystis sp. strain pcc 6803. *Plant and cell physiology*, 47(10):1432–1436, 2006.

[171] Neil T. Miller, Ghada Ajlani, and Robert L. Burnap. Cyclic electron flow-coupled proton pumping in synechocystis sp. pcc6803 is dependent upon nadph oxidation by the soluble isoform of ferredoxin:nadp-oxidoreductase. *Microorganisms*, 10(5):855–855, 2022.

[172] Crispin A Howitt, Pacer K Udall, and Wim FJ Vermaas. Type 2 nadh dehydrogenases in the cyanobacterium synechocystis sp. strain pcc 6803 are involved in regulation rather than respiration. *Journal of bacteriology*, 181(13):3994–4003, 1999.

Part I
Appendices

Appendix A
Details of results

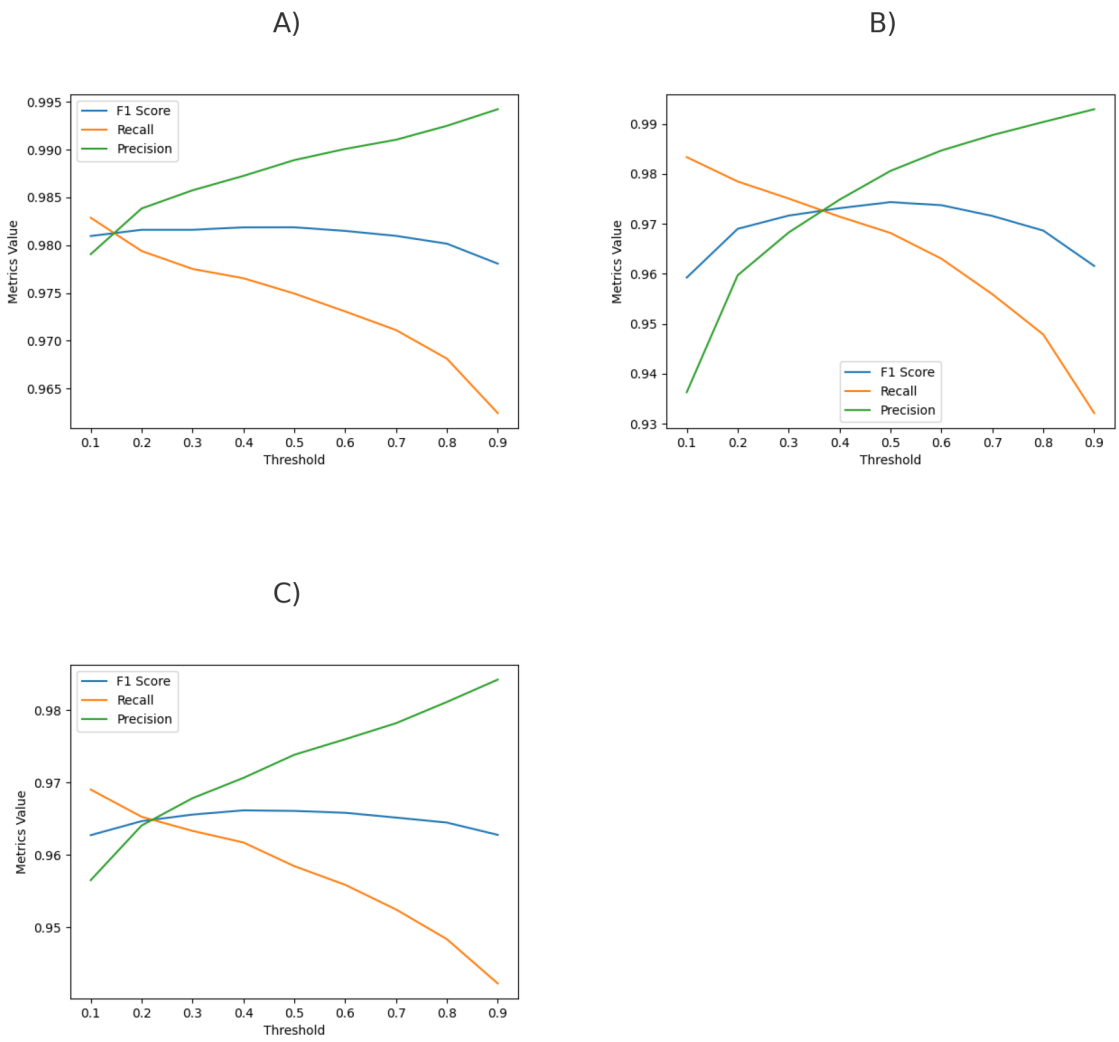


Figure 15: **F1 Score, Precision, and Recall at Varying Thresholds.** (A) represents the results for ESM2 embeddings, (B) represents the results for SeqVec embeddings, and (C) represents the results for FlashProt embeddings. The plots demonstrate the trade-off between precision, recall, and F1 Score across different thresholds, with 0.5 being selected as the optimal threshold for all models.

