Machine Learning for Prediction of Redox Potentials in Iron-Sulfur Proteins

Laetitia Guérin

BSc Nanobiology End Project Report

Supervisors: Dr. Nikolina Šoštarić, Dr. Jos Zwanikken
Daily supervisor: Stefan Loonen
Defense committee: Dr. Jos Zwanikken, Dr. Jasmijn Baaijens

Department of Bionanoscience
Delft University of Technology
The Netherlands
9 July 2025

Contents

Nomenclature

CV	Cross-Validation
EN	Elastic Net
F3S	[3Fe-4S] cluster (Incomplete cubane)
FE	Single Fe ion (Rubredoxin cofactor)
Fe-S	Iron-Sulfur
FES	[2Fe-2S] cluster (Ferredoxin type)
GPR	Gaussian Process Regression
LR	Linear Regression
MAE	Mean Absolute Error
ML	Machine Learning
RF	Random Forest
RMSE	Root Mean Square Error
SC	Spearman Correlation
SF4	[4Fe-4S] cluster (Cubane type)
SVM	Support Vector Machine
XGB	Extreme Gradient Boosting

Abstract

Iron-sulfur (Fe-S) proteins are ubiquitous metalloproteins involved in essential redox reactions across all domains of life. Their midpoint redox potentials (E_m) are critical for function but remain challenging to predict due to structural complexity and diverse physicochemical environments. This study developed a machine learning (ML) framework to predict Fe-S protein redox potentials based on structural and physicochemical features extracted from protein structures.

A dataset of 168 redox potential entries from 130 proteins, covering four cofactor types ([4Fe-4S], [3Fe-4S], [2Fe-2S], Fe3+), was compiled from 113 publications. Features were extracted from both AlphaFold-predicted and crystal structures, incorporating cofactor implantations and energy minimizations where necessary. Seven regression models (linear, kernel-based, tree-based) were evaluated.

Results showed that tree-based models outperformed others, with Extreme Gradient Boosting (XGB) achieving the lowest mean absolute errors (MAE) across datasets (84.09 mV for radius-independent data and 88.17 mV for radius-dependent data). Random Forest achieved the best single performance of 61.49 mV MAE on [2Fe-2S] cofactors with bar features. Indications of the features that most critically affect redox potential are also provided thanks to SHAP analyses. Compared to Galuzzi et al.'s flavoprotein benchmark (36.4 mV) [Galuzzi et al., 2022], absolute errors were higher, but relative errors remained comparable (6.83–7.65% vs. 7.70%) despite a broader redox range (1196 mV vs. 472.5 mV), reflecting dataset complexity and potential for improvement.

Limitations included dataset heterogeneity, reliance on predicted structures with cofactor implantation inaccuracies, pH distribution biases in [4Fe-4S] datasets, and computational restrictions limiting hyperparameter optimization. Future directions include expanding datasets for underrepresented cofactors, integrating dynamic and quantum mechanical descriptors, and employing advanced modeling approaches such as graph neural networks.

This study demonstrates the feasibility of predicting Fe-S protein redox potentials using ML, with implications for protein engineering and biocatalyst design, and lays the foundation for improved models integrating structural, electronic, and environmental factors to better understand these evolutionarily ancient cofactors.

All code is available in a public GitHub repository at https://github.com/laetichou/FeS_Em_Prediction.git.

Keywords: iron-sulfur proteins, redox potential, machine learning, Gradient Boosting, Random Forest, SHAP analysis, hydrophobicity, burial depth, protein engineering, computational biology

Acknowledgments

I would like to thank my supervisor, Dr. Nikolina Šoštarić, and my daily supervisor, Stefan Loonen, for their guidance and support throughout this project. I would also like to thank Dr. Jos Zwanikken for helping me wrap up this project nicely and pushing me to the finish line.

I will write better acknowledgments later...

Use of AI: I would like to acknowledge the use of AI tools such as ChatGPT (by OpenAI), Claude (by Anthropic), and Grok (by xAI), which assisted me in drafting code, debugging scripts, and clarifying technical concepts during the course of this research. All writing in this report is mine, with the same AI tools used for editing purposes.

1

Introduction

The prediction of protein properties through computational methods has become a cornerstone of modern biochemistry, enabling insights into complex biological systems critical for advancing biotechnology and understanding cellular processes. Among these properties, redox potentials, particularly those of proteins containing iron-sulfur clusters, are essential due to their role in electron transfer processes fundamental to life, such as respiration, photosynthesis, and DNA repair [Cardenas-Rodriguez et al., 2018]. However, predicting redox potentials remains challenging due to the intricate interplay of structural, physicochemical, and environmental factors influencing cofactor behavior. This study attempts to address this challenge by developing a machine learning (ML) framework to predict the redox potentials of iron-sulfur proteins, leveraging physicochemical features extracted from structural data from AlphaFold, and training seven different models on this data.

1.1 Scientific Context and Motivation

1.1.1 Redox Potentials and Biological Significance

Redox potentials, measured in millivolts (mV), quantify a molecule's propensity to gain or lose electrons. A potential difference arises when two species with different redox potentials interact, sometimes leading to an electron transfer, flowing from the lower potential to the higher one. This value governs biological processes like cellular metabolism and energy production. In metalloproteins, such as iron-sulfur proteins, these potentials are finely tuned by the protein microenvironment, including ligand coordination, hydrophobicity, and pH [Sticht and Rösch, 1998]. Knowing the relationship between a protein's composition, structure, and its redox potential is crucial to understanding its physiological roles and for designing proteins for biotechnological applications. The variability of conformations of iron-sulfur clusters and their redox potentials, combined with the fact that experimental redox measurement methods are extremely laborious, underscores the need for predictive models to complement sparse experimental data. Predicting these

potentials computationally could revolutionize our understanding of protein function and enable targeted protein engineering for various applications, like enhancing nitrogen fixation in crops [Bennett et al., 2023].

1.1.2 Iron-Sulfur Clusters as Electron Transfer Specialists

Iron-sulfur clusters, comprising [2Fe-2S], [4Fe-4S], and many other configurations, are electron transfer specialists in nature, facilitating redox reactions in enzymes involved in energy production and DNA repair, amongst other things. They are highly evolutionarily conserved and present in all kingdoms of life, making them particularly important and interesting to study. Their structural diversity also makes them cover a broad range of redox potentials, allowing them to participate in many different processes, in one stoichiometry or another. This study aims to elucidate their complex structure-function relationships through computational analysis.

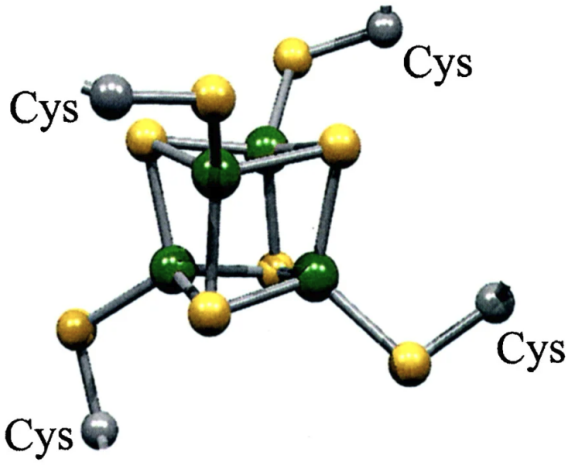

Figure 1.1: Structure of a typical [4Fe-4S] iron-sulfur cluster found in proteins. The iron atoms (green) and sulfur atoms (yellow) are in a cubic arrangement, coordinated by 4 cysteine residues (whose sulfur atoms are also colored yellow, and backbone Carbon atoms are in grey). Adapted from [Frazzon and Dean, 2001].

1.1.3 Challenges in Redox Potential Determination

Determining redox potentials experimentally and computationally presents significant challenges. Experimental methods, such as cyclic voltammetry and spectroelectrochemistry, are resource-intensive, requiring high-purity samples and stable protein conditions, often yielding sparse datasets due to difficulties in purifying proteins [Banci et al., 1999]. Furthermore, redox potential is extremely sensitive to experimental conditions, and experimental setups themselves are often susceptible to variation, leading to a lack of standardization in experimental redox values. Computational approaches, such as density functional theory (DFT) or molecular dynamics, offer insights into electronic structures but are computationally expensive and struggle with the scale of protein systems and dynamic interactions [Sticht and Rösch, 1998]. These limitations motivate the development of a tailored ML framework to predict iron-sulfur protein redox potentials efficiently and accurately.

1.2 Galuzzi et al.'s Work on Flavoproteins

Galuzzi et al.'s pioneering study on flavoprotein redox potential prediction serves as a critical benchmark for this work [Galuzzi et al., 2022]. Their dataset comprises 141 entries from experimentally determined structures (X-ray crystallography or NMR), focusing on flavin adenine dinucleotide (FAD) and flavin mononucleotide (FMN) cofactors (Figure 1.2). They extract physicochemical features from the entire protein, from around the cofactor, and from around the flavin N5 atom, which changes protonation state upon reduction, and employ six supervised regression models (Linear Regression, Gaussian Process Regression, Support Vector Regression, K-Nearest Neighbors, Random Forest, and Extreme Gradient Boosting). Their best model, Extreme Gradient Boosting, achieved a mean absolute error (MAE) of 36 mV, on par with some computational prediction methods ([Sticht and Rösch, 1998]), setting a high standard for redox potential prediction. However, flavoproteins differ greatly from iron-sulfur proteins in cofactor chemistry and structural complexity. FAD and FMN cofactors are structurally simpler than iron–sulfur clusters, as they are single organic molecules and do not require complex coordination to protein residues–they bind non-covalently to proteins. In contrast, iron-sulfur clusters are inorganic metal–sulfur assemblies with diverse structures and coordination chemistries, employing covalent bonds, making their stability and redox properties intrinsically more complex.

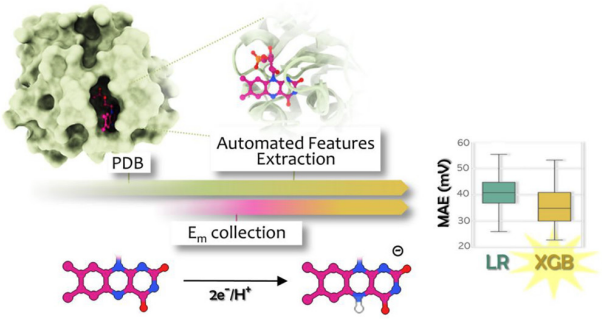

Figure 1.2: Figure summarizing the work done on the prediction of redox potentials in flavoproteins by Galuzzi et al. Reproduced from [Galuzzi et al., 2022].

1.3 The Promise of Machine Learning

Machine learning offers a scalable, data-driven approach to overcome the limitations of experimental and computational methods by modeling complex, non-linear relationships in protein data. It can find and learn underlying patterns in extremely complex data, and, once it has gone through the computationally expensive training and testing steps, it can be employed to quickly and accurately form predictions on new data. These advantages would make it ideal for redox potential prediction in iron-sulfur proteins, if it could be trained on enough data.

These considerations lead to the **research question** of this project:

Given the limited amount of experimental data available and the diversity of iron-sulfur clusters, is it possible to train a Machine Learning model to accurately predict redox potentials in iron-sulfur proteins?

Building on the success of prior ML approaches, namely Galuzzi et al.'s flavoprotein study [Galuzzi et al., 2022], this work aims to create a tailored model for iron-sulfur proteins, addressing their unique electrochemical properties. The study is structured as follows: Chapter 2 provides the theoretical background, Chapter 3 details the methodology, Chapter 4 presents the dataset, ML performance, and feature importance findings, Chapter 5 interprets these results, highlighting implications, limitations, and future directions, and Chapter 6 summarizes all the findings and attempts to answer the research question.

2

Theoretical Background

This chapter provides the theoretical framework to understand the prediction of redox potentials in iron-sulfur proteins using computational methods. It covers electrochemical principles of redox, the biological role of iron-sulfur clusters, and protein structural context, setting the stage for energy minimization and machine learning fundamentals.

2.1 Redox Potentials

"Redox" stands for "reduction-oxidation," referring to electron transfer reactions between chemical species, changing their oxidation states [OpenStax, 2019]. **Oxidation** is electron loss (increasing oxidation state), while **reduction** is electron gain (decreasing oxidation state). These occur simultaneously: the oxidized species (reducing agent) donates electrons to the reduced species (oxidizing agent).

Redox reactions are central to life functions such as photosynthesis, respiration, and metabolism, and also underlie corrosion, rusting, food oxidation, breathalyzer tests, and batteries [Österlund et al., 2010]. This section focuses on electron-transfer reactions, as they are relevant to iron-sulfur clusters.

Redox potentials quantify a species' tendency to gain or lose electrons and are expressed in millivolts (mV). Molecules with higher (more positive) potentials are more likely to gain electrons, while those with lower (more negative) potentials tend to lose them. Thus, redox potential **determines electron flow**: a molecule with a higher potential can "steal" electrons from one with a lower potential, becoming reduced while oxidizing the other. Figure 2.1 shows a sodium atom reducing chlorine to form $NaCl^{-}$ (salt).

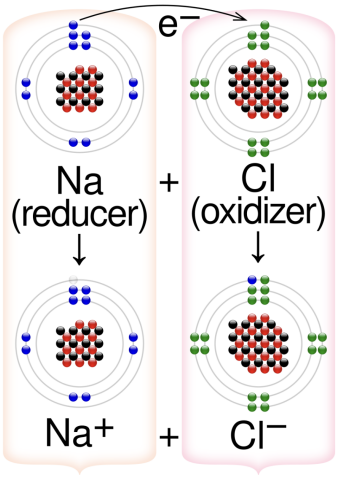

Figure 2.1: Example of a reduction-oxidation reaction yielding sodium chloride. Sodium (Na) is oxidized by chlorine (Cl), which is reduced by sodium. Adapted from [Cmglee, 2021], licensed under CC BY-SA 4.0.

2.1.1 Redox Measurement Methods

Redox values are measured against the **standard hydrogen electrode (SHE)**, defined as 0.00 V at all temperatures, corresponding to proton reduction to hydrogen gas (Equation 2.1).

$$2H^+(aq) + 2e^- \rightarrow H_2(g) \tag{2.1}$$

The Nernst equation (Equation 2.2) relates the reduction potential E to the standard potential E^0, temperature T, number of electrons n, gas constant R, Faraday's constant F, and the oxidized/reduced species concentrations.

$$E = E^0 + \frac{RT}{nF} \ln\left(\frac{[Ox]}{[Red]}\right), \tag{2.2}$$

Experimental methods. Redox potentials of proteins and cofactors are measured by **spectroelectrochemistry** and **protein film voltammetry (PFV)** with accuracies from ±1 to 10 mV under controlled conditions, though calibration, pH, and protein immobilization can increase uncertainty [Hagen, 2003].

Computational methods. These are faster and less variable than experimental methods but less accurate, with errors of ∼50–100 mV depending on the system and function [Yan et al., 2016].

Thus, experimental methods remain the gold standard, while computational estimates are valuable for rapid screening and protein engineering.

2.2 Iron-Sulfur Clusters in Biology

Iron-sulfur (Fe-S) clusters, composed of iron and sulfur atoms, are among the oldest cofactors, having existed since early anaerobic Earth conditions [Imlay, 2006], and are

highly conserved across all life forms. Their redox potentials range widely, from -600 mV to +450 mV [Jafari et al., 2022].

The simplest Fe-S cluster is an iron ion coordinated by four cysteines, forming **Rubredoxins**. More common clusters include [2Fe-2S], [3Fe-4S], and [4Fe-4S] types, shown in Figure 2.2 [Djaman et al., 2009]. Complex clusters can also involve two simpler clusters bridged by a sulfide [Rees, 2002]. Most Fe-S proteins are **ferredoxins**, electron-transfer proteins with [2Fe-2S] or [4Fe-4S] clusters, but families like nitrogenases, hydrogenases, Rieske proteins, and HiPIPs also exist [Guiza Beltran et al., 2024, García-Guerrero et al., 2021].

Cysteine is the main ligand, though histidine, aspartate, and arginine can coordinate Fe-S clusters [Berkovitch et al., 2004]. For example, [2Fe-2S] Rieske-type clusters are coordinated by 2 cysteines and 2 histidines (Figure 2.2). [4Fe-4S] clusters exist in ferredoxin- and HiPIP-type forms, the latter stabilizing higher oxidation states due to greater burial within proteins, raising their redox potentials.

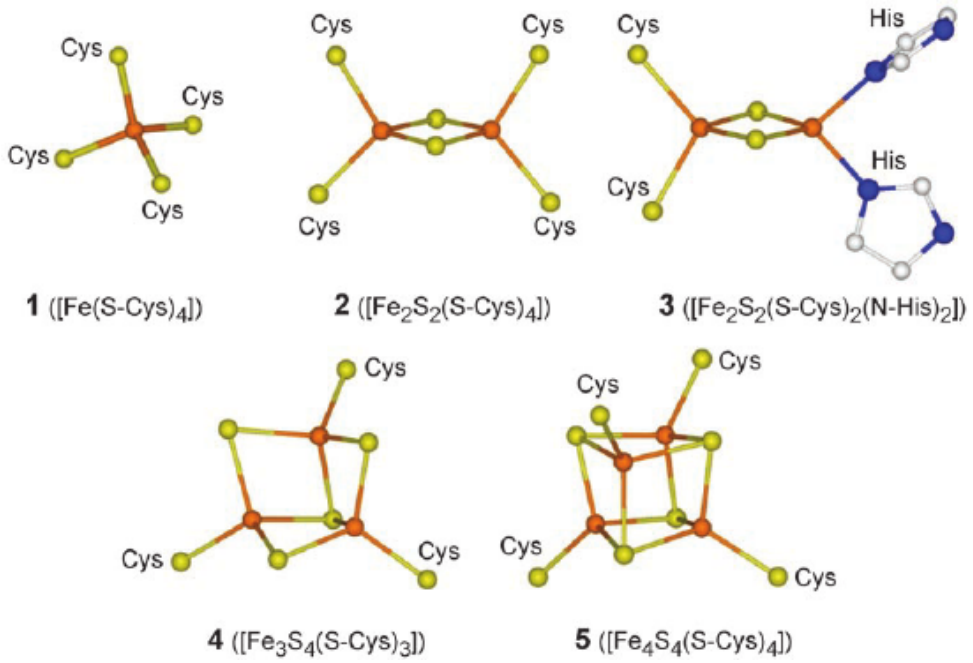

Figure 2.2: Structures of common Fe-S clusters: **(1)** Rubredoxin, **(2)** [2Fe-2S], **(3)** Rieske-type, and **(4)** [4Fe-4S]. Iron: orange; sulfur: yellow. Reproduced from [Djaman et al., 2009].

Their diverse conformations result in a wide range of redox potentials (Figure 2.3), central to many biological processes and motivating their study in this project.

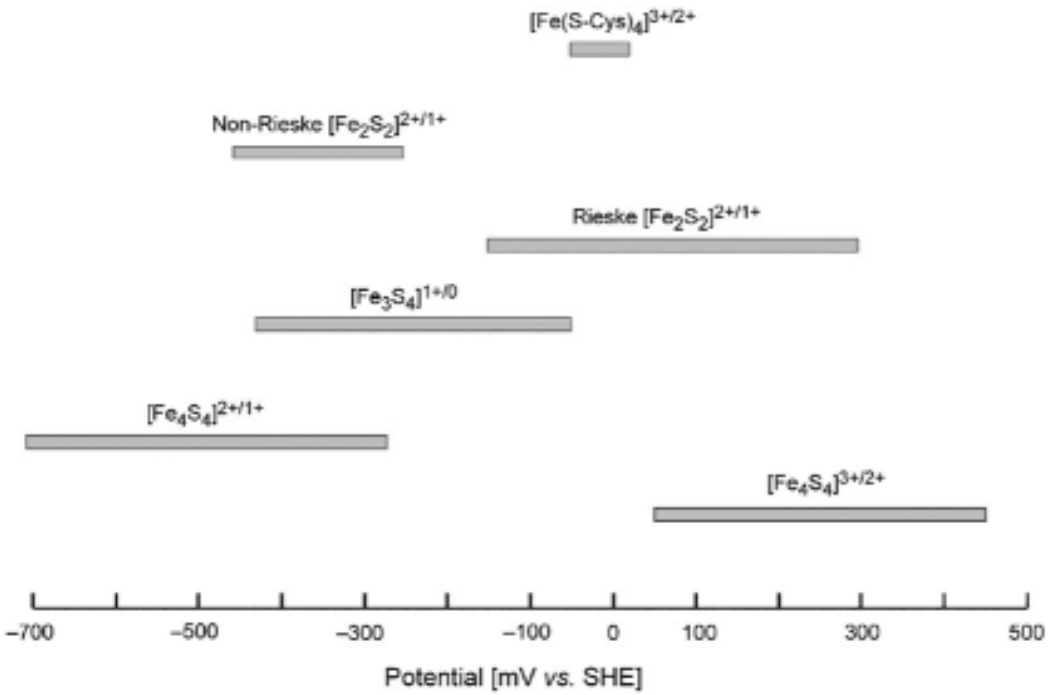

Figure 2.3: Redox ranges of Fe-S clusters: Rubredoxins, [2Fe-2S] Ferredoxins, Rieske proteins, [3Fe-4S] Ferredoxins, [4Fe-4S] Ferredoxins, and HiPIPs. Reproduced from [Djaman et al., 2009].

2.3 Protein Structure and Feature Context

Proteins are polymers of **amino acids** (AA), each comprising a carboxyl group, amino group, and side chain (R group) attached to the alpha carbon (C_α) [National Center for Biotechnology Information, 2023b]. During synthesis, peptide bonds form, creating a polypeptide chain of **residues**.

The chain forms the **primary structure**. Secondary structures, stabilized by hydrogen bonding, include α-helices and β-sheets (Figure 2.4), with loops and turns as other secondary forms. Tertiary structure is the 3D conformation driven by hydrophobic/hydrophilic interactions. Quaternary structure refers to complexes formed from multiple polypeptide chains [National Center for Biotechnology Information, 2023a].

Cysteines and histidines often coordinate Fe-S clusters (Figure 2.5) [Ahern et al., 2024].

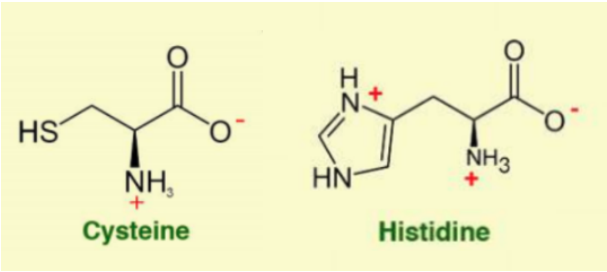

Figure 2.5: Skeletal formulas for cysteine (left) and histidine (right), key residues coordinating Fe-S clusters.

Proteins undergo **post-translational modifications** (PTMs), including phosphory-

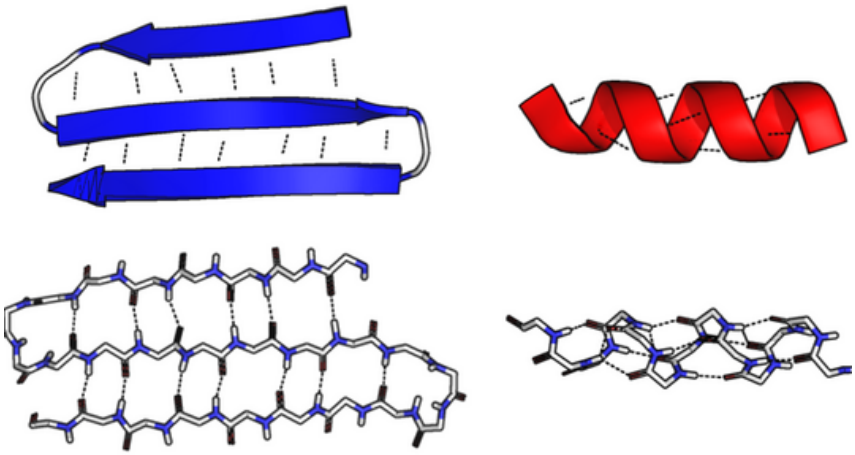

Figure 2.4: Protein secondary structures: α-helices (left, blue) and β-sheets (right, red).

lation, N-terminal methionine excision, and signal peptide cleavage [Zhong et al., 2023].

Mutations, such as AA substitutions, deletions, or insertions, can alter protein conformation and function, and are often induced to study residue roles.

Ultimately, protein structure dictates function, including redox potential, necessitating structural measurement or prediction.

2.3.1 Databases and Visualization Tools

AlphaFold Predicts protein structures via deep learning with confidence scoring [Jumper et al., 2021], used for most dataset structures.

PyMOL PyMOL is an open-source molecular visualization tool, enabling the 3D rendering and manipulation of protein structures, accessed via its Python API (Application-Programming Interface) in this project.

UniProt UniProt (Universal Protein Resource) offers a comprehensive database of protein sequences, functional annotations, and all available structures from the Protein Data Bank and AlphaFold [UniProt Consortium, 2008] (Figure 2.6).

Protein Data Bank (PDB) The PDB is a repository for experimentally determined protein structures (Figure 2.7).. It employs a standardized nomenclature for iron-sulfur clusters, which will also be used in this report, interchangeably with the nomenclature introduced in Section 2.2: namely, **FES** for [2Fe-2S], **SF4** for [4Fe-4S], and **F3S** for [3Fe-4S].

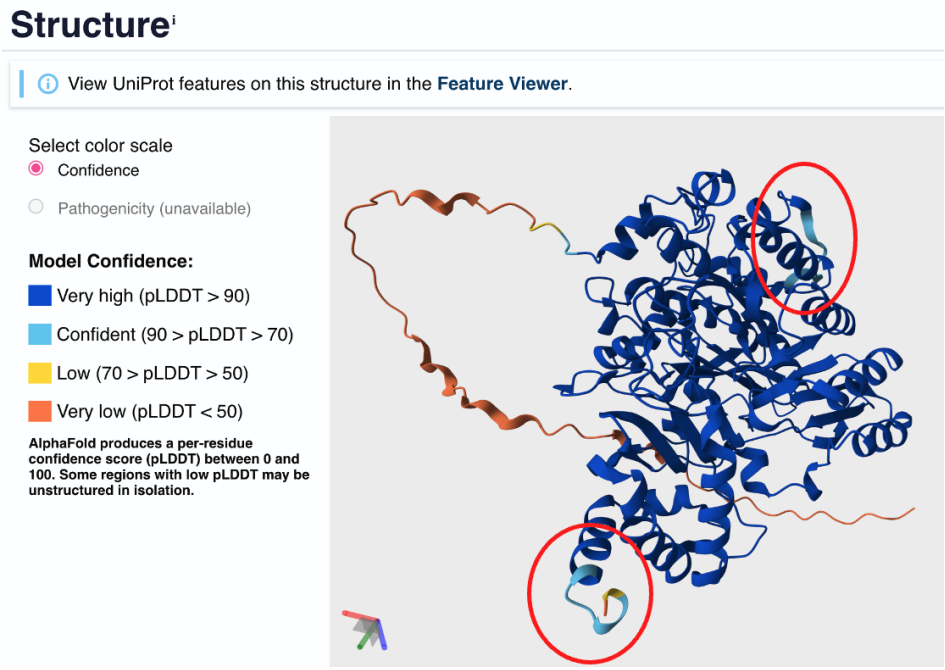

Figure 2.6: UniProt entry O23813's AlphaFold structure, colored by prediction confidence.

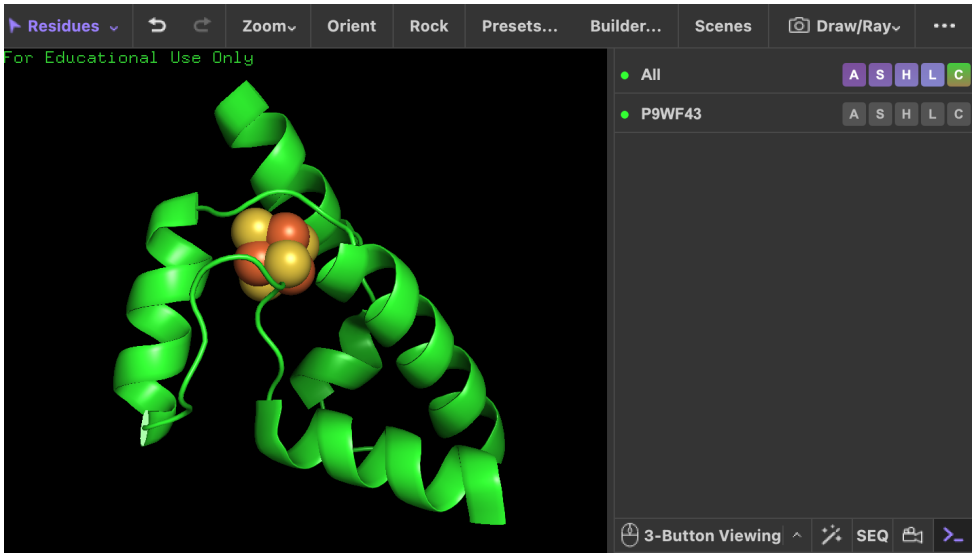

Figure 2.7: Structure of WhiB1, a [4Fe-4S] protein (PDB: 5OAY), Fe-S cluster shown as spheres.

2.3.2 Protein Structure Measurement

Most PDB structures are determined by X-ray crystallography or NMR spectroscopy. **X-ray crystallography** involves protein crystallization and X-ray diffraction to generate electron density maps for atom placement [RCSB PDB, nd], achieving resolutions to 1. Files are saved in *.pdb* format, which can be read by PyMOL. **NMR Spectroscopy** uses nuclear spin resonances to determine atomic proximities and conformations, also achieving 1 precision.

2.4 Energy Minimization of Protein Structures

Energy minimization optimizes molecular geometry by reducing potential energy, adjusting atomic positions to relieve steric clashes, and achieving physically stable conformations. It is especially important after structural modifications, such as point mutations, to eliminate unrealistic geometries.

Force Fields Potential energy is estimated using force fields—mathematical models describing atomic interactions—including:

- **Bonded interactions:** bond stretching, angle bending, torsional rotations.
- **Non-bonded interactions:** electrostatic and van der Waals forces.

Bonded interactions are modeled as harmonic springs (Equation 2.3), where r is the current bond length, r_0 is the equilibrium length, and k_b is the force constant.

$$E_{\text{bond}} = \frac{1}{2} k_b (r - r_0)^2 \tag{2.3}$$

This study uses the AMBER force field family, widely applied to proteins and nucleic acids.

Solvation and Neutralization Proteins are placed in solvent boxes (typically water) and neutralized with counterions (Na^+, Cl^-), essential for stable molecular dynamics and electrostatics.

Minimization Algorithms Algorithms like steepest descent or conjugate gradient iteratively adjust atom positions to reduce total potential energy, yielding realistic structures suitable for further analysis.

2.4.1 Machine Learning Fundamentals

Machine learning (ML) builds predictive models by learning patterns from data. Predicting redox potentials is a regression task (continuous outcome), using supervised learning with labeled datasets: structural features labeled by experimental redox values. Model training minimizes a loss function, often with regularization to prevent overfitting.

2.4.2 Core Concepts

Train-validation-test split Data is split into training, validation, and testing sets. Models learn feature-label relationships from training data, tuned using validation error (via grid search) to optimize hyperparameters, and finally evaluated on unseen test data. Figure 2.8 illustrates this process.

Feature preprocessing Features are cleaned (e.g., removing zero-variance or highly correlated features) and scaled for algorithms sensitive to input magnitude, such as Support Vector Regressor and Gaussian Process Regressor [Gambella et al., 2021].

Overfitting vs generalization Overfitting occurs when models learn training data too closely, reducing performance on unseen data. Generalization avoids this by balancing model complexity, using techniques like cross-validation and regularization.

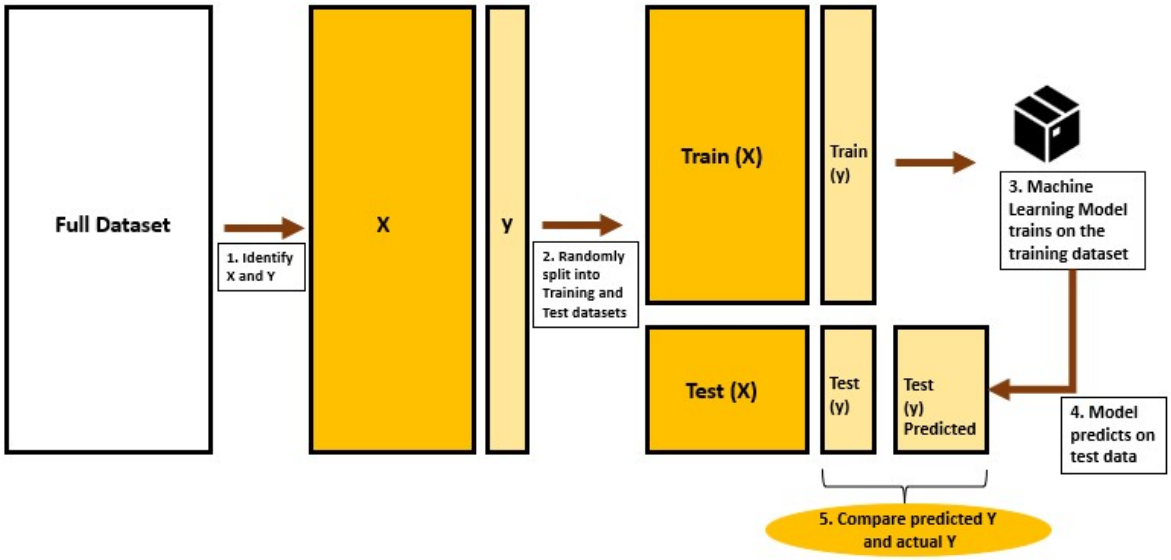

Figure 2.8: ML training workflow with train-test split. Reproduced from [Patel, 2022].

2.4.3 Cross-Validation

Cross-validation (CV) is a statistical technique used to assess the generalizability of machine learning models. In k-fold CV, the dataset is divided into k equal parts, or folds. The model is trained on $k-1$ folds and tested on the remaining fold. This process is repeated k times, each time using a different fold as the test set. The performance metrics from each iteration are then averaged to provide an overall estimate of model performance.

Using CV ensures that the model's evaluation is not dependent on a single train-test split, reduces the risk of overfitting, and provides a more robust estimate of its predictive capability on unseen data.

2.4.4 Performance Metrics

Regression performance was assessed using:

Mean Absolute Error (MAE) MAE is defined by Equation 2.4, where n is the total number of data points, y_i is the true value of the i-th data point, $\hat{y}_i$ is the predicted value of the i-th data point, and $|y_i - \hat{y}_i|$ is the absolute difference between the true and predicted values. It quantifies the **average absolute error** of the model between predicted and actual values.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{2.4}$$

R^2 The coefficient of determination assesses how well a model explains the variance in the data. It is measured by Equation 2.5, where n is the total number of data points, y_i

is the true value of the i-th data point, $\hat{y}_i$ is the predicted value of the i-th data point, and $\bar{y}$ is the mean of the true values.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{2.5}$$

Standard Deviation (SD) Standard deviation measures the spread of model performance across CV folds, defined by Equation 2.6, where x_i is each individual data point, $\bar{x}$ is the sample mean, and n is the total number of data points.

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{2.6}$$

Root Mean Square Error (RMSE) RMSE is defined by Equation 2.7, where y_i is the actual value, $\hat{y}_i$ is the predicted value, and n is the number of predictions. It penalizes larger errors more than MAE due to the squaring term (Equation 2.7).

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{2.7}$$

Spearman Correlation (SC) Measures monotonic relationship between predicted and true rankings, essentially measuring how well a model can rank proteins by their redox potentials. It is defined by Equation 2.8, where d_i is the difference in ranks of the i-th predicted and actual value, and n is the number of observations. It ranges from -1 to 1, with 1 indicating perfect monotonic agreement.

$$\rho = 1 - \frac{6\sum d_i^2}{n(n^2 - 1)} \tag{2.8}$$

Mann-Whitney Tests Assesses whether there is a statistically significant difference between the distribution of two independent samples.

2.4.5 SHAP for Interpretability

SHAP (SHapley Additive exPlanations) is used to enhance model interpretability by attributing the contribution of each feature to the predicted redox potential. Based on cooperative game theory, SHAP values give a unified measure of feature importance, providing insights into structural determinants of redox potential.

2.5 Models Overview

The study evaluated seven regression models: Linear Regression (LR), Elastic Net (EN), K-Nearest Neighbors Regression (KNN), Gaussian Process Regression (GPR), Support Vector Regression (SVR), Random Forest Regression (RF), and Extreme Gradient Boosting (XGB), each with distinct features, which are described below.

Linear Regression

Linear Regression models the relationship between features and redox potential as a linear combination, providing a baseline for comparison, as it is strongly limited by its assumption of linearity [Hastie et al., 2009].

Elastic Net Regression

Elastic Net extends Linear Regression by incorporating $L1$ and $L2$ regularization, balancing feature selection and shrinkage to handle multicollinearity in the feature set [Zou and Hastie, 2005].

K-Nearest Neighbors Regression (KNN)

KNN predicts redox potential by averaging the values of the k nearest data points in feature space, relying on distance metrics (e.g., Euclidean) and adaptable to local patterns [Syriopoulos et al., 2023].

Gaussian Process Regression (GPR)

GPR models the data as a distribution over functions, using a kernel (e.g., Radial Basis Function) to capture non-linear relationships and provide uncertainty estimates, though it is computationally intensive [Rasmussen and Williams, 2006].

Support Vector Regression (SVR)

SVR employs a kernel transformation to map features into a high-dimensional space where a linear decision boundary can be found, optimizing a margin-based loss function to handle non-linearities effectively with proper tuning [Drucker et al., 1997].

Random Forest Regression (RF)

Random Forest builds an ensemble of decision trees, reducing overfitting through bagging and feature randomness, making it robust for complex datasets [Breiman, 2001].

Extreme Gradient Boosting (XGB)

XGB enhances Gradient Boosting by optimizing tree construction with regularization and parallelization, offering high performance but requiring careful hyperparameter tuning [Chen and Guestrin, 2016].

3

Methodology

This chapter outlines the computational methodology employed to predict redox potentials of iron-sulfur proteins, building upon the theoretical foundations presented in Chapter 2. The following section describes the approach used in this work, which integrates data collection, energy minimization of protein structures to generate mutant conformations, a machine learning pipeline for model training and evaluation, and a post-analysis framework for result interpretation. For a detailed methodology, see Appendix E.

All scripts mentioned below are available in the GitHub repository at (link).

3.1 Data Collection

To build the dataset for this project, experimentally measured redox potentials for iron-sulfur proteins and their corresponding structures had to be collected.

Experimental redox potentials were sourced from the UniProt database and Scopus, a comprehensive abstract and citation database. Within UniProt, two search filters were applied, joined by an "AND" operator:

1. *Function / Cofactor / ChEBI term*: "30408"
2. *Function / Biophysicochemical properties / Redox potential*: "mV"

In Scopus, the search query is given below, with keywords in bold, and the asterisk representing any string of characters.

```
(TITLE-ABS-KEY( iron-sulfur ) AND TITLE-ABS-KEY( redox ) AND
TITLE-ABS-KEY( mv ) AND (TITLE-ABS-KEY( ferredoxin* ) OR TITLE-ABS-KEY(
rubredoxin* ) OR TITLE-ABS-KEY( rieske ) OR TITLE-ABS-KEY( hipip* )))
```

Abstracts were scanned to select only papers where the redox potential for an iron-sulfur protein was either measured or referenced with a source. Once a redox potential was found, the 3D AlphaFold structure for the corresponding protein was downloaded from UniProt. When available, the experimental structure was also downloaded for use

in the cofactor implants (Section E.4) and also for the input ML data, if a big discrepancy between the AlphaFold and experimental structures was observed.