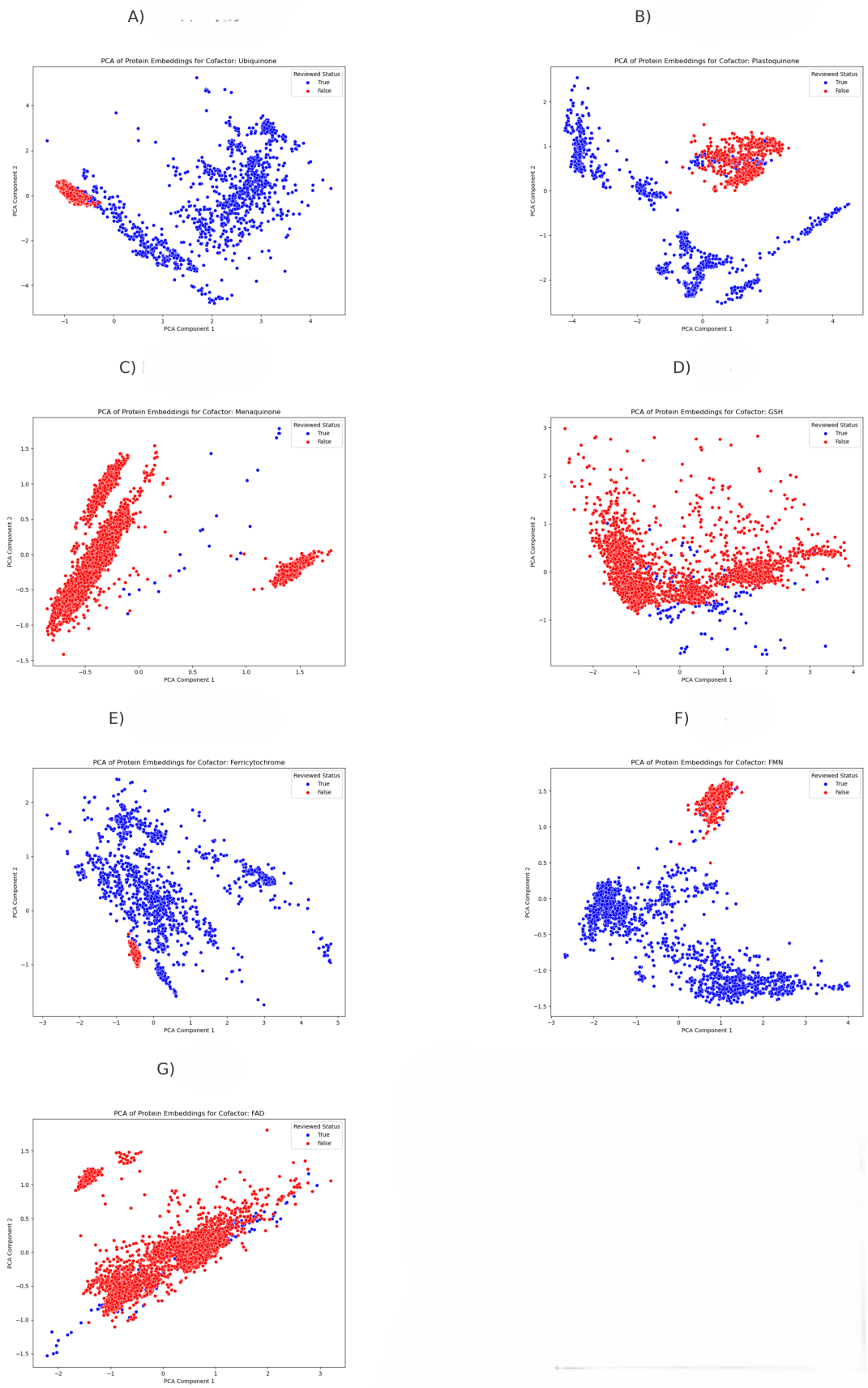


Figure 16: **PCA of Protein Embeddings for Different Cofactors.** The plots display PCA projections of reviewed (blue) and unreviewed (red) protein embeddings for different cofactors: A) Ubiquinone, B) Plastoquinone, C) Menaquinone, D) GSH, E) Ferricytochrome, F) FMN, G) FAD.

Table 17: Summary of case studies showing the total number of reactions, the number of reactions involving cofactors, reactions with sequence data, and the number of missing sequences.

GSM model	**Total Number of Reactions**	**Number of Reactions with Cofactors**	**Reactions with Sequence**	**Missing Sequences in Reactions with Cofactors**
iAF692	690	113	99	14
iAF987	1285	223	177	46
iAM_Pb448	1067	120	52	68
iAM_Pc455	1074	121	55	66
iAM_Pf480	1083	121	49	72
iAM_Pv461	1078	121	56	65
iAPECO1_1312	2735	339	207	132
iBWG_1329	2741	341	222	119
ic_1306	2726	332	210	122
iCN718	1015	225	208	17
iCN900	1229	240	127	113
iE2348C_1286	2703	328	210	118
iEC042_1314	2714	335	216	119
iEC1356_Bl21DE3	2740	351	224	127
iEC1372_W3110	2758	353	228	125

Continued on next page

Table 17: (Continued)

GSM model	**Total Number of Reactions**	**Number of Reactions with Cofactors**	**Reactions with Sequence**	**Missing Sequences in Reactions with Cofactors**
iEC55989_1330	2756	345	224	121
iECABU_c1320	2731	332	210	122
iECB_1328	2748	347	225	122
iECBD_1354	2748	347	225	122
iECDH1ME8569_1439	2755	344	219	125
iECDH10B_1368	2742	343	217	126
iEcE24377_1341	2763	346	224	122
iECED1_1282	2706	332	216	116