The data for this project was recorded in the following formats:

- An Excel sheet containing one row per cofactor/redox potential, and the information listed in Table E.1.
- A folder containing the 3D protein structures from AlphaFold, as *.pdb* files.
- A folder containing the 3D protein structures from crystallography, if available, in the default *.ent* file format.

3.2 Mutant Structures

During the data collection step, some mutant entries were recorded containing 1 to 3 amino acid substitutions. Unfortunately, none of these entries have crystal structures available, and AlphaFold is not sensitive enough to these small sequence changes to provide accurate mutant predictions. Therefore, to obtain the structures of mutant entries, mutations were first induced in the wild-type AlphaFold PDB files, followed by energy minimization to ensure physically realistic conformations.

3.2.1 Tools and Packages for Energy Minimization

A combination of specialized software tools and Python packages is used to carry out energy minimization on the mutated protein structures. Each tool contributed to specific stages in the pipeline, from structural manipulation to physical optimization. Table E.2 summarizes the sequential steps and corresponding software tools used in this process.

To ensure reproducibility and avoid conflicts between software dependencies, separate virtual environments were created for different stages of the project. More information is provided in Table A.1, which summarizes the main virtual environments created for this project, their respective roles, and the key packages installed in each.

Detailed implementation of this pipeline, including code-level execution and file handling, is described in the following subsection.

3.2.2 Energy Minimization Detailed Pipeline

This subsection provides a technical description of the energy minimization pipeline used for the mutant protein structures. Each stage builds upon the tools introduced in Section E.2.1, implementing them via Python scripting or shell commands to automate a reproducible workflow.

Step 0: Mutation indices First, as the AlphaFold structure contains the entire, immature protein, while the protein sequence used by researchers typically has already undergone all PTMs, the mutation indices extracted from the papers had to be mapped to the index in the AlphaFold PDB file. Looking in the *PTM/Processing* section of a protein's UniProt page could tell me how much longer the AlphaFold sequence was than the protein used in the papers to find the new mutation index.

Step 1: Inducing mutations All mutations were induced via the PyMOL API in Python, using the script `mutate_proteins.py`.

Step 2: Assigning protonation states Protonation states are assigned using `pdb2pqr` with the script `assign_protonation.sh`, according to the pH value recorded in the Excel dataset.

Step 3: Calculating protein charge The charge of the protonated structures is calculated in the script `generate_solvation_parameters.py`, which returns the solvation parameters: a CSV file with the charge of the protein, and the number of Na+ or Cl- ions needed to neutralize the structure.

Step 4.a: `tleap` preparation The `tleap` program command files are generated for each mutant with the script `generate_tleap_inputs.py`. The solvent boxes are added with 10Å of padding around the protein, which is stripped of its hydrogens to prevent potential unrecognized H-atom errors in the next step.

Step 4.b: System preparation The `tleap` commands are executed using AMBER via the script `run_tleap_all.py`, which returns the solvated system (protein structure with hydrogens, in a water box with ions).

Step 5: AMBER to GROMACS file conversion The AMBER-format output files from **Step 4.b** are converted to formats supported by GROMACS using `ParmEd` via the script `convert_to_gromacs.py`.

Step 6: Energy minimization Finally, the energy minimization is run using GROMACS via the script `run_minimization.py`.

Step 7: File conversion from GROMACS to PDB The energy minimization outputs a lot of files, but we are only interested in the file ending in "_minim.gro" which contains the final coordinates of our system after minimization. To use these for the rest of the project, they are converted to PDB format using the script `prepare_alphafill_inputs.py`.

Step 8: Structure extraction The next step is to clean up the PDB file by removing all water atoms and ions added for minimization, leaving behind only the minimized structure. This is done with the `remove_water_box.py` script.

Step 9: Reassigning chain IDs The energy minimization removed the chain IDs, which are needed to obtain the protein sequence during the feature extraction step. This is fixed by assigning all amino acids to chain 'A,' correct since all the structures involved are monomers, using the script `add_chain_ids.py`.

3.3 Multimers

Some of the proteins in this dataset are only biologically relevant as multimers, or protein complexes. Due to lack of consistent information, it was decided that:

- If a protein complex binds one cofactor per subunit, and that subunit is modeled correctly in AlphaFold, then only that subunit with its cofactor will be used in the dataset.
- If one cofactor binds per subunit, and there is a large discrepancy between the AlphaFold subunit prediction and the experimental structure, the experimental structure is used instead.
- If a protein complex binds one cofactor in the whole complex, then the AlphaFold structure(s) for the subunit(s) will be multimerized before implanting the cofactor and using it in the dataset.

Structures are multimerized using ColabFold, a faster and more user-friendly platform for protein structure prediction using AlphaFold2 and AlphaFold-Multimer [Mirdita et al., 2022], available at ColabFold's Github (`https://github.com/sokrypton/ColabFold`).

3.4 Cofactor Implantation

Once all the protein structure files are downloaded, prepped, and converted to PDB format, they are ready to be implanted with their corresponding cofactor. In this section, a detailed implementation of the pipeline used to insert cofactors into the protein structures is described.

3.4.1 AlphaFill transplants

AlphaFill, an algorithm that "uses sequence and structure similarity to 'transplant' such 'missing' small molecules and ions from experimentally determined structures to predicted protein models" [Hekkelman et al., 2023]. The database is available at `https://alphafill.eu/`. Enriched structures were downloaded with the correct added compounds selected, from the best sequence match (25–70% identity), and optimized when possible. Figure 3.1 shows what the AlphaFill interface looks like for an enriched protein with an optimization option.

AlphaFill structures files are converted from their native *.cif* format to *.pdb* format for the rest of the analysis using `gemmi`, via the Python script `convert_cif_to_pdb.py`.

3.4.2 "Manual" transplants

Structures lacking suitable AlphaFill results were transplanted via the PyMOL API on Python. Four main protocols were used:

1. Transplants from crystal or NMR structures Experimental structures, which contain the cofactors, were aligned via backbone alignment with their corresponding AlphaFold structure, and the cofactor was copied ("implanted") into the AlphaFold structure. This is performed with the Python script `transplant_cofactors_from_crystal.py`

2. Transplants from structurally similar donors Donor proteins were chosen for the target proteins lacking a cofactor, inferred by similarity visually in PyMOL with backbone alignment. Proteins from the same family often "filled" each other. The transplant

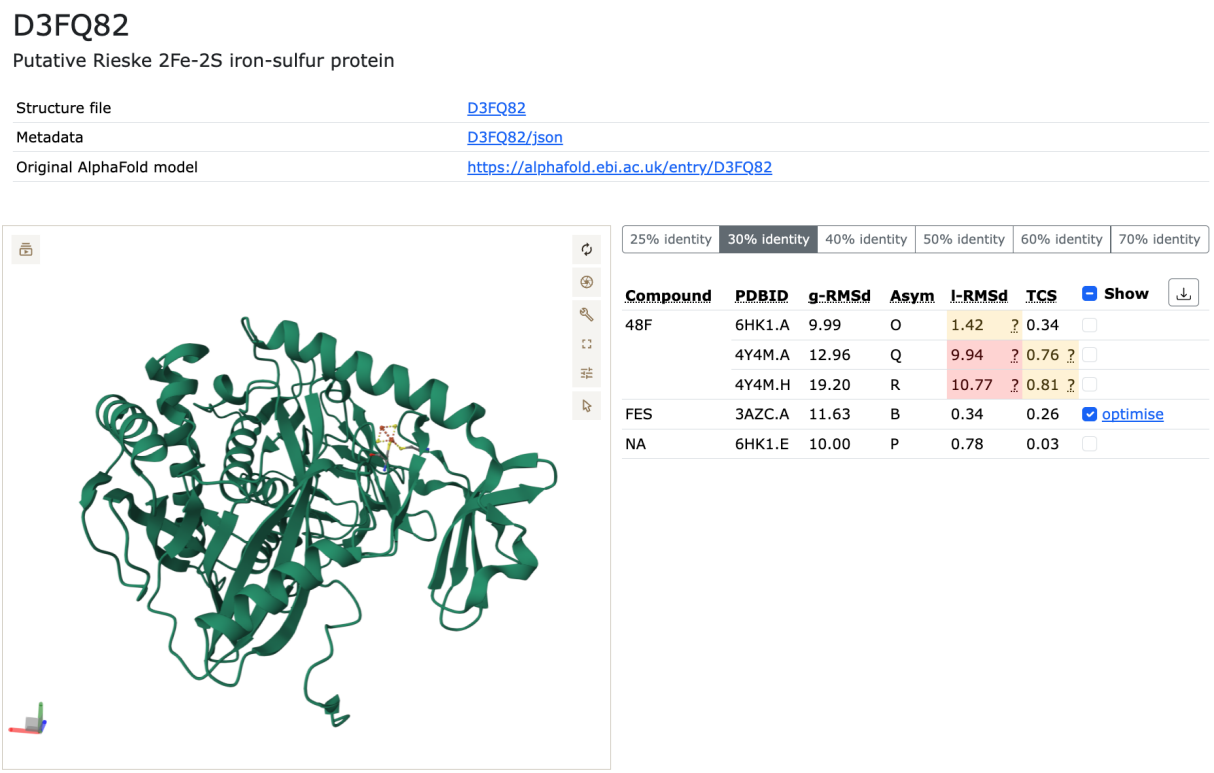

Figure 3.1: Screenshot of AlphaFill's interface showing entry D3FQ82, a Rieske protein with a [2Fe-2S] (FES) cofactor being the only compound selected to be shown on the structure, with the "optimise" button next to it in the Compound table. For this entry, 30% was the highest identity for which AlphaFill enriched the structure.

itself was performed via the `manual_cofactor_transplant.py` script, using local alignment (binding site residue alignment) when available; otherwise, backbone alignment was used. Listing E.1 shows the initiation of this manual transplant script.

Listing 3.1: Configuration and first lines of the `manual_cofactor_transplant.py` script

```
# === User Config ===
# Input files
DONOR_FILE = "donor_file_path"
TARGET_FILE = "target_file_path"
OUTPUT_FILE = "output_file_path"

# Toggle for alignment strategy
USE_LOCAL_ALIGNMENT = True  # Set to False to use backbone alignment

# Only used if USE_LOCAL_ALIGNMENT is True
DONOR_SELECTION = "resi 9+12+15+19"
TARGET_SELECTION = "resi 38+41+50+54"

# Cofactor residue names to extract
COFACTOR_RESN = ["SF4"]  # ["FES", "SF4", "FE", "FE2", "F3S"]

# Log file path
LOG_FILE = "CSV_log_file_path"

# === Start PyMOL
cmd.reinitialize()
```

```
cmd.load(DONOR_FILE, "donor")
cmd.load(TARGET_FILE, "target")

# For the CSV log file
alignment_type = "Local" if USE_LOCAL_ALIGNMENT else "Global"
```

3. Transplants from wild-type counterparts All minimized mutant structures are filled with their wild-type counterparts, using the script `transplant_cofactors_mutants.py`.

4. Transplant exceptions For proteins lacking clear binding sites, multimerized structures, and proteins that underwent a cofactor structural change (either induced or caused by a mutation), the cofactor had to be implanted directly in the PyMOL application to visually assert its position.

3.4.3 Validation and cleanup

For every cofactor transplant, the PyMOL-calculated root mean square deviation (RMSD) was recorded in a separate log file corresponding to each transplant script. Every structure was also verified visually in PyMOL to see if the correct and right number of cofactors were implanted in each structure. If a cofactor needed rearranging due to a large RMSD, coordinating residues were shown as sticks on PyMOL, and the cofactor manipulated to try to make it fit better.

Once all the protein structures are complete and validated, they are ready for the next step of the project, the feature extraction.

3.5 Feature Extraction

My feature extraction code was adapted from the methodology of Galuzzi et al (2022), extending their framework to iron-sulfur proteins by incorporating iron-sulfur cluster-specific structural features while maintaining their core concept of radius-based local feature extraction on top of protein-wide feature extraction.

The features in this project were extracted for both the whole protein and for a region within a radius ranging from 1Åto 16Åfrom the cofactor barycenter, in steps of 1, which then yields 16 datasets per run of the feature extraction script.

The feature extraction is done via the script `extract_features.py`, which imports functions from the supporting scripts `utils.py` and `utils_structure.py`.

The following subsections describe the feature extraction process in detail.

3.5.1 Feature Categories

Molecular descriptors considering only the region around the cofactor are subsequently referred to as "Bar region features" or simply "Bar features" and are prefixed in the extracted dataset with "`Bar.`", while those considering the whole protein are referred to as "protein features" and prefixed with "`Protein.`". These prefixes only apply to the first two of the four feature categories generated by the feature extraction pipeline, as described below:

Amino acid counts First, the amino acids within each region (Bar region and whole protein) are counted. These features are prefixed with "`.count`," and yield **40 features** (20 counts for each region).
Examples of these feature names: `Bar.count.ALA` (number of alanines within the specified radius around the cofactor barycenter) and `Protein.count.VAL` (number of valines in the entire protein).

Properties and means Values of 28 different physicochemical properties for all amino acids (described in Table A.2) are then calculated for each region. The features prefixed by "`.prop`" are the sum of the property values for all the amino acids in the region considered, and the features prefixed by "`.mean`" are the property value divided by the total number of amino acids in that region. This extraction step yields 28×4 features, so **112 features**.
Examples of these feature names: `Bar.mean.Hydrophobicity` (average hydrophobicity value in the Bar region considered) and `Protein.prop.P(helix)` (tendency of the amino acids in the entire protein to adopt a helix secondary structure).

Structural features The features referred to as "Structure features" include **4 features** whose names are prefixed by "`struct.`":

1. `burial_depth`: calculated as the minimum distance from the cluster barycenter to the protein surface, modeled as a convex hull using `SciPy`'s `ConvexHull` function, in Angstrom.
2. `iron_count`: the number of iron atoms in the cofactor.
3. `is_hipip`: a binary indicator of whether the protein is a HiPIP (1 if yes, 0 otherwise).
4. `is_rieske`: a binary indicator of whether the protein is a Rieske protein (1 if yes, 0 otherwise).

Sequence features These features are prefixed by "`seq.`" and are generated using `PyBioMed`'s `CalculateC` and `CalculateT` function from the Composition-Transition-Distribution (`CTD`) module. This module calculates descriptors for seven physicochemical properties of amino acids, classifying each amino acid in one of 3 predefined groups, based on literature-derived scales. These properties, groups, and their descriptions can be found in Table A.3. `CalculateC` returns the fraction of amino acids in each group, while `CalculateT` returns the frequency of the transition between amino acids of different groups along the sequence, for each property. Respectively, both these functions return 7 properties $\times$ 3 groups $= 21$ features, and 7 properties $\times$ 3 transitions $= 21$ features, so $2 \times 21 =$ **42 features** in total.
Examples of these feature names: `seq.CTDT__HydrophobicityT12` (frequency of transitions between groups 1 and 2 qua hydrophobicity) and
`seq.CTDC__SolventAccessibilityC2` (fraction of amino acids in group 2 qua solvent accessibility).

All of these molecular descriptors together, plus the pH value at which the redox potential was measured, give **199 features** for each cofactor.

3.5.2 Feature Extraction Pipeline

The following paragraphs will detail the pipeline of the feature extraction script, `extract_features.py`.

All validated protein structures are placed in a directory with file names as `{UniProt ID}_{mutation, if any}.pdb`, which serves as the input for the feature extraction script. The pH and redox potential values are then added to the feature files from the Excel dataset via the script `merge_with_ph_em.py`.

3.5.3 Dataset Division

The feature datasets were then divided into subsets, using the `create_feature_subsets.py` script. Separate datasets were then created for the 3 cofactor groups: all cofactors, only [4Fe-4S], only [2Fe-2S]. Then, for each cofactor group, 3 feature sets were created, containing: all features, only bar features, and only protein features. The structure features (with "`struct.`" prefix) and the pH and Em columns were kept in all the data subsets.

The resulting dataset names, which they will be referred to by in the rest of the report, are **complete_dataset** (all cofactors, all features), **all_cofactors_bar** (all cofactors, bar features only), **all_cofactors_protein** (all cofactors, protein features only), **FES_all** (FES cofactors, all features), **FES_bar**, **FES_protein**, **SF4_all**, **SF4_bar**, and **SF4_protein**. The "_protein" datasets, being radius-independent, contain only one dataset each.

Dividing the feature files into different subsets marks the last stage of the feature extraction phase, and the data is now ready to be run through the Machine Learning training pipeline, the implementation of which will be described in detail in the following section.

3.6 Model Training Pipeline

This section describes the methodology employed for training machine learning models to predict the redox potential of iron-sulfur proteins, based on the computational framework adapted from Galuzzi et al. for flavoproteins [Galuzzi et al., 2022]. The training pipeline, implemented in Python, integrates data preprocessing, nested cross-validation, model training, performance evaluation, and feature importance analysis. The pipeline was implemented using the Scikit-learn library [Pedregosa et al., 2011], with additional feature importance analysis conducted using SHAP (SHapley Additive exPlanations) [Lundberg and Lee, 2017] when available. The following subsections detail each component of the pipeline.

3.6.1 Data Preprocessing

The input features are loaded from CSV files containing the extracted molecular descriptors and the target variable (redox potential, E_m, in millivolts). Preprocessing steps included:

- **Feature Selection**: Non-numeric columns are excluded, and features with zero variance across samples are removed using `VarianceThreshold`. Highly correlated features (Pearson correlation > 0.95) are also eliminated to reduce redundancy.

- **Missing Value Imputation**: Missing values are filled with the median of the respective feature.
- **Scaling**: For models requiring normalized inputs, features are standardized using `StandardScaler` to ensure zero mean and unit variance.

A crucial detail is that the preprocessing is performed within each cross-validation fold to prevent data leakage, ensuring parameters are derived solely from training data.

3.6.2 Nested Cross-Validation (CV)

A nested cross-validation strategy is used for hyperparameter optimization and performance evaluation. It is repeated 10 times with varying random seeds to assess generalization performance. The outer loop employs 5-fold KFold cross-validation, while the inner loop uses 3-fold KFold cross-validation and `GridSearchCV` to tune hyperparameter combinations. Since there is one new model instantiated and trained every outer CV fold, and the entire nested CV is repeated 10 times, the final model performance metrics are the average over $5 \times 10 = 50$ models.

3.6.3 Model Selection and Training

Seven machine learning models are trained, as mentioned in section 2.5: Linear Regression, Elastic Net, Support Vector Regression (SVR), Gaussian Process Regression (GPR), K-Nearest Neighbors (KNN), Random Forest, and Extreme Gradient Boosting(XGB). Each model is integrated into a `Scikit-learn` Pipeline, incorporating preprocessing steps (scaling and feature selection) as needed. Table E.3 summarizes the models, their hyperparameter grids, and preprocessing requirements.

Feature selection is implemented within each outer CV fold to avoid overfitting by passing too many features to the models. For models needing feature selection (LR, EN, SVR, GPR, KNN), f-regression scores are used to measure the linear dependency between each feature and the target variable, in the end selecting only the top $\min(50, n/2)$ features, where n is the number of features in the training set. RF and XGB do not need feature selection as they already have built-in feature selection mechanisms through their training process, and are generally more robust to irrelevant features.

The best model for each radius/dataset is saved as a pickle file for further analysis after all iterations of the nested CV. Then, SHAP values are computed and graphs are generated for the top 3 models (for radius-independent datasets) or the top 3 model-radius combinations (for the radius-dependent datasets). These values are subsequently analyzed using the script `find_recurrent_shap_features.py` as described in Subsection E.8.6.

3.7 Performance Evaluation and Interpretation

Model performance is assessed using multiple metrics: Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), R^2 score, Spearman and Pearson correlations, and Explained Variance. These metrics are computed and saved for each fold and repeat, with means and standard deviations reported to quantify performance stability. Learning curves and prediction analysis plots (scatter, error distribution, and residual plots) are generated for each model and radius to visualize training dynamics and prediction quality.

For feature importance, SHAP analysis is performed on the top 3 performing models (based on MAE) across all radii. SHAP values are calculated using `TreeExplainer` for tree-based models (Random Forest, Gradient Boosting) and `KernelExplainer` for the others, analyzing feature contributions separately for protein-wide, local (Bar region), and all features. Summary SHAP violin plots and feature importance rankings are saved for the top 20 features per group.

3.8 Results Analysis and Visualization

The analysis and visualization of the iron-sulfur protein dataset and its machine learning (ML) predictions are conducted using a suite of custom Python scripts, tailored to handle both radius-dependent and radius-independent data. This section outlines the methodologies employed to process, analyze, and visualize the results, ensuring a comprehensive evaluation of dataset characteristics and model performance.

3.8.1 Dataset Statistical Analysis

Dataset statistics are computed using the `dataset_statistics.py` script, which analyzes structural and experimental data from PDB files, feature CSV files, and metadata Excel files. The script calculates protein size metrics using the BioPython PDB parser. Cofactor type distributions, protein family proportions, burial depths, redox potential statistics, pH values, mutation frequencies, and protein coverage are also plotted in various forms. Statistical measures (mean, median, standard deviation) are calculated, and visualizations (histograms, boxplots, and pie charts) are generated using Matplotlib and Seaborn, saved as high-resolution PNG files in a designated output directory. Detailed statistics are exported as CSV and JSON files for further analysis.

3.8.2 ML Training Result Interpretation

Dedicated scripts, `interpretation_protein.py` and `interpretation.py`, were developed to interpret and visualize ML training results from both radius-independent (processed with `ml_training_protein.py`) and radius-dependent (processed with `ml_training.py`) datasets, respectively. The script takes as input the complete output folder from the ML training script, loads the results from `ml_results.json`, and identifies the top 5 models by MAE, evaluates model stability by CV standard deviation, plots MAE and R^2 versus radius, and benchmarks the MAE performances against Galuzzi et al.'s best performance (36.5mV).

3.8.3 Comparative Heatmap Generation

Comparative heatmaps are generated using `compare_datasets_heatmaps.py` for radius-independent data and `generate_heatmaps_complete.py` for radius-dependent data. For radius-independent data, `compare_datasets_heatmaps.py` creates heatmaps with dataset types (All cofactors, FES, SF4) on the x-axis and models on the y-axis, using MAE, RMSE, and R^2 metrics. For radius-dependent data, `generate_heatmaps_complete.py` produces dataset-specific heatmaps (radii on x-axis, models on y-axis) and model-specific heatmaps (datasets on y-axis, radii on x-axis), excluding LinearRegression from color range calculations to ensure consistency across all radii.

3.8.4 Aggregated Results Analysis

To consolidate and compare ML results across multiple datasets, two dedicated scripts were developed: `aggregate_radius_dependent.py` for radius-dependent data and

`aggregate_radius_independent.py` for radius-independent data. These scripts aggregate results from multiple output folders, each containing `ml_results.json` files, and generate comprehensive visualizations and reports. The scripts asses performance trends by performing linear regression on MAE versus radius, generates stability heatmaps with CV standard deviations, and computes mean MAE and mean standard deviations per model across all datasets, extracting the top 5 models according to mean MAE.

3.8.5 SHAP Violin Plot Analyses

A Python script, `generate_shap_table.py`, was developed to automate the identification of recurrent features and generate a structured output. The script outputs a CSV file in the `output/` directory, with columns for Feature, Description, and binary indicators for recurrence in all the CSV files.

3.8.6 Results Availability

Results are systematically stored in structured output directories. Training outputs include JSON files, CSV summaries, and serialized best models as pickle files. Grid search parameters are saved as JSON files. Visualizations encompass performance plots, SHAP summaries, model stability analyses, comprehensive comparisons, heatmaps, and aggregated collages, all saved as high-resolution PNG files. Analysis and aggregation reports are saved as .txt files, ensuring accessibility for further review.

The ML model training and visualization pipeline is summarized in Algorithm 1.

3.9 Supercomputer Usage

All the ML training scripts were run using DelftBlue, the TU Delft's supercomputer. I first recreated my `redox-env` virtual environment on DelftBlue, according to DelftBlue's documentation (https://doc.dhpc.tudelft.nl/delftblue/howtos/conda/), and using the Windows/Linux-adapted `portable_environment.yml` file I generated from my `redox-env`. The resources I asked of DelftBlue are shown in Listing E.2. Running both ML training scripts on all datasets with these resources took 23 hours.

3.10 Code Availability

The custom Python scripts developed for this study are available to support reproducibility and transparency. These scripts implement the statistical analysis, ML training result interpretation, and heatmap generation described in Sections E.8. The code, training data, and results are hosted on a public GitHub repository at https://github.com/laetichou/FeS_Em_Prediction.git. All references for the papers where the data was collected can be found in the Excel datasheet on the GitHub repository.

4

Results

This chapter presents the empirical findings from the analysis of the iron-sulfur protein dataset and the machine learning (ML) models trained on both radius-independent and radius-dependent data.

4.1 Introduction to Results

The final collected dataset comprised 168 entries (redox potentials) from 130 proteins. This chapter starts by presenting a statistical analysis of the collected dataset, showing various distributions, proportions, and detailed statistics.

Then, the ML training results are interpreted, treating radius-independent datasets (protein features only for all cofactors, SF4 cofactors, and FES cofactors–three datasets) separately from radius-dependent datasets (all features and bar features only, for the same three sets of cofactors–six datasets in total). For both sets, the overall MAE statistics across all datasets in that set are shown for all models, and the performance MAE and R^2 statistics for the top five model-dataset combinations are shown as tables. For radius-dependent datasets, graphs showing model stability and performance by MAE across all radii are also presented. A table comparing the best MAE to Galuzzi et al.'s results is shown at the end, combining results from both radius-dependent and radius-independent datasets.

Then, the overall best performances are chosen according to specific criteria: the best model according to the lowest MAE across all datasets, and the best model-dataset combination according to the lowest MAE. Heatmaps showing these models' performance are shown. Finally, a SHAP analysis graph is shown for the top model trained on the complete dataset, and the top features extracted in other SHAP graphs are summarized as tables for all data subsets. The detailed methodology for these results' analysis and visualization is described in Section E.8.

The following section uses the following nomenclature for the ML models: EN for Elastic Net, GPR for Gaussian Process Regressor, XGB for Gradient Boosting Regressor,

KNN for K Neighbors Regressor, RF for Random Forest Regressor, and SVR and Support Vector Regressor.

4.2 Dataset Statistical Analysis

4.2.1 Cofactor Distribution and Protein Coverage

Figure 4.1 shows two pie charts. On the left one, the chart breaks down the four cofactor types in the dataset ([4Fe-4S], [3Fe-4S], [2Fe-2S], Fe3+) with their counts and proportions. The right chart shows the number of unique proteins in the dataset and their distribution across cofactor types.

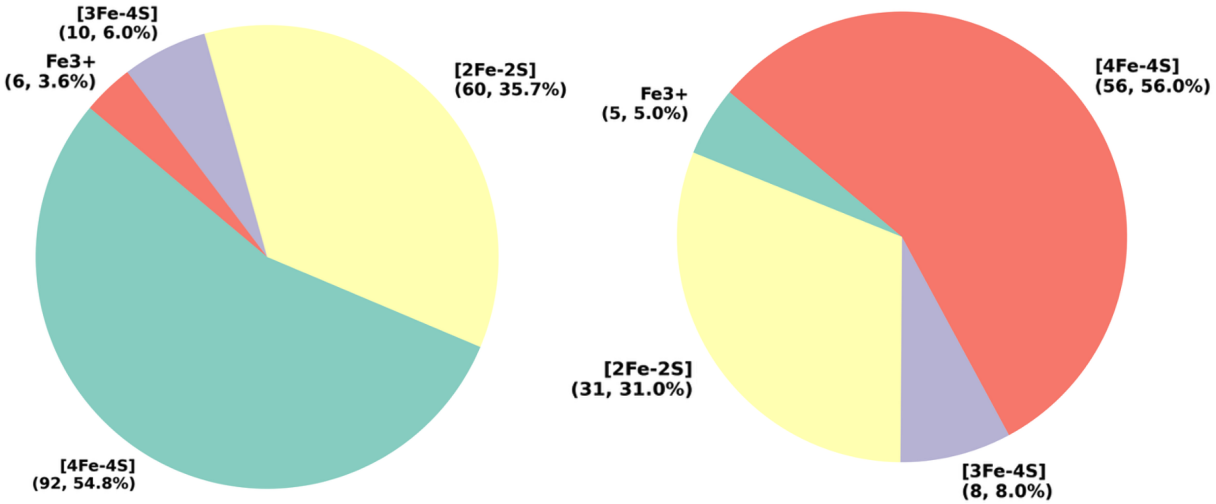

Figure 4.1: Pie charts of cofactor type distribution and unique protein coverage.

Figure 4.2 shows two pie charts: on the left are the proportions of unique [4Fe-4S] proteins that are HiPIPs, and on the right, the proportions of unique [2Fe-2S] proteins that are Rieske proteins.

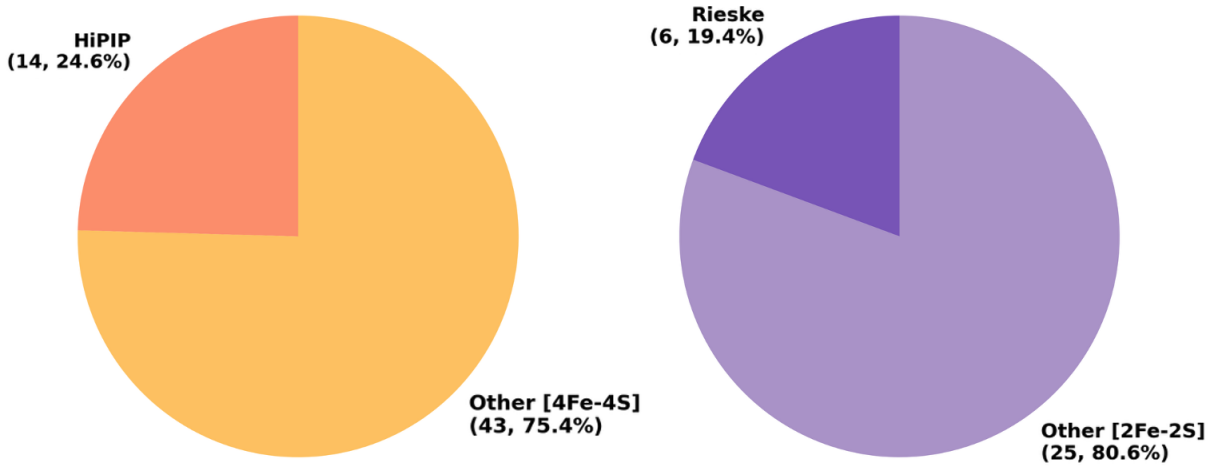

Figure 4.2: Pie charts of HiPIP and Rieske proportions among [4Fe-4S] and [2Fe-2S] proteins.

4.2.2 Mutation Analysis

Figure 4.3 shows the proportion of wild-type vs. mutant entries in the dataset. We can see that approximately one-third of the dataset is comprised of mutant entries.

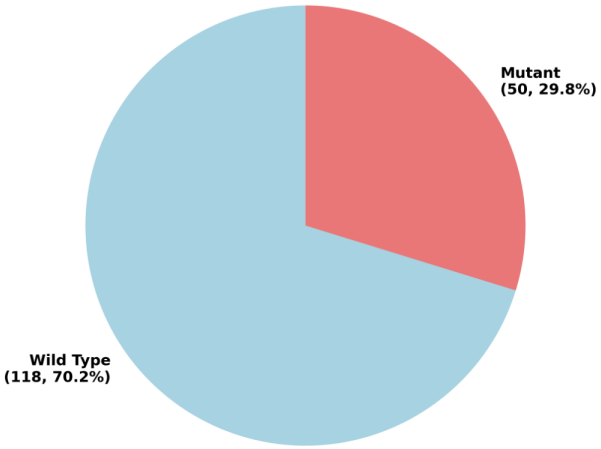

Figure 4.3: Pie chart of mutation proportions.

4.2.3 Protein Size and Structural Characteristics

Figure 4.4 shows the distribution of protein sizes in the dataset, by residue count (left) and by maximum diameter in Angstrom (right). Table B.1 in Appendix B presents these data in more detail, including median and standard deviation, and divides the proteins according to their cofactor. Burial depth distribution by cofactor type is shown in Figure 4.8.

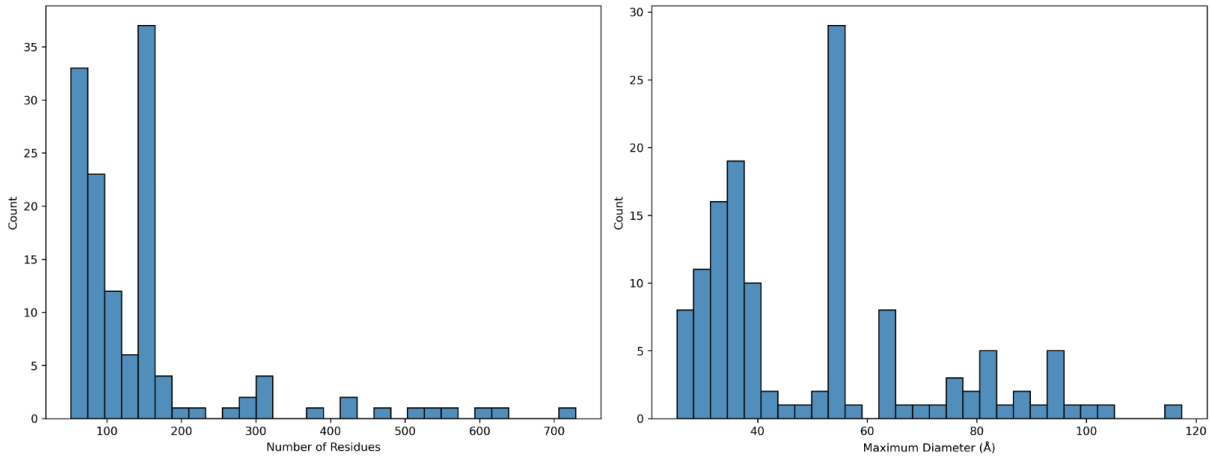

Figure 4.4: Distribution of protein sizes by residue count and maximum diameter.

4.2.4 Redox Potential and Experimental Conditions

The distribution of redox potentials and their experimental conditions (pH at measurement) are presented in Figures 4.5 and 4.6, respectively. As expected, most experimental pH levels are around the physiological pH of 7.4. Redox potentials per cofactor are shown in Figure 4.7. We can see that the ranges match those given in Section 2.2, the outliers

for the boxplots of [2Fe-2S] and [4Fe-4S] cofactors corresponding to Rieske proteins and HiPIPs, respectively. The more detailed statistics (mean, median, standard deviation) of burial depth and redox potential per cofactor are available in Table B.2.

The pH distribution across the dataset was analyzed in more depth to understand its recurrence in SF4_bar (Table 4.14). Table 4.1 shows that 38 entries have high pH values ($\geq$ 7.6), with 30 (78.9%) corresponding to SF4 cofactors, compared to 8 (21.1%) for non-SF4 cofactors (FES, [3Fe-4S], Fe3+). Most pH values cluster around physiological pH (7.0), with SF4 entries showing a higher proportion of high pH values (30/92, 32.6%) compared to non-SF4 entries (8/76, 10.5%).

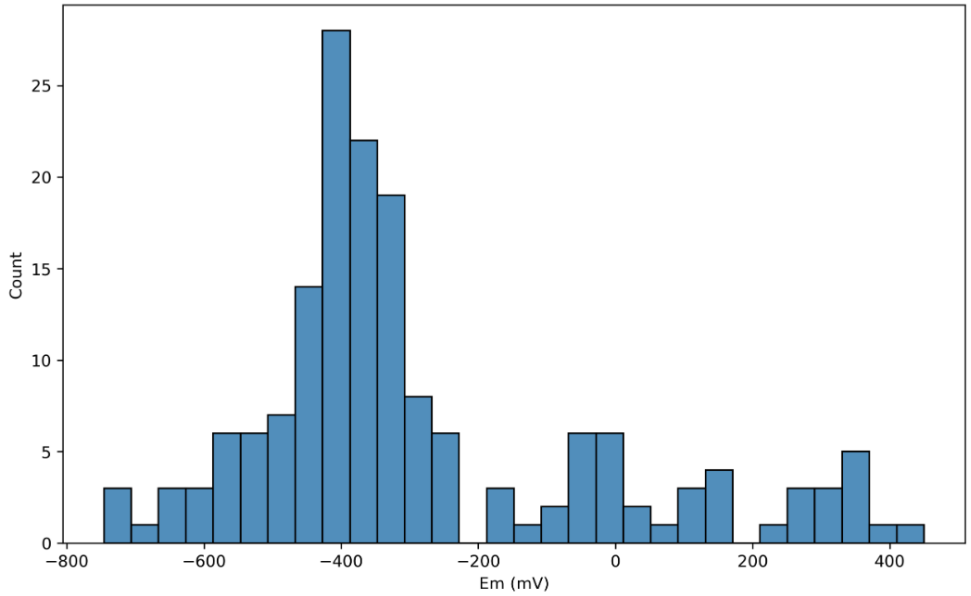

Figure 4.5: Distribution of redox potentials (E_m).

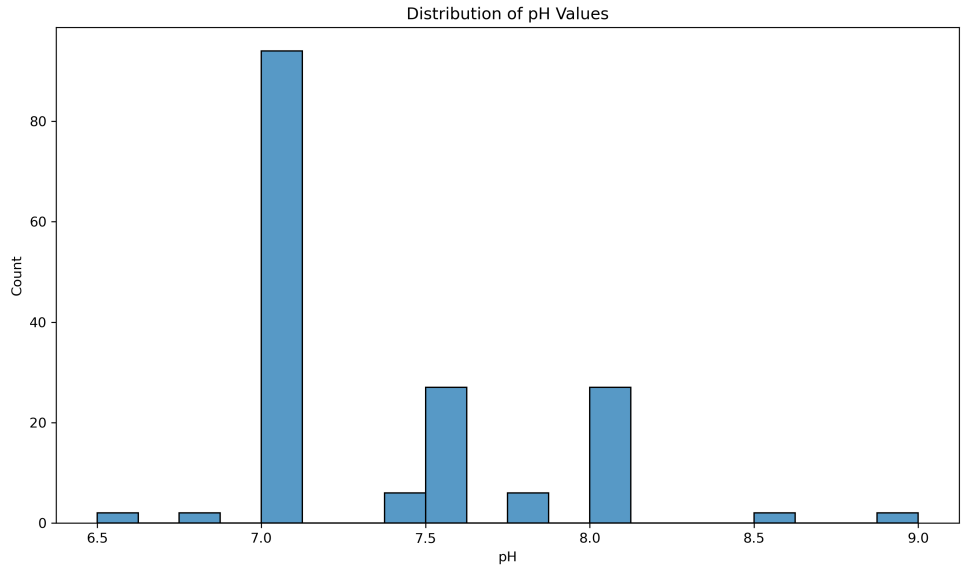

Figure 4.6: Distribution of pH values.

Table 4.1: pH Distribution Across Cofactor Types. All values are absolute counts, extracted directly from the Excel dataset "Cofactor" and "pH" columns.