iECH74115_1262	2694	331	210	121
iEcHS_1320	2753	347	226	121
iECIAI1_1343	2765	342	223	119
iECIAI39_1322	2721	341	217	124
iECNA114_1301	2718	337	210	127
iECO26_1355	2780	353	224	129
iECO111_1330	2760	351	219	132
iECOK1_1307	2729	338	216	122

Continued on next page

Table 17: (Continued)

GSM model	**Total Number of Reactions**	**Number of Reactions with Cofactors**	**Reactions with Sequence**	**Missing Sequences in Reactions with Cofactors**
iEcolC_1368	2768	348	228	120
iECP_1309	2739	337	209	128
iECs_1301	2720	338	215	123
iECS88_1305	2729	335	211	124
iECSE_1348	2768	347	226	121
iECSF_1327	2742	334	217	117
iEcSMS35_1347	2746	338	221	117
iECSP_1301	2712	334	216	118
iECUMN_1333	2740	345	229	116
iECW_1372	2782	355	226	129
iEK1008	1226	272	217	55
iEKO11_1354	2778	350	229	121
iETEC_1333	2756	343	219	124
iG2583_1286	2704	335	212	123
iHN637	785	132	109	23
iIS312	519	74	48	26

Continued on next page

Table 17: (Continued)

GSM model	**Total Number of Reactions**	**Number of Reactions with Cofactors**	**Reactions with Sequence**	**Missing Sequences in Reactions with Cofactors**
iIT341	554	77	72	5
iJN1463	2927	480	285	195
iLB1027_lipid	4456	527	139	388
iLF82_1304	2726	339	214	125
iLJ478	652	100	93	7
iML1515	2712	350	251	99
iMM904	1577	271	168	103
iNF517	754	104	82	22
iNRG857_1313	2735	339	215	124
iPC815	1961	221	133	88
iS_1188	2619	312	185	127
iSB619	743	148	115	33
iSbBS512_1146	2591	318	184	134
iSBO_1134	2591	313	179	134
iSDY_1059	2539	298	169	129
iSF_1195	2630	314	183	131