Cofactor	**Total Entries**	**pH** < 7	$7 \leq$ **pH** < 7.6	**pH** ≥ 7.6
All	168	4	126	38
[2Fe-2S]	60	2	55	3
[4Fe-4S]	92	1	61	30
[3Fe-4S]	10	1	4	5
Fe3+	6	0	6	0

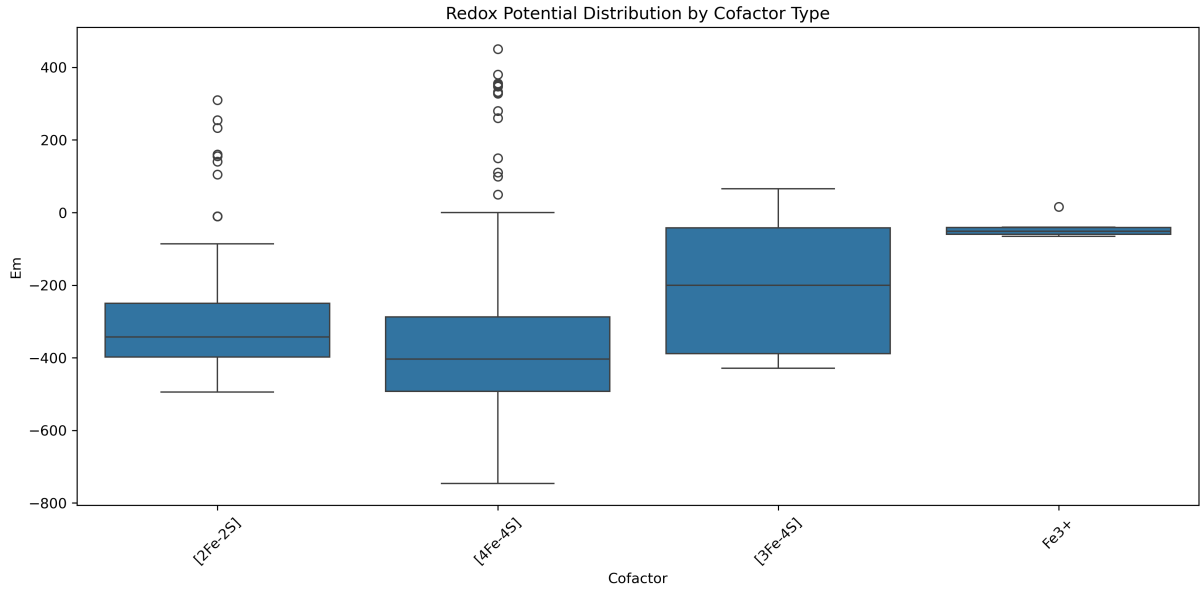

Figure 4.7: Boxplot of redox potential (E_m) by cofactor type.

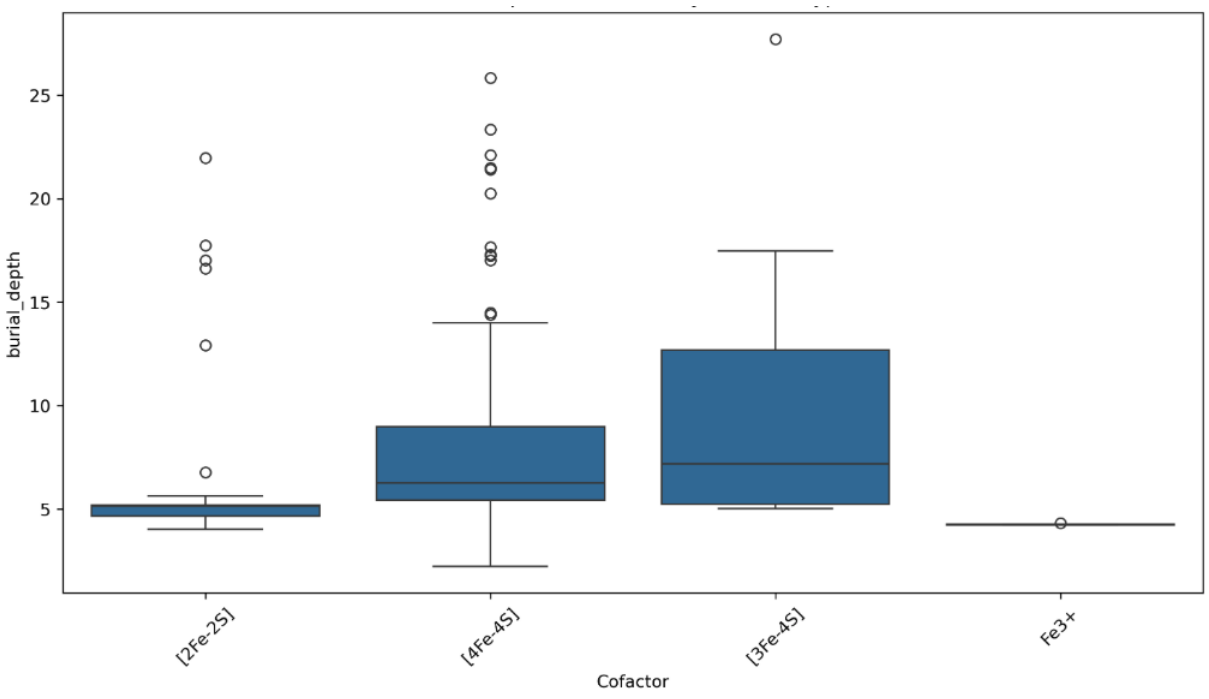

Figure 4.8: Boxplot of burial depth by cofactor type.

4.2.5 Cofactor Implant Validation

Cofactor implants were validated with PyMOL's alignment RMSD, the detailed statistics of which are presented in Table 4.2. They are calculated once with all data, and once excluding an outlier with an RMSD of 14Å. This outlier is entry P73276, which only has one residue listed as a binding site on its UniProt page, which likely explains the inaccuracy in implanting the cofactor. IT is a subunit of a larger protein, which probably means part of the binding site is missing in the AlphaFold structure.

Table 4.2: RMSD statistics for cofactor implant alignments. Values are in Å, comparing alignments with and without outliers removed. The outlier threshold is set to 10Å. Both mean RMSDs, 0.53Å and 0.38Å, are under 1Å. indicating high accuracy.

Statistic	**With Outliers**	**Without Outliers**
Count	98.0	97.0
Mean	0.53	0.38
Std Dev	1.60	0.70
Median	0.10	0.09
Min	0.0	0.0
Max	14.71	3.83

4.3 ML Training Result Interpretation

4.3.1 Radius-Independent Model Performance

The heatmap of model performance across dataset types (All cofactors, FES, SF4) according to MAE is shown in Figure 4.9. While Linear Regression is included in this heatmap, it is not included in the range for the color mapping to avoid skewing the scale too much. The data on this heatmap is organized in a more legible format below, in Tables 4.3 and 4.4, which clearly show the best performing models and datasets. RF achieves the best singular performance on the FES dataset, at 77.0mV, while XGB follows closely at 77.2mV on the same dataset, while also outperforming all other models on the other two datasets (93.6mV for all cofactors and 81.6mV for SF4 cofactors).

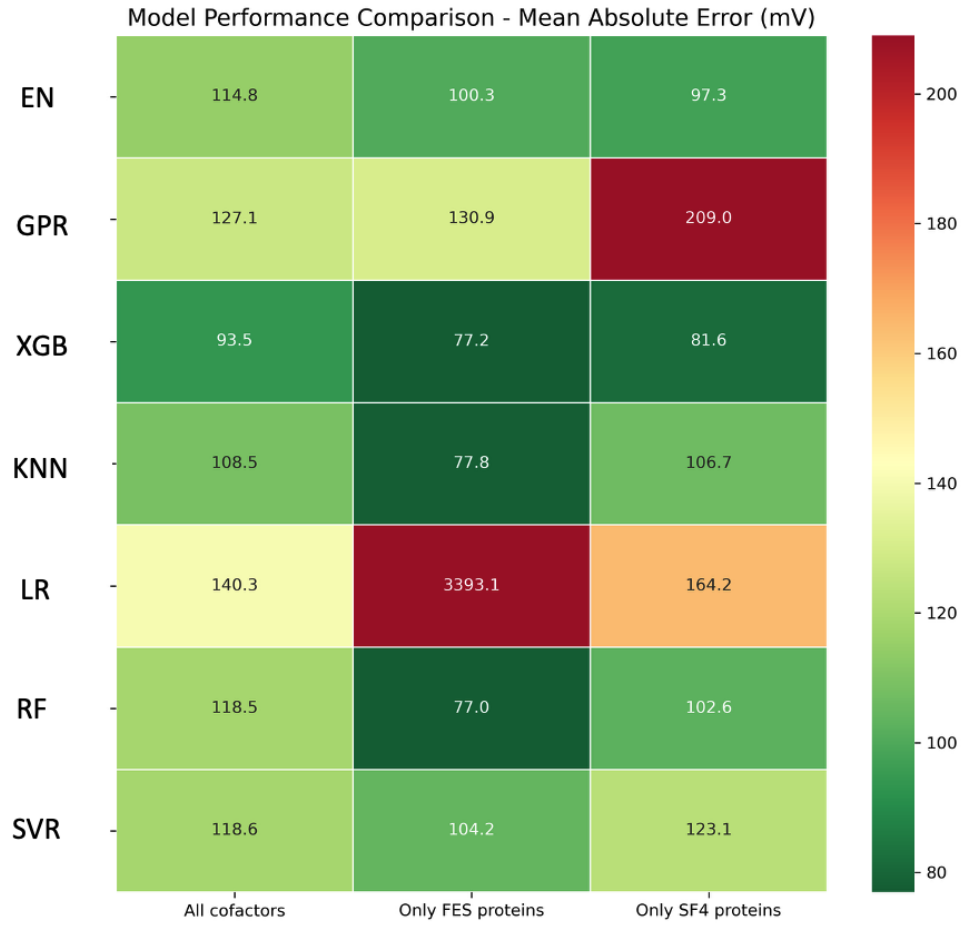

Figure 4.9: Heatmap of model performance, as measured by MAE in mV, across radius-independent datasets. The best performance is achieved by RF on the FES dataset, at 77.0mV, followed closely by XGB at 77.2mV, also on the FES dataset.

The top five model-dataset combinations by MAE, including MAE, RMSE, R^2, and SC values, are presented in Table 4.3, essentially summarizing the heatmap in Figure 4.9. We see that the FES dataset and XGB both appear in three of the top five combinations.

Table 4.3: Performance metrics for the top 5 model-dataset combinations by MAE across radius-independent datasets. Means of MAE (mV), RMSE (mV), R^2, and SC are shown, with standard deviations indicated by $\pm$. FES datasets and XGB both appear three times.

Model (Dataset)	**MAE**	**RMSE**	R^2	**SC**
RF (FES)	76.98 ± 31.33	114.80 ± 45.54	0.552 ± 0.341	0.652 ± 0.214
XGB (FES)	77.15 ± 32.64	114.84 ± 51.45	0.498 ± 0.488	0.639 ± 0.224
KNN (FES)	77.76 ± 35.66	118.42 ± 55.69	0.492 ± 0.459	0.658 ± 0.190
XGB (SF4)	81.59 ± 22.54	124.55 ± 33.47	0.761 ± 0.225	0.795 ± 0.094
XGB (All)	93.53 ± 20.56	139.12 ± 29.45	0.672 ± 0.159	0.774 ± 0.090

Overall MAE, RMSE, R^2, and SC statistics across all radius-independent datasets for all models are shown in Table 4.4. XGB presents the lowest mean MAE at 84.09mV,

13.56mV lower than the next-best model, KNN, with a mean MAE of 97.65mV.

Table 4.4: **Aggregated performance metrics for all models except LR across all radius-independent datasets.** Means of MAE (mV), RMSE (mV), R^2, and SC are shown, with standard deviations indicated by ±. XGB outperforms all other models according to all these metrics.

Model	MAE	RMSE	R^2	SC
XGB	84.09 ± 8.47	126.17 ± 12.22	0.644 ± 0.134	0.736 ± 0.085
KNN	97.65 ± 17.25	141.51 ± 20.01	0.582 ± 0.079	0.691 ± 0.031
RF	99.36 ± 20.94	138.74 ± 22.89	0.621 ± 0.093	0.708 ± 0.051
EN	104.12 ± 9.34	143.66 ± 13.04	0.545 ± 0.250	0.635 ± 0.127
SVR	115.31 ± 9.89	173.55 ± 13.18	0.384 ± 0.264	0.680 ± 0.055
GPR	155.66 ± 46.24	200.63 ± 59.82	0.080 ± 0.361	0.629 ± 0.113

4.3.2 Radius-Dependent Model Performance

The top five model-dataset combinations by MAE at optimal radii, including MAE and R^2 values, are shown in Table 4.5. We see that all combinations concern an FES dataset, and RandomForest appears twice.

Table 4.5: **Top five machine learning model-dataset combinations by MAE at optimal radii (Å) for radius-dependent datasets.** Means of MAE (mV), RMSE (mV), R^2, and SC are shown, with standard deviations indicated by ±. These best performances were all on FES datasets.

Model (Dataset)	MAE	RMSE	R^2	SC
RF (FES_bar, 13)	61.49 ± 28.39	92.72 ± 47.40	0.732 ± 0.159	0.715 ± 0.174
KNN (FES_all, 16)	62.86 ± 22.73	90.69 ± 35.22	0.713 ± 0.201	0.701 ± 0.168
RF (FES_all, 11)	64.69 ± 29.74	99.19 ± 44.25	0.691 ± 0.172	0.732 ± 0.206
EN (FES_bar, 11)	65.00 ± 24.43	92.31 ± 37.39	0.719 ± 0.148	0.654 ± 0.194
GPR (FES_all, 12)	65.31 ± 26.68	95.46 ± 38.59	0.667 ± 0.315	0.758 ± 0.157

Overall MAE statistics across all radius-dependent datasets for all models except LR are shown in Table 4.6. As with the radius-independent datasets, XGB presents the lowest mean MAE at 88.17mV, 6.17mV lower than the next-best model, KNN, with a mean MAE of 94.34mV.

The best-performing radius for each model and dataset type is provided in Table 4.7. It can be seen that the non-tree-based models EN, GPR, SVR, and KNN perform better at higher radii (>9Å), while the tree-based models XGB and RF have more variation in their optimal radii. Per dataset, FES_all, FES_bar, and all_cofactors_bar seem to lead to better model performance at higher radii (between 9 and 16Å), while the other datasets have more variation.

Table 4.6: **Aggregated performance metrics for the top 5 models across all radius-dependent datasets.** Means of MAE (mV), RMSE (mV), R^2, and SC are shown, with standard deviations indicated by ±. XGB is the best model by MAE and RMSE, while RF outperforms the others by R^2 and SC.

Model	**MAE**	**RMSE**	R^2	**SC**
XGB	88.17 ± 9.60	128.31 ± 13.92	0.635 ± 0.174	0.706 ± 0.118
KNN	94.34 ± 15.32	139.26 ± 19.43	0.603 ± 0.122	0.689 ± 0.086
RF	95.54 ± 16.73	136.15 ± 21.41	0.645 ± 0.094	0.710 ± 0.106
EN	105.29 ± 13.62	147.72 ± 19.35	0.530 ± 0.221	0.603 ± 0.114
SVR	150.28 ± 24.82	229.80 ± 39.04	0.091 ± 0.050	0.636 ± 0.123
GPR	121.49 ± 31.84	164.17 ± 37.90	0.405 ± 0.210	0.653 ± 0.101

Table 4.7: For the top 6 models (all excluding Linear Regression), the optimal radius for each dataset type is shown, in Angstrom. All optimal radii for FES_all and FES_bar are above 9Å, and all optimal radii for all_cofactors_bar are at 12Å, except for RF at 5Å. Tree-based models (XGB and RF) have more range in their optimal radii, while non-tree-based models perform better at higher radii (>10Å).

Dataset	EN	GPR	XGB	KNN	RF	SVR
	Optimal Radius (Å)					
FES_all	9	12	15	16	11	11
FES_bar	11	15	15	15	13	11
SF4_all	16	12	3	13	8	11
SF4_bar	16	14	1	12	1	16
all_cofactors_bar	12	12	12	12	4	12
complete_dataset	5	12	1	14	14	15

The stability of model predictions, in MAE standard deviation, for all radius-dependent data aggregated per radius, is shown in the heatmap in Figure 4.10. LR was excluded because its values were too large to fit within the cells of the heatmap. XGB and Random Forest show the most consistently low values, followed by KNN and EN, meaning that these are the most stable models, suggesting that their predictions are more reliable and not overly sensitive to changes in the input data's spatial context (radius around the cofactor). The average standard deviation averaged across radii for all models except LR is available in Table 4.6.

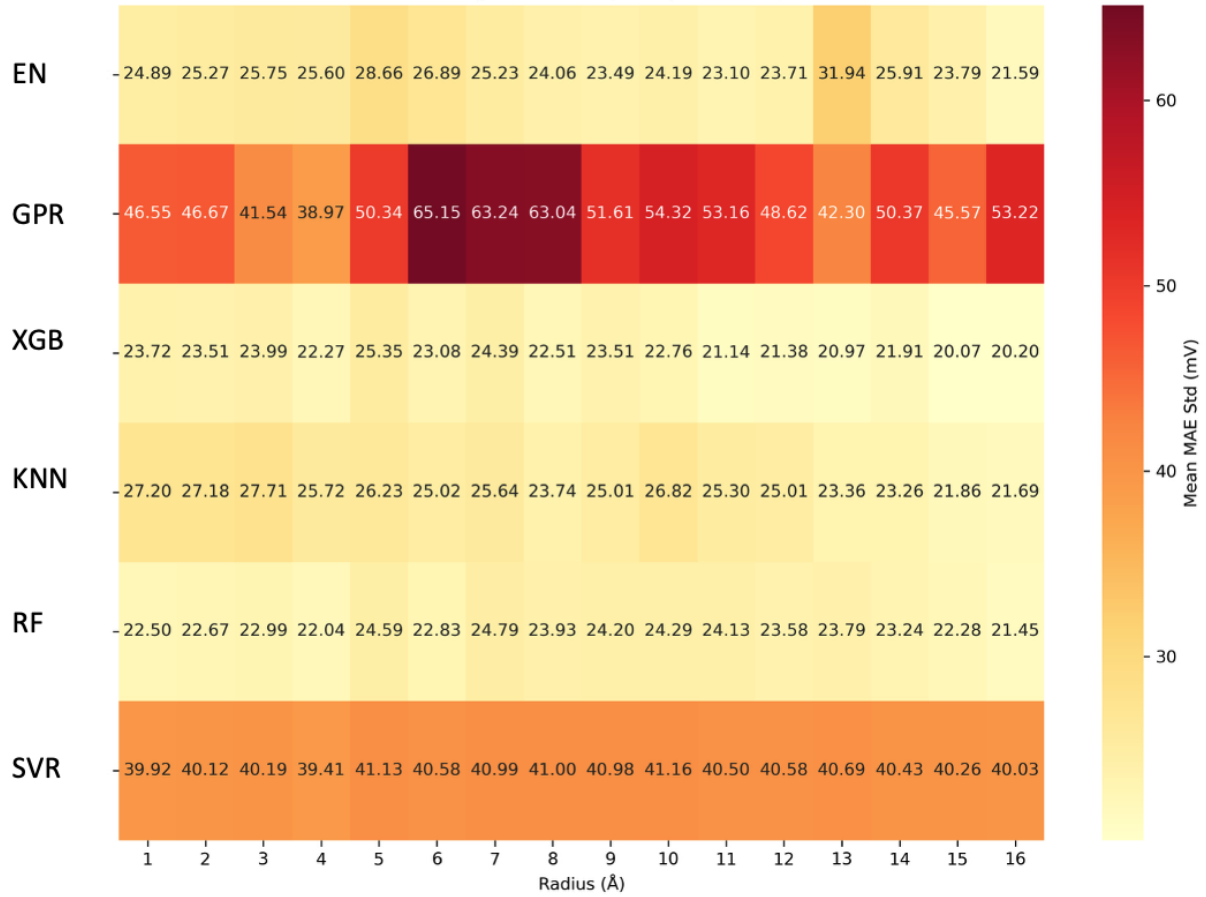

Figure 4.10: **Stability analysis for radius-dependent data** (excluding LinearRegression), showing mean MAE standard deviation across all radii, aggregated for all datasets. XGB and RF emerge as the most stable models, with similar standard deviation across all radii. They are followed closely by KNN and EN.

The following Figures 4.13 and 4.12 are colored according to this legend below in Figure 4.11.

Figure 4.11: Legend for color-coding of models in subsequent Figures 4.13 and 4.12.

Performance trends per radius for the three radius-dependent datasets with all features are shown in Figure 4.12, for all models except Linear Regression, since its bad performance skewed the scale of the graph too much. Trends are difficult to see, except for the FES_all dataset, where performance seems to improve as the radius gets larger. However, there is no clear optimal radius, which confirms the data shown in Table 4.7. The performance trends for the other three radius-dependent datasets can be found in Appendix B.2.

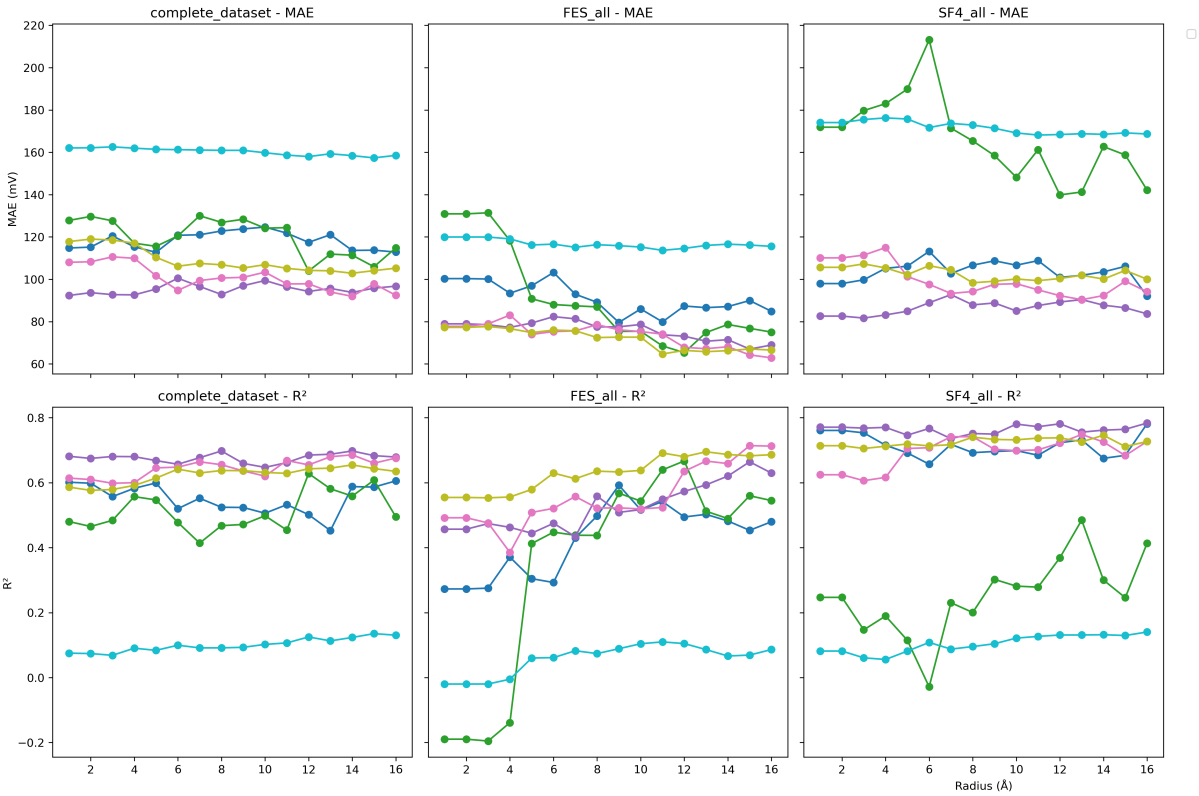

Figure 4.12: **Performance trends (MAE vs radius and R^2 vs radius) for all models except Linear Regression across three radius-dependent datasets (complete_dataset, FES_all, SF4_all)**. Performance improves with radius on the FES_all dataset, while trends remain unclear in the other datasets.

Figure 4.13 shows boxplots for the MAE distribution of each model per dataset, aggregated over all radii. It is a different way of representing the data in Heatmap 4.10– aggregated across radii rather than datasets. XGB, in purple, seems to perform consistently better than other models, with a lower MAE and smaller standard deviations, except for datasets FES_bar and FES_all, where it is slightly outperformed by Random Forest Regressor, in yellow.

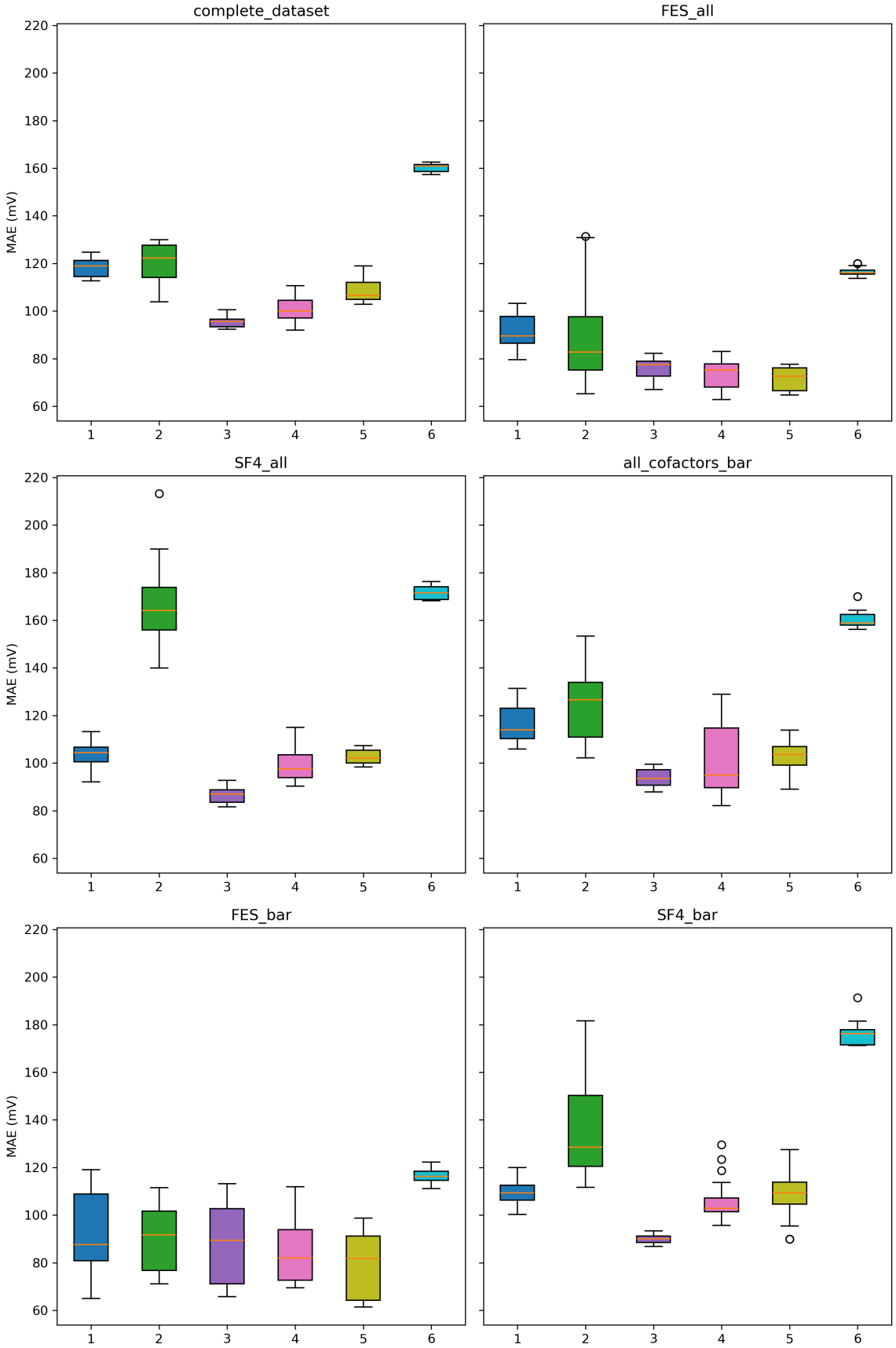

Figure 4.13: **Box plots of MAE distribution across radii for the top six models across six radius-dependent datasets**. XGB, in purple, consistently outperforms the other models, with lower mean MAE and lower spread, except in the FES_bar and FES_all datasets, where it is outperformed by RF (yellow).

4.4 Literature Comparison

The best MAE for each dataset is reported and compared with a literature MAE of (36 mV) [Galuzzi et al., 2022] in Table 4.8. It can be seen that the prediction on iron-sulfur proteins is consistently worse than that on flavoproteins, with the closest performance being 61.5mV, a difference of 25.1mV, corresponding to a degradation of 69% in performance.

Table 4.8: **Best MAE obtained on each dataset across all models**, benchmarked against the best MAE found for flavoproteins, of 36.4mV [Galuzzi et al., 2022]. A negative value for improvement shows degradation of performance.

Dataset (feature type)	Best MAE (mV)	Improvement (%)
All cofactors (all)	91.9	-152.6
FES (all)	62.9	-72.7
SF4 (all)	81.7	124.3
All cofactors (bar)	82.1	-125.5
FES (bar)	61.5	-68.9
SF4 (bar)	86.9	-138.7
All cofactors (protein)	93.53	-156.9
FES (protein)	77.98	-111.5
SF4 (protein)	81.6	-124.2

The relative error of the best performance on each cofactor type is reported and also compared with the literature MAE of (36 mV) [Galuzzi et al., 2022] in Table 4.9. According to this metric, the results obtained in this study were consistently better.

Table 4.9: **Redox potential ranges, best mean absolute error (MAE) in mV, and relative errors** by cofactor type and for the complete dataset, benchmarked against a literature MAE of 36.5 mV [Galuzzi et al., 2022]. Relative error is calculated as (MAE / Range) × 100, where Range is Max - Min. The results obtained in this study are consistently better than the literature.

Dataset	**Count**	**Min**	**Max**	**Range**	**MAE**	**Relative Error (%)**
Galuzzi	141	−400	73	473	36	8
All cofactors	168	−746	450	1196	82	7
[2Fe-2S]	60	−494	310	804	62	8
[4Fe-4S]	92	−746	450	1196	82	7
[3Fe-4S]	10	−429	65	494	–	–
Fe3+	6	−66	16	82	–	–

4.5 Overall Best Performances

4.5.1 Best Performing Model

The best-performing model was selected based on the lowest mean absolute error (MAE) across all datasets, consistent with Galuzzi et al.'s global MAE benchmark of 36 mV for flavoproteins [Galuzzi et al., 2022].

For radius-independent datasets, the overall MAE statistics, derived from `aggregate_radius_independent.py`, are presented in Table 4.4. XGB achieved the lowest mean MAE of 84 mV, with a median of 82 mV and standard deviation of 8 mV. Similarly, for radius-dependent datasets, the overall MAE statistics derived from `aggregate_radius_dependent.py` are presented in Table 4.6. It shows that XGB achieved the lowest mean MAE of 88 mV, with a median of 90 mV and standard deviation of 10 mV. This outperforms other models, including RandomForestRegressor, despite its dataset-specific optimum on FES_bar of 61 mV

This consistent performance across datasets justifies the presentation of the XGB heatmap (Figure 4.14) as the primary visualization, with additional model heatmaps detailed in Appendix C. From this heatmap, we can see that XGB achieves its best 3 performances on the FES_bar dataset, at 15Å(65.7 mV), then 16Å(66.3 mV), and then 13Å(68.4 mV).

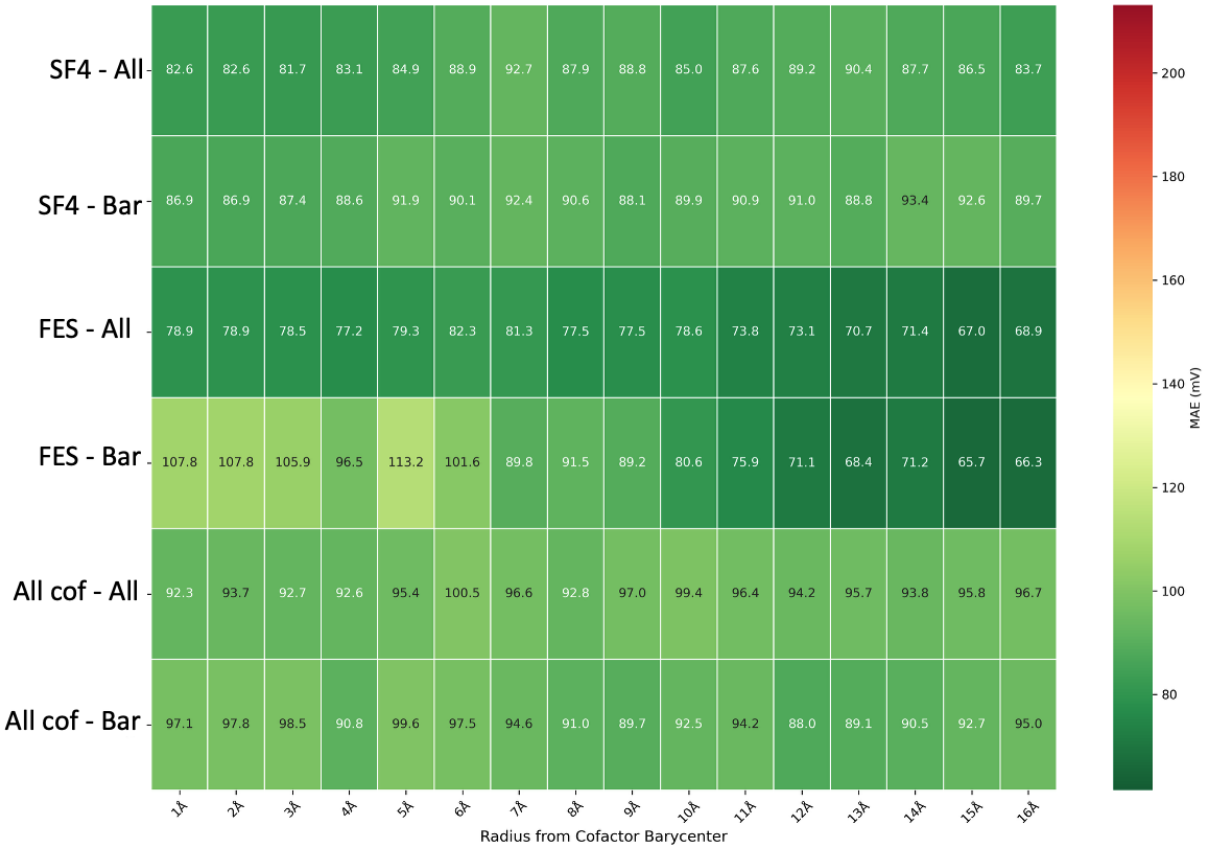

Figure 4.14: **XGB-specific heatmap for radius-dependent data** with datasets on the y-axis and radius in Å on the x-axis. All values shown are the MAE in mV. The best performances are almost all contained within the FES_all and FES_bar datasets, with the lowest MAE being 65.7mV for FES_bar at radius 15Å.

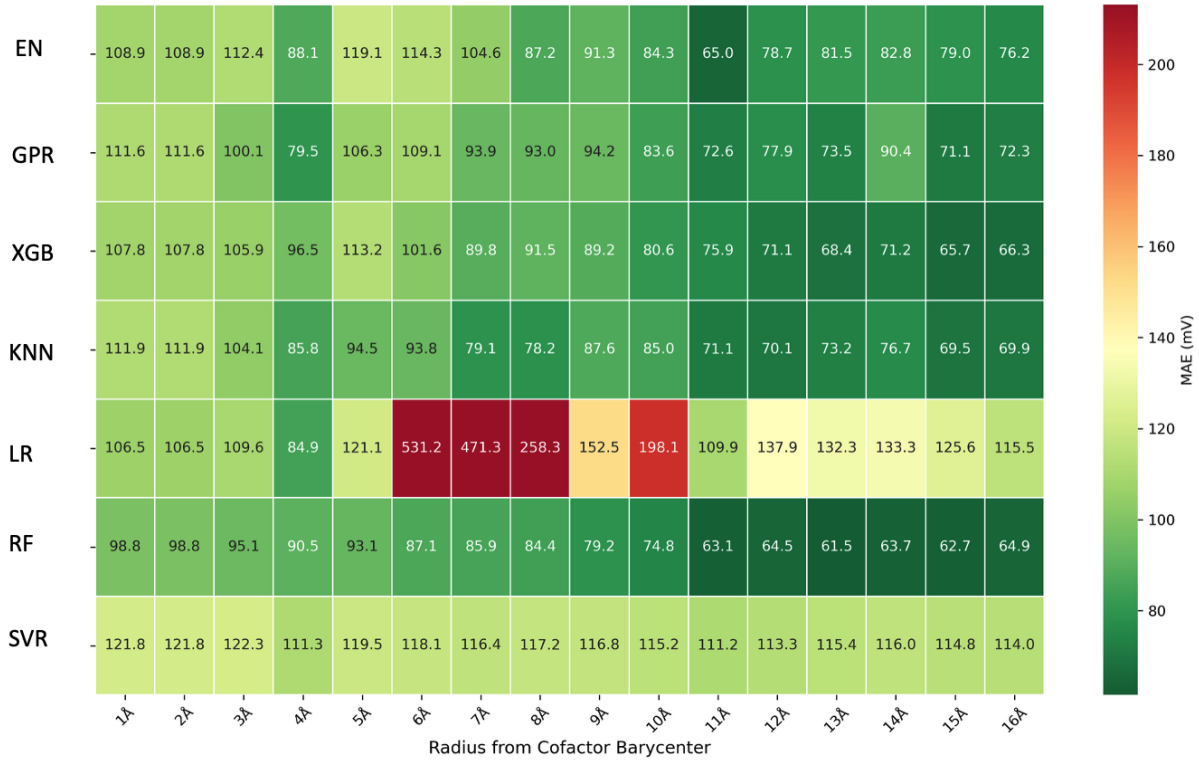

Figure 4.15: **Performance heatmap of all models on the FES_bar dataset**, with models on the y-axis and radius in Å on the x-axis. All values shown are the MAE in mV.

4.5.2 Best Performing Dataset

As seen on Tables 4.3 and 4.5, the best performances are almost always involving the FES dataset (5/5 top 5 for radius-dependent and 3/5 top 5 for radius-independent). The performance heatmaps of all models across FES_bar datasets and FES_all datasets are shown below in Figures 4.15 and 4.16, respectively.