Continued on next page

Table 17: (Continued)

GSM model	**Total Number of Reactions**	**Number of Reactions with Cofactors**	**Reactions with Sequence**	**Missing Sequences in Reactions with Cofactors**
iSFV_1184	2621	319	191	128
iSFxv_1172	2638	312	185	127
iSSON_1240	2693	329	205	124
iSynCJ816	1044	201	157	44
iUMN146_1321	2735	338	212	126
iUMNK88_1353	2777	353	222	131
iUTI89_1310	2725	329	214	115
iYS854	1455	221	134	87
iZ_1308	2721	338	215	123
iYO844	1250	186	148	38
iYS854	1455	221	134	87
STM_v1_0	2545	303	212	91

Table 18: GSM Model Distribution of Predicted Genes Matching and Mismatching Reaction Cofactor Usage

GSM model	Organism	Match	%	Mismatch	%
iAF692	*Methanosarcina barkeri* str. Fusaro	79	79.8	20	20.2
iAF987	*Geobacter metallireducens* GS-15	147	83.1	30	16.9
iAM_Pb448	*Plasmodium berghei*	41	78.8	11	21.2
iAM_Pc455	*Plasmodium cynomolgi* strain B	47	85.5	8	14.5
iAM_Pf480	*Plasmodium falciparum* 3D7	42	87.7	7	14.3
iAM_Pv461	*Plasmodium vivax* Sal-1	46	82.1	10	17.9
iAPECO1_1312	*Escherichia coli* APEC O1	176	85.0	31	15.0
iBWG_1329	*Escherichia coli* BW2952	187	84.0	35	15.8
ic_1306	*Escherichia coli* CFT073	176	83.8	34	16.2
iCN718	*Acinetobacter baumannii* AYE	190	91.3	18	8.7
iCN900	*Clostridioides difficile* 630	99	78.0	28	22.0
iE2348C_1286	*Escherichia coli* O127:H6 str. E2348/69	177	84.3	33	15.7
iEC042_1314	*Escherichia coli* 042	182	84.3	34	15.7
iEC1356_Bl21DE3	*Escherichia coli* BL21(DE3)	190	84.8	34	15.2
iEC1372_W3110	*Escherichia coli* str. K-12 substr. W3110	192	84.2	36	15.8

Continued on next page

Table 18: (Continued)

GSM model	Organism	Match	%	Mismatch	%
iEC55989_1330	*Escherichia coli* 55989	189	84.4	35	15.6
iECABU_c1320	*Escherichia coli* ABU 83972	176	83.8	34	16.2
iECB_1328	*Escherichia coli* B str. REL606	192	85.3	33	14.7
iECBD_1354	*Escherichia coli* 'BL21-Gold(DE3)pLysS AG'	192	85.3	33	14.7
iECDH1ME8569_1439	*Escherichia coli* DH1	184	84.0	35	16.0
iECDH10B_1368	*Escherichia coli* str. K-12 substr. DH10B	182	83.9	35	16.1
iEcE24377_1341	*Escherichia coli* O139:H28 str. E24377A	188	83.9	36	16.1
iECED1_1282	*Escherichia coli* ED1a	180	83.3	36	16.7
iECH74115_1262	*Escherichia coli* O157:H7 str. EC4115	180	85.7	30	14.3
iEcHS_1320	*Escherichia coli* HS	191	84.5	35	15.5
iECIAI1_1343	*Escherichia coli* IAI1	186	83.4	37	16.6
iECIAI39_1322	*Escherichia coli* IAI39	184	84.8	33	15.2
iECNA114_1301	*Escherichia coli* NA114	179	85.2	31	14.8
iECO26_1355	*Escherichia coli* O26:H11 str. 11368	187	83.5	37	16.5

Continued on next page

Table 18: (Continued)