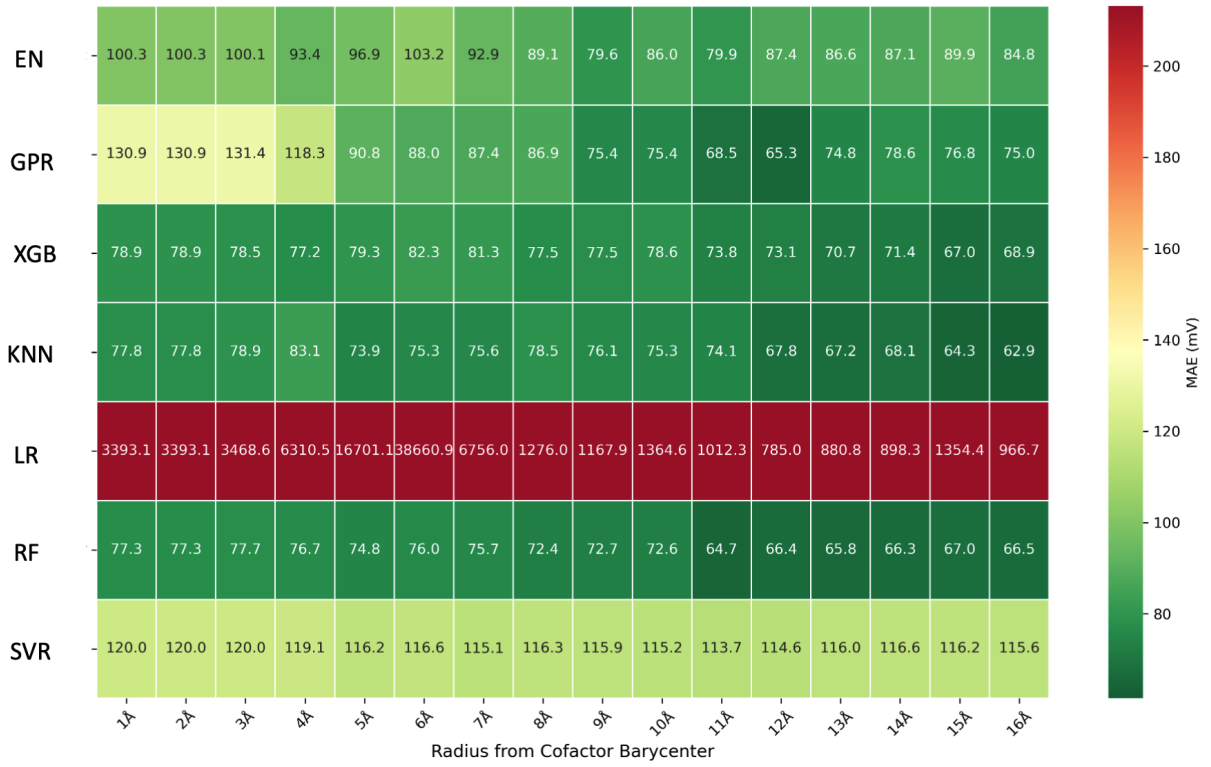

Figure 4.16: **Performance heatmap of all models on the FES_sll dataset**, with models on the y-axis and radius in Å on the x-axis. All values shown are the MAE in mV.

4.5.3 Statistical Significance of Model Performance

To assess the robustness of model performance differences, particularly given the non-negligible standard deviations in MAE (e.g., 8 mV for XGB on radius-independent datasets, Table 4.4), I conducted Mann-Whitney U tests [Krol et al., 2010]. These tests check if XGB performs significantly better on radius-independent and radius-dependent datasets at 15Å (separately) compared to the other models. I reject the null-hypothesis (that XGB is not significantly better) for a p-value < 0.05. We see from Table 4.10 that all values are below 0.05 for all radius-independent datasets and for radius-dependent datasets at radius 15Å (the optimal radius for FES_bar for GPR, XGB, and KNN), so we can more confidently **confirm that XGB performs significantly better** than the other models.

Comparison	Adj. p-value	Effect Size
Test 1: XGB on Radius-Independent Datasets		
vs. ElasticNet	0.0000	0.5712
vs. GaussianProcess	0.0002	0.4624
vs. KNeighbors	0.0006	0.4320
vs. LinearRegression	0.0000	0.8504
vs. RandomForest	0.0000	0.6704
vs. SVR	0.0000	0.6112
Test 2: XGB on Radius-Dependent Datasets (Radius 15)		
vs. ElasticNet	0.0000	0.3872
vs. GaussianProcess	0.0000	0.3997
vs. KNeighbors	0.0093	0.1208
vs. LinearRegression	0.0000	0.8661
vs. RandomForest	0.0000	0.1935
vs. SVR	0.0000	0.6975

Table 4.10: **Mann-Whitney U Test Results for XGB Performance against all other models on both radius-independent and radius-dependent (15Å) datasets**. We see that all values are below 0.05 for all radius-independent datasets and for radius-dependent datasets at radius 15Å (the optimal radius for FES_bar for GPR, XGB, and KNN), so we reject the null hypothesis and confirm that XGB performs significantly better than the other models.

4.5.4 Visualization Consistency

For all heatmaps except the stability heatmap, the same value range is used for the color mapping. While Linear Regression is present in the heatmaps, it was excluded from the color map range calculation to avoid skewing the scale too much.

4.6 SHAP Analyses

The SHAP violin plots display features in descending order of importance along the y-axis, and the distribution of the SHAP values along the x-axis, color-coded for their directional impact (red for higher feature values, and blue for lower feature values.

4.6.1 Feature Retention and SF4_bar SHAP plot

The SHAP analysis is complemented by an evaluation of feature retention across datasets and radii after feature selection (removing zero-variance and high-correlation features). The average number of features retained over all CV iterations (simulated by the script `preprocessing_analysis.py` across radii is shown in Figure 4.17. As expected, it can be seen that the bar features-only datasets have fewer features than the full datasets across

all radii. The percentage of features retained per radius is provided in the Appendix (Figure B.2).

It is worth noting that for the SF4_bar dataset, only 8 features were retained at radii 1 and 2 Å due to the limited number of amino acids within these small radii around the [4Fe-4S] cofactor (diameter 4 Å). This is visualized in the SHAP violin plot for XGB at radius 1 Å (Figure 4.18), which includes `struct.is_hipip`, `struct.burial_depth`, `pH`, `Bar.prop.Hydrophobicity`, `Bar.prop.Volume`, `Bar.prop.Log(Solubility (m at 20°C))`, `Bar.count.ASN`, and `Bar.count.MET`. Despite the reduced feature set, XGB at radii 1 and 2 Å achieved the best performance for SF4_bar (MAE: 81.7 mV), suggesting that these features capture the critical determinants of [4Fe-4S] redox potentials. Since the strong impact of the `struct.is_hipip` and `struct.burial_depth` features skews the scale of the plot, the absolute SHAP values for each of these 8 features are also provided in Table 4.11. We see that indeed, the other 5 features (not including pH) have negligible influence on the final redox potential prediction compared to the top 3 features.

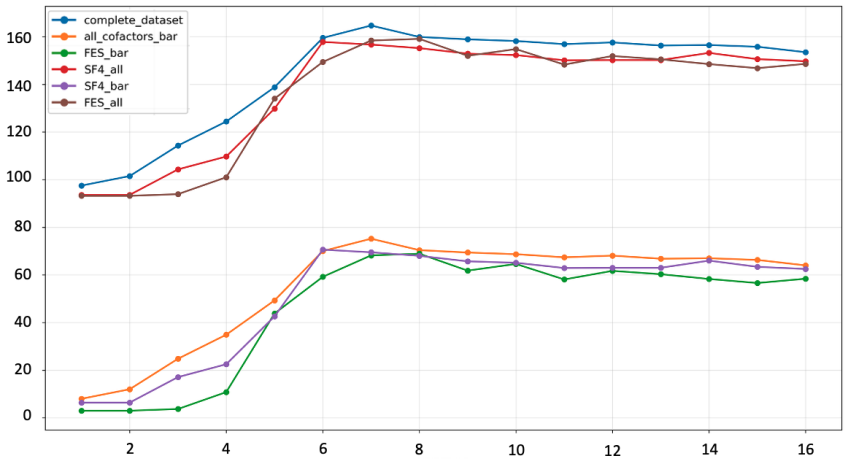

Figure 4.17: **Absolute count of features retained per radius after preprocessing**, for radius-dependent datasets. The y-axis shows the average absolute count of features retained, and the x-axis shows the radius in Angstrom. Bar features-only datasets consistently have less than half the features of the datasets with all features, with an especially low count at 4Å and below (between 2 and 40 features).

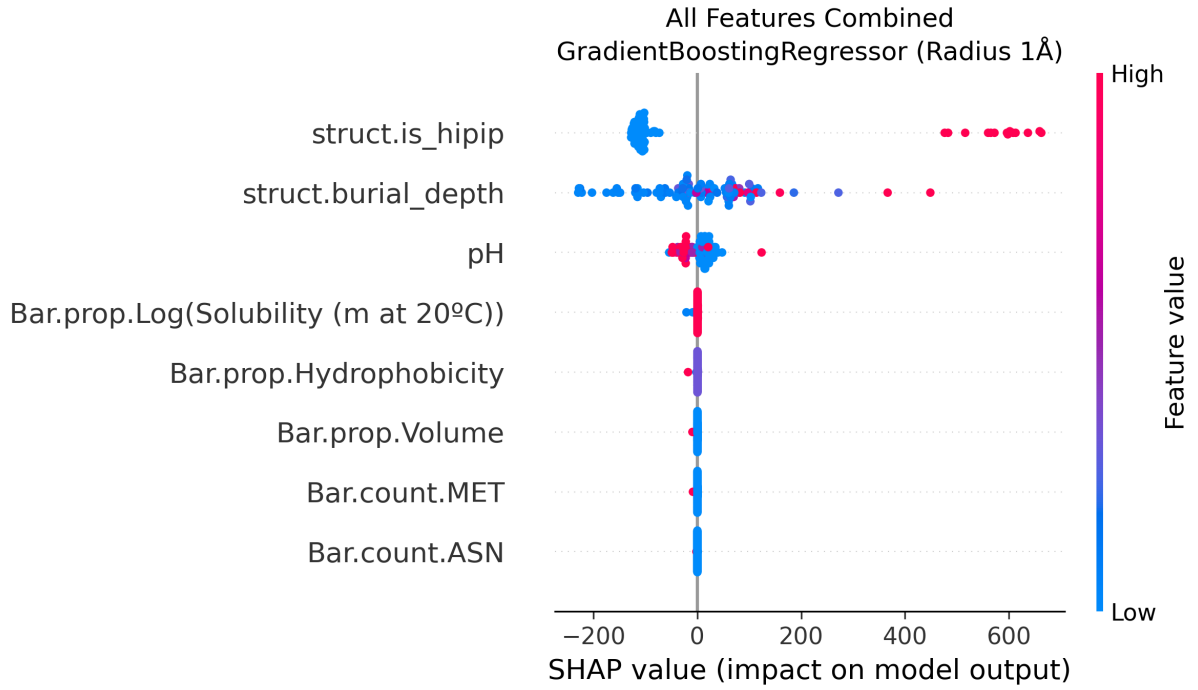

Figure 4.18: **SHAP violin plot of the top 8 features for the SF4_bar dataset at radius 1Å**, displayed in descending order of importance along the y-axis. Top performance was achieved by XGB. The HiPIP binary indicator was the more impactful feature, with high values (indicating a HiPIP) raising the predicted redox potential, and vice versa. A low pH also contributes to a higher redox potential prediction, and a high pH to a lower one. The other features show less contribution to the final redox potential prediction.

Table 4.11: **SHAP feature importance values for the XGB model on the SF4_bar dataset at radius 1Å**. Features `struct.is_hipip`, `struct.burial_depth`, and pH dominate, reflecting their critical role in predicting redox potentials (Section 4.6), while the negligible values of the other features show the restriction of the small spherical region considered around the cofactor barycenter.

Feature	**Importance**
`struct.burial_depth`	82.14
`struct.is_hipip`	181.87
`Bar.count.ASN`	0.06
`Bar.count.MET`	0.23
`Bar.prop.Volume`	0.32
`Bar.prop.Log(Solubility (m at 20°C))`	0.69
`Bar.prop.Hydrophobicity`	0.55
`pH`	21.56

4.6.2 Feature Importance SHAP Plot for the Complete Dataset

The SHAP violin plot for the top model for the complete dataset, containing all cofactors and all features, is shown in Figure 4.19. This plot is chosen as the primary SHAP visualization as it encompasses the entire dataset.

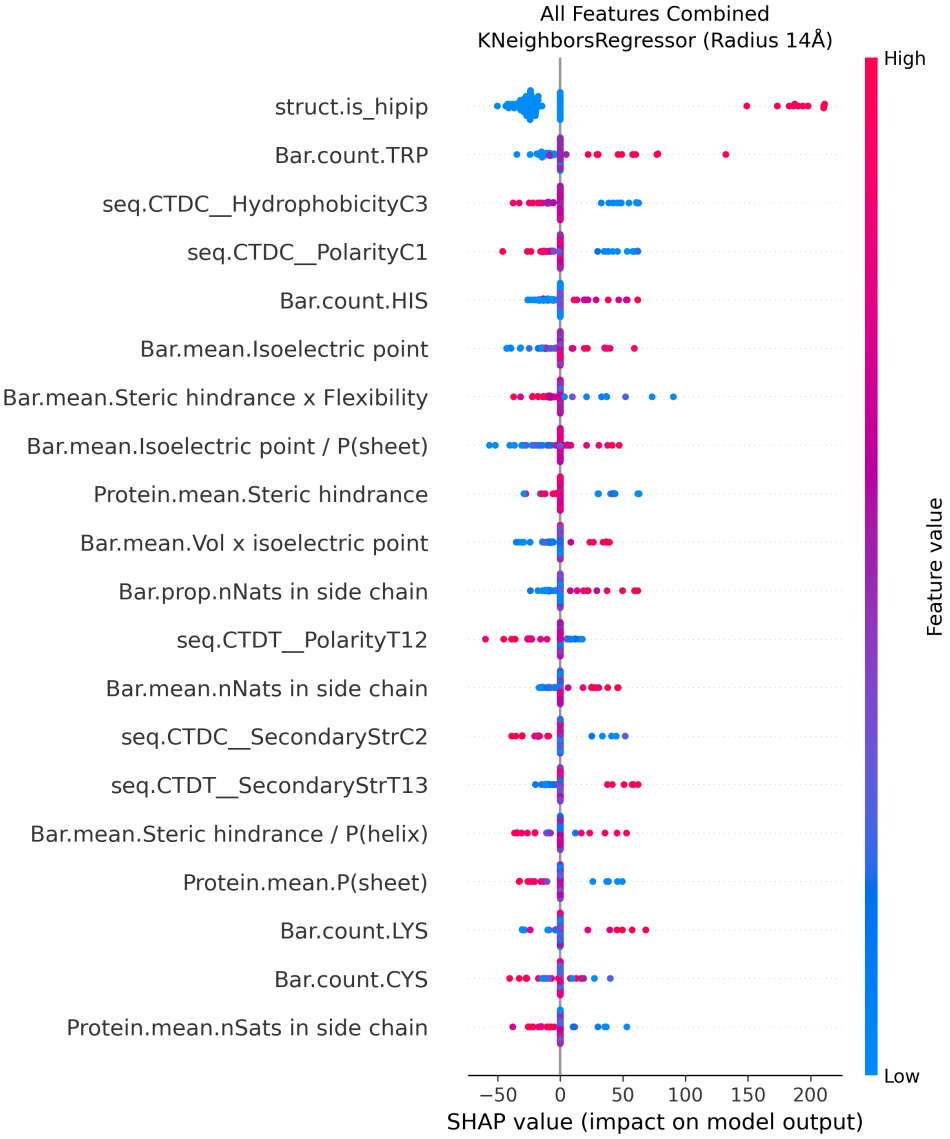

Figure 4.19: **SHAP violin plot of the top 20 features for the complete dataset**. The best model here was KNN for a radius of 14Å.

4.6.3 Important Recurrent Features Across All Datasets

The top features found by SHAP analysis for the top 3 models (or model-radius combination, for radius-dependent datasets) per cofactor subset are shown in Tables 4.12, 4.13, and 4.14. The recurrence criteria marks features in the top 5 of at least two SHAP plots within each subset with a checkmark.

Table 4.12: Recurrent Features Across Top 3 Model-Dataset Combinations for All Cofactors. The HiPIP indicator is the most important feature across all dataset types. Negatively charged (hydrophilic) amino acids, the number of nitrogen atoms in the side chains of residues, and isoelectric properties are also recurrent in some datasets.

Feature	**Description**	**All**	**Bar**	**Protein**
struct.is_hipip	Indicator for HiPIP structure	✓	✓	✓
seq.CTDC_PolarityC1	Proportion of amino acids in polarity group 1 (negatively charged)	✓		
Bar.prop.nNats in side chain	Number of nitrogen atoms in side chains of residues in the bar region		✓	
Protein.mean.Steric hindrance	Average steric hindrance of protein residues			✓
Protein.mean.Isoelectric point / P(sheet)	Average ratio of isoelectric point to sheet propensity across all protein residues			✓

Table 4.13: Recurrent Features Across Top 3 Model-Dataset Combinations for FES Cofactors. Helix propensity, isoelectric point, and the number of nitrogen atoms in protein residue side chains are the most important.

Feature	**Description**	**All**	**Bar**	**Protein**
Bar.mean.Isoelectric point	Average isoelectric point of residues within the bar region	✓		
Bar.mean.Vol / P(helix)	Ratio of volume to helix propensity of bar residues, averaged over residues in bar region		✓	
Bar.mean.P(helix) x Flex.	Average of the product of helix propensity and flexibility of the residues in the bar region		✓	
Protein.mean.nNats in side chain	Fraction of nitrogen atoms in protein residue side chains			✓

Table 4.14: Recurrent Features Across Top 3 Model-Dataset Combinations for SF4 Cofactors. Cofactor burial depth and the HiPIP indicator are the most recurrent features, hydrophobicity is recurrent in two datasets, and structural properties like sheet and helix propensity are also very important.

Feature	**Description**	**All**	**Bar**	**Protein**
seq.CTDC_ HydrophobicityC3	Fraction of amino acids in hydrophobicity group 3 (hydrophobic)	✓		
struct.is_hipip	Indicator for HiPIP protein	✓	✓	✓
struct.burial_depth	Cofactor burial depth	✓	✓	
Protein.mean.P(sheet)	Average sheet propensity of protein residues	✓		
Protein.mean.P(helix) x Flex.	Product of helix propensity and flexibility of protein residues	✓		
pH	Acidity level of the environment		✓	
Bar.prop.Hydrophobicity	Total hydrophobicity of the residues in the bar region		✓	

4.7 Summary of Results

The iron-sulfur protein dataset comprised 168 redox potential entries from 130 unique proteins, sourced from 113 scientific papers, covering four cofactor types: [4Fe-4S] (SF4), [3Fe-4S], [2Fe-2S] (FES), and Fe3+. Approximately one-third of entries were mutants, with protein sizes ranging widely (median residue count: 100, median diameter: 40Å) and burial depths varying by cofactor type (e.g., SF4 median: 8Å). Redox potentials spanned a broad range (-600 mV to +400 mV), with distinct distributions for each cofactor, and pH values typically ranged from 6 to 8. Cofactor implantation was validated with PyMOL's RMSD (median: 1Å), though one outlier (P73276, RMSD 14Å) highlighted limitations in AlphaFold structures for certain subunits.

Machine learning models were trained on nine dataset subsets: all cofactors, FES, and SF4, each with all features, bar features (radius-dependent), and protein features (radius-independent). For radius-independent datasets, Gradient Boosting Regressor (XGB) achieved the lowest mean MAE of 84.09 mV (SD: 8.47 mV), outperforming other models like KNN (97.65 mV) and RandomForest (RF). For radius-dependent datasets, XGB again led with a mean MAE of 88.17 mV (SD: 9.60 mV), but RF achieved the best single performance (MAE: 61.49 mV) on the FES_bar dataset at an optimal radius of 11Å. No consistent optimal radius emerged across datasets, though FES datasets favored larger radii (9–16Å). Model stability analysis showed XGB and RF as the most reliable, with low MAE standard deviations. Compared to Galuzzi et al.'s flavoprotein benchmark (MAE: 36.4 mV), the best iron-sulfur prediction (61.49 mV) was 69% worse, indicating greater complexity in modeling iron-sulfur proteins.

SHAP analysis revealed key features driving model predictions. For the complete dataset (all cofactors, all features), the top model (KNN, radius 14Å) highlighted features like hydrophobicity and structural properties. Recurrent features across datasets included `struct.is_hipip` (all SF4 subsets), `seq.CTDC__HydrophobicityC3` (SF4 all features, protein features), and Protein.mean.P(sheet) (SF4 all features, protein features), suggesting the importance of cofactor-specific structural and physicochemical properties. Visualizations, including heatmaps and performance trend plots, consistently highlighted XGB and RF's robustness, with FES datasets generally outperforming SF4 and complete datasets.

5

Discussions

5.1 Introduction

This study developed a machine learning framework to predict redox potentials of iron-sulfur proteins, using a dataset of 168 entries from 130 proteins across four cofactor types ([4Fe-4S], [3Fe-4S], [2Fe-2S], Fe3+), as detailed in Chapter 3. The results (Chapter 4) reveal a diverse dataset with significant structural and physicochemical variability, with the most robust ML performance led by Gradient Boosting Regressor (overall mean MAE: 84.09 on radius-independent data–88.17 mV on radius-dependent) and RandomForestRegressor (best MAE: 61.49 mV on FES_bar), and key features identified through SHAP analysis, such as `struct.is_hipip` and `seq.CTDC__HydrophobicityC3`. This chapter interprets these findings, evaluates model effectiveness, compares performance to the flavoprotein benchmark of Galuzzi et al. [Galuzzi et al., 2022], discusses limitations, and proposes future directions to enhance redox potential prediction for iron-sulfur proteins.

5.2 Dataset Characteristics and Implications

Dataset diversity The final dataset is extremely diverse, encompassing 168 redox potential entries from 130 unique proteins, and 6 different cofactor types (counting HiPIP and Rieske-type cofactors separately), reflecting the structural and functional complexity of iron-sulfur proteins. Approximately one-third of entries were mutants (Figure 4.3), which were included to hopefully enable models to capture subtle structural changes, as intended (Section E.1). Protein sizes (median: 100 residues, 40Å diameter; Figure 4.4) and burial depths (Figure 4.8) varied significantly. The redox potentials in the whole dataset (-746 mV to +450 mV; Figure 4.7) also span a larger range than that of flavoproteins, whose redox potentials for the FMN and FAD cofactors in Galuzzi et al.'s dataset ranged from -399.5 mV to 73 mV [Galuzzi et al., 2022]. The use of AlphaFold structures, chosen for availability and consistency, expanded the dataset beyond the 141-entry target inspired by Galuzzi et al. [Galuzzi et al., 2022], but introduced challenges, such as

needing to minimize mutant structures and implant cofactors, leading to structures that are likely inaccurate, such as the outlier entry P73276 (RMSD 14Å; Table 4.2), where a missing binding site likely skewed feature extraction due to incomplete AlphaFold modeling of a subunit. Furthermore, the inherent variability in experimental conditions, most notably in pH value, likely introduced noise, as many proteins have pH-dependent redox potentials [Krishtalik, 2003].

Cofactor implantation Cofactor implantation via AlphaFill and manual transplants (Section E.4) achieved a median RMSD of 0.1Å, ensuring reliable structural integration for most entries. However, complex cases like P0DUV7 (homotrimer) and P18187_P238C ([3Fe-4S] to [4Fe-4S] conversion) required tailored manual approaches, which are less reliable, highlighting the need for robust implantation methods.

5.3 Model Performance and Visualization Insights

XGB and RF as the best performers The ML models's performance was compared across nine dataset subsets (all cofactors, FES, SF4–and for each, all features, bar features, protein features). **XGB** achieved the lowest mean MAE for radius-independent (84.09 mV, Table 4.4) and radius-dependent (88.17 mV, Table 4.6) datasets, reflecting its ability to capture non-linear relationships in complex physicochemical data. Its ensemble approach, iteratively building decision trees, and built-in feature selection (Section E.6.1) minimized overfitting, ensuring stability across diverse datasets, including FES and SF4 subsets. **RF** followed closely, with a mean MAE of 99.36 mV (Standard Deviation (SD): 20.94 mV) for radius-independent and 95.54 mV (SD: 16.73 mV) for radius-dependent datasets, achieving the best single performance on FES_bar (61.49 mV at 11Å; Table 4.5). RF's robustness to irrelevant features suited the high-dimensional dataset (199 features), and its success on FES_bar suggests sensitivity to local microenvironmental effects for [2Fe-2S] clusters, and is likely due to the simpler coordination chemistry of these clusters. Heatmaps (Figures 4.9, 4.15) and stability analyses (Figure 4.10) confirmed XGB and RF as the most reliable models, with low MAE standard deviations, indicating robustness to changes in input data's spatial context.

Models in between KNN also performed relatively well, with a mean MAE of 97.65 mV (SD: 17.25 mV) for radius-independent and 94.34 mV (SD: 15.32 mV) for radius-dependent datasets. KNN directly uses feature distances for predictions, making it naturally sensitive to the addition of local spatial features, which can be seen from its downward trend in MAE vs radius on feature-dependent datasets (Figure 4.12. While KNN captures local patterns, it still likely struggles with global non-linearities, contributing to its higher MAE. EN showed moderate performance (104.12 mV, SD: 9.34 mV for radius-independent; 105.29 mV, SD: 13.62 mV for radius-dependent), limited by its linear assumptions, which fails to capture complex patterns in protein stucture-property relationships; however, the fact that its performance comes relatively close to that of the other superior models suggests that its hyperparameter tuning was mostly succesful. Due to its limitations (linear assumptions), it is unclear how much better its performance could get on such a complex task.

Underperformers SVR and GPR underperformed, with mean MAEs of 115.31 mV (SD: 9.89 mV) and 155.66 mV (SD: 46.24 mV) for radius-independent, and 150.28 mV (SD: 24.82 mV) and 121.49 mV (SD: 31.84 mV) for radius-dependent datasets, respectively. Although SVR, with an `rbf` kernel, should be able to model nonlinear relationships, its relatively high MAE suggests that either its hyperparameters were not tuned well enough, or the complex structure-property relationships in iron-sulfur proteins are too complex to capture. Furthermore, SVR consistently showed high and radius-invariant MAE, suggesting that it failed to incorporate radius-dependent features effectively into its predictions. This could stem from its reliance on global feature scaling and/or simply its inability to link structural properties to the redox potentials in the diverse and complex dataset of iron-sulfur proteins. Conversely, GPR exhibited large fluctuations in MAE across radii. While the overall trend is a lower MAE with increasing radius, indicating that it utilized spatial context, its performance instability suggests overfitting or that the chosen kernel did not adequately generalize across varying local environments. Indeed, the covariance kernel used (Table E.3) assumes data smoothness, which the heterogeneous iron-sulfur dataset violates. Both SVR and GPR are highly sensitive to noise and feature scaling, so the dataset's high dimensionality and variability likely exacerbate errors.

Worst model LR was by far the worst model, with extreme mean MAEs, a whole order of magnitude higher than that of the other models (1232.51 mV, SD: 1871.13 mV for radius-independent; 1168.37 mV, SD: 4342.74 mV for radius-dependent). However, for the 2 datasets, all cofactors and SF4 cofactors with only protein features, its performance of 140.3 mV and 164.2 mV in MAE, respectively (Figure 4.9) is comparable with that of SVR and GPR, which puts into perspective the under. LR's poor performance shows that a simple linear model is not enough to represent the complex relationship between protein structures and their cofactors' redox potentials. It had to be excluded from color scales in heatmaps (Section E.8) to improve visualization clarity, as its high MAE (>300 mV) skewed interpretations.

Feature scaling and feature selection. It is worth noting that the two best models (XGB and RF) are the only ones insensitive to feature scaling and with internal feature selection (Section E.6). XGB and RF are tree-based models that rely on decision trees, which split data based on feature value thresholds rather than their absolute magnitudes. This makes them insensitive to feature scaling (e.g., standardization or normalization), unlike models like KNN, SVR, EN, GPR, and LR, which depend on distance metrics or linear relationships sensitive to feature scales. This suggests that some features may lose informative value after scaling, and/or that my feature selection method is not optimal. For example, scaling features to a uniform range (e.g., standardization to mean 0, variance 1) can reduce the relative importance of features with naturally larger ranges (e.g., burial depth in Ångstroms vs. binary features like `struct.is_hipip`). XGB and RF's insensitivity to scaling preserves these variations, potentially retaining critical structural and chemical information about cofactor environments. My results (superior performance of tree-based XGB and RF, underperformance of non tree-based scaling-sensitive models) strongly suggest that unscaled features better capture the dataset's physicochemical complexity. Scale-sensitive models like SVR and GPR, with high MAE and variability (e.g., GPR average SD: 46.24 mV for radius-independent), likely suffered from scaling-related information loss. The external feature selection (top 50 features, Section E.6.1) was rigorous but applied uniformly across all non-tree-based models, which may not have been

optimal for them, as it didn't account for their specific sensitivities to feature distributions or noise. XGB and RF's built-in feature selection dynamically prioritizes relevant features (e.g., `struct.is_hipip`, Table 4.12), likely better adapting to dataset heterogeneity (e.g., SF4 vs. FES subsets). It is also possible that the feature selection process may have excluded some relevant features or included noisy ones, particularly for smaller subsets (e.g., FES with fewer entries). The dataset's modest size (168 entries) and variability (e.g., inaccurate cofactor implants) could amplify the impact of suboptimal feature selection for scale-sensitive models. Larger, cleaner datasets might reduce the performance gap for scale-sensitive models.

Optimal radius and dataset The lack of a clear optimal radius (Table 4.7) suggests that the microenvironment's influence on redox potential varies widely by cofactor type. FES datasets favored larger radii (9–16Å), possibly capturing broader electrostatic interactions, while SF4 datasets showed more variability, reflecting the structural diversity of [4Fe-4S] proteins (e.g., HiPIPs). Performance trends (Figure 4.12) showed improved MAE for FES_all at larger radii, but no universal optimal radius emerged, which complicates feature extraction strategies. However, from these performance trends, the XGB-specific heatmap (Figure 4.14) and the RF-specific heatmap (Figure C.5), it seems like the downwards trend in MAE with increasing radius in the FES_bar and FES_all datasets might have continued had there been data for larger radii, which is something that can be tested in the future. Heatmaps (Figures 4.15, 4.16) highlighted FES datasets' superior performance, likely due to simpler [2Fe-2S] chemistry.

Benchmarking against Galuzzi et al. Compared to Galuzzi et al.'s flavoprotein benchmark (MAE: 36.4 mV), the best iron-sulfur performance for all cofactors (bar features only, 91.9 mV) was 152% worse, and the FES subset (bar features only, 61.49 mV) was 69% worse, with the SF4 subset (protein features only, 81.6 mV) also 124% worse (Table 4.8). This gap may stem from the smaller dataset size, the structural complexity of iron-sulfur clusters, and experimental variability. However, relative errors reveal comparable performance, with relative error across all datasets being consistently lower than that of Galuzzi et al.'s. The best relative error achieved here, of 6.83% for the SF4 datasets, is 0.87 percentile points lower than the literature value of 7.70% [Galuzzi et al., 2022], and the worst relative error on the FES cofactor dataset, of 7.65%, is 0.05 percentile points lower (Table 4.9.

This indicates that, even with larger absolute errors, the models achieve similar accuracy, despite this study's dataset's range being 2.53 and 1.7 times larger than Galuzzi et al.'s for all cofactors/SF4 cofactors and FES cofactors, respectively (1196mV vs. 472.5 mV and 804mV vs. 472.5mV) and the entries being more diverse (4 cofactor types vs. 2 in flavoproteins).

5.4 Feature Importance Insights

The complete dataset's SHAP violin plot (Figure 4.19) highlighted a mix of local (`Bar.`) and global (`Protein.`,`seq.`, `struct.`) features, suggesting that both microenvironmental and protein-wide properties contribute to redox potential prediction. The `generate_shap_table.py` script's configurable recurrence criteria (top 5 features, at least 2 combinations) ensured robust feature identification, but deeper biological interpretation is needed

to elucidate mechanistic roles, such as how hydrophobicity influences electron transfer pathways.

SHAP analysis (Section E.8.6) identified recurrent features driving redox potential predictions across all cofactors, FES, and SF4 datasets, as summarized in Tables 4.12, 4.13, and 4.14. These features, detailed in Appendix B.4, reveal the interplay of hydrophobicity, electrostatic, and structural properties in modulating iron-sulfur protein electrochemistry.

Hydrophobicity-related features Hydrophobicity-related features, such as `seq.CTDC__HydrophobicityC3` (SF4_all) and `Bar.prop.Hydrophobicity` (SF4_bar), highlight the role of non-polar environments in stabilizing [4Fe-4S] clusters, reducing solvent exposure and increasing redox potentials, particularly in High Potential Iron-Sulfur Proteins (HiPIPs). The feature `struct.is_hipip` (all cofactors and SF4; all, bar, protein subsets) is universally recurrent, reflecting its critical role in identifying high redox potentials, as HiPIPs have hydrophobic pockets that stabilize the oxidized $[4Fe\text{-}4S]^{3+}$ state, elevating redox potentials (+50 to +450 mV) [Moreno et al., 1994]. In the SF4_bar dataset at radius 1 Å (Figure 4.18), `Bar.prop.Hydrophobicity` and related features (`Bar.prop.Volume`, `Bar.prop.Log(Solubility (m at 20°C))`) contribute to the high performance, further reinforcing that the immediate hydrophobic environment is a primary determinant of [4Fe-4S] redox potentials.

Electrostatic properties Electrostatic properties, including `pH` (SF4, bar), `Bar.mean.Isoelectric point` (FES, all features), and `seq.CTDC__PolarityC1` (all cofactors, all features), modulate the local charge environment. The pH is particularly significant for SF4_bar, where high pH values ($\geq$7.6, 30/38 entries in SF4; Table 4.1) are associated with a lower redox potential (Figure 4.18). This is because the alkaline environment can deprotonate residues near the [4Fe-4S] cluster, lowering redox potentials by reducing positive charge in the cofactor's environment [Moreno et al., 1994]. However, the predominance of high pH values in SF4 entries may amplify `pH`'s importance, suggesting a dataset bias that warrants further investigation. For FES datasets, `Bar.mean.Isoelectric point` influences [2Fe-2S] redox stability through local charge distribution, while `seq.CTDC__PolarityC1` reflects the role of negatively charged amino acids in modulating the dielectric environment across all cofactors.

Structural properties Structural features, such as `struct.burial_depth` (SF4, all and bar) and `Protein.mean.P(sheet)` (SF4_all), reflect cofactor accessibility and protein rigidity. Deeper burial reduces solvent access, increasing [4Fe-4S] redox potentials by stabilizing the reduced state, and beta sheet propensity enhances the protein's low-dielectric environment, stabilizing the cofactor's redox state [Stephens et al., 1996].
The high performance of XGB on SF4_bar at radii 1 and 2 Å with only 8 features (Figure 4.18) suggests that these structural and microenvironmental features, including `struct.burial_depth` and `Bar.count.ASN`, capture the primary drivers of [4Fe-4S] redox potentials, with additional features at larger radii potentially introducing noise. It can be reasoned that the large structural difference between [4Fe-4S] clusters in ferredoxins versus in HiPIPs (and by proxy, the large difference in their redox potentials), means the cofactor's immediate microenvironment is that much more important, leaving features further away with just weaker or indirect effects, effectively only becoming noise to the ML models. Furthermore, as most [4Fe-4S] proteins contain multiple cofactors, the growing bar radii might also catch those other cofactors' environments, potentially confusing

the model even more.
Surprisingly, `struct.is_rieske` did not appear as a recurrent feature in either the FES or all_cofactors datasets, as it appeared in the SHAP plots of only 2 of the 3 top models, thus missing the recurrence criteria. This may indicate that other features were consistently more predictive, or that this feature was not well picked up on due to the low representation of Rieske proteins in the dataset (Figure 4.2).

For FES_bar, `Bar.mean.Vol / P(helix)` indicates that residue packing density in the bar region affects [2Fe-2S] cofactor stability, influencing electron transfer. `Protein.mean.nNats in side chain`, the fraction of Nitrogen atoms in the side chains of the residues considered, is also found to be important in both FES_protein, with higher values raising redox potential (Figure 4.19). Amino acids with high numbers of N atoms are lysine, arginine, histidine, and tryptophan, the former 3 of which can all be protonated, which influences the molecular neighborhood charge [Galuzzi et al., 2022].

These findings highlight that the immediate cofactor microenvironment, particularly for [4Fe-4S] clusters, dominates redox potential prediction, with hydrophobicity, pH, and burial depth being key determinants, though pH's prominence in SF4 may reflect dataset bias (Table 4.1). For [2Fe-2S] clusters, both local (bar) and global (protein) properties drive their electrochemical behavior.

Non-Zero Amino Acid Features at 1Å Radius The SHAP analysis for XGB on the SF4_bar dataset at a 1Å radius (Table 4.11) revealed non-zero importance for amino acid-related features `Bar.count.ASN` (0.06), `Bar.count.MET` (0.23), `Bar.prop.Hydrophobicity` (0.55), and `Bar.prop.Volume` (0.32), despite the typical amino acid side chain size of 3 Å. Due to steric constraints and van der Waals radii (∼1 Å–1 Å), and a typical Fe-S bond (betweeen irons in the cofactor and sulfurs in the coordinating cysteines) measuring around 2.2Å [Terranova, 2024], non-coordinating residues like asparagine or methionine are unlikely to have atoms within 1 Å of the cofactor's center. This suggests that cofactor implantation inaccuracies, particularly from manual transplants (Section E.4), may have shifted the cofactor's center, erroneously including residue atoms in the 1Å radius.

5.5 Limitations

The study faced several limitations, every step of the way, from data acquisition to model training.

Dataset limitations The dataset's diversity (168 entries, four cofactor types) limited model generalization compared to Galuzzi et al.'s flavoprotein study (141 entries, two cofactor types) [Galuzzi et al., 2022]. This diversity, while valuable for capturing variability, may constrain model generalization to underrepresented cofactors, particularly [3Fe-4S] (10 entries) and Fe3+ (6 entries). The skewed pH distribution (Figure 4.6), with 30 of 38 high pH values (≥7.6) corresponding to SF4 cofactors, likely amplified `pH`'s importance in SF4_bar predictions (Table 4.14), reflecting potential dataset bias rather than universal electrochemical effects. Data overrepresentation was a concern, as Clostridial-type ferredoxins with isopotential [2Fe-2S] or [4Fe-4S] clusters [Arcinas et al., 2019], [Prince and Adams, 2018] were counted as separate entries despite their likely similar cofactor environments, potentially skewing microenvironment representation. Similarly, 20 of 50 mutant entries from a single protein's saturation mutagenesis study (replacement of an

amino acid at one position with all other possible amino acids) [Birrell et al., 2016] may have overemphasized specific structural changes. Additionally, reliance on older sources, such as [Meyer et al., 1983], which provided a dozen HiPIP [4Fe-4S] redox potentials from 1983 and earlier, may introduce inconsistencies due to outdated experimental methods or conditions not aligned with modern standards.

Structural Modeling Limitations The use of AlphaFold structures (Section E.1) enabled a larger dataset but introduced inaccuracies, as AlphaFold omits small-molecule coordinates, necessitating AlphaFill and manual cofactor transplants (Section E.4). AlphaFill's failure to enrich over half the structures required many subjective manual transplants, introducing variability, particularly for complex cases like P0DUV7 (homotrimer) and P18187_P238C ([3Fe-4S] to [4Fe-4S] conversion). The outlier P73276 (alignment RMSD 14Å, Table 4.2) highlighted AlphaFold's limitations for subunits with incomplete binding sites. Energy minimization of mutant structures (Section E.2.2), performed before cofactor implantation due to the lack of AMBER force fields for iron-sulfur clusters, likely reduced cofactor placement accuracy, affecting features like `struct.burial_depth`. With one-third of entries being mutants and no crystal structures for validation, these structural uncertainties may have impacted feature extraction and prediction reliability.

Model and Feature Extraction Limitations The feature extraction process (Section E.5), while comprehensive with 199 physicochemical and structural