GSM model	Organism	Match	%	Mismatch	%
iECO111_1330	*Escherichia coli* O111:H- str. 11128	183	83.6	36	16.4
iECOK1_1307	*Escherichia coli* IHE3034	181	83.8	35	16.2
iEcolC_1368	*Escherichia coli* ATCC 8739	192	84.2	36	15.8
iECP_1309	*Escherichia coli* 536	177	84.7	32	15.3
iECs_1301	*Escherichia coli* O157:H7 str. Sakai	183	85.1	32	14.9
iECS88_1305	*Escherichia coli* S88	177	83.9	34	16.1
iECSE_1348	*Escherichia coli* SE11	190	84.1	36	15.9
iECSF_1327	*Escherichia coli* SE15	183	84.3	34	15.7
iEcSMS35_1347	*Escherichia coli* SMS-3-5	187	84.6	34	15.4
iECSP_1301	*Escherichia coli* O157:H7 str. TW14359	184	85.2	32	14.8
iECUMN_1333	*Escherichia coli* UMN026	192	83.8	37	16.2
iECW_1372	*Escherichia coli* W	188	83.2	38	16.8
iEK1008	*Mycobacterium tuberculosis* H37Rv	175	80.6	42	19.4
iEKO11_1354	*Escherichia coli* KO11FL	191	83.4	38	16.6
iETEC_1333	*Escherichia coli* ETEC H10407	187	85.4	32	14.6

Continued on next page

Table 18: (Continued)

GSM model	Organism	Match	%	Mismatch	%
iG2583_1286	*Escherichia coli* O55:H7 str. CB9615	182	85.8	30	14.2
iHN637	*Clostridium ljungdahlii* DSM 13528	91	83.5	18	16.5
iIS312	*Trypanosoma cruzi* Dm28c	41	85.4	7	14.6
iIT341	*Helicobacter pylori* 26695	45	62.5	27	37.5
iJN1463	*Pseudomonas putida* KT2440	243	85.3	42	14.7
iLB1027_lipid	*Phaeodactylum tricornutum* CCAP 1055/1	115	82.7	24	17.3
iLF82_1304	*Escherichia coli* LF82	181	84.6	33	15.4
iLJ478	*Thermotoga maritima* MSB8	69	74.2	24	25.8
iML1515	*Escherichia coli* str. K-12 substr. MG1655	213	84.9	38	15.1
iMM904	*Saccharomyces cerevisiae* S288C	144	85.7	24	14.3
iNF517	*Lactococcus lactis* subsp. *cremoris* MG1363	72	87.8	10	12.2
iNRG857_1313	*Escherichia coli* O83:H1 str. NRG 857C	180	83.7	35	16.3
iPC815	*Yersinia pestis* CO92	121	91.0	12	9.0

Continued on next page

Table 18: (Continued)

GSM model	Organism	Match	%	Mismatch	%
iS_1188	*Shigella flexneri* 2a str. 2457T	157	85.0	28	15.0
iSB619	*Staphylococcus aureus* subsp. *aureus* N315	91	79.1	24	21.0
iSbBS512_1146	*Shigella boydii* CDC 3083-94	160	87.0	24	13.0
iSBO_1134	*Shigella boydii* Sb227	153	85.5	26	14.5
iSDY_1059	*Shigella dysenteriae* Sd197	146	86.4	23	13.6
iSF_1195	*Shigella flexneri* 2a str. 301	155	84.7	28	15.3
iSFV_1184	*Shigella flexneri* 5 str. 8401	160	83.8	31	16.2
iSFxv_1172	*Shigella flexneri* 2002017	160	86.5	25	13.5
iSSON_1240	*Shigella sonnei* Ss046	175	85.4	30	14.6
iSynCJ816	*Synechocystis* sp. PCC 6803	105	66.9	52	33.1
iUMN146_1321	*Escherichia coli* UM146	177	35	84.5	16.5
iUMNK88_1353	*Escherichia coli* UMNK88	190	32	85.6	14.4
iUTI89_1310	*Escherichia coli* UTI89	180	34	84.1	15.9
iYL1228	*Klebsiella pneumoniae* subsp. *pneumoniae* MGH 78578	193	30	86.5	13.5
iYO844	*Bacillus subtilis* subsp. *subtilis* str. 168	114	34	77	23

Continued on next page

Table 18: (Continued)

GSM model	Organism	Match	%	Mismatch	%
iYS854	*Staphylococcus aureus* subsp. *aureus* USA300_TCH1516	108	26	81	19.4
iZ_1308	*Escherichia coli* O157:H7 str. EDL933	183	32	85.1	14.9
STM_v1_0	*Salmonella enterica* subsp.	168	79.2	44	20.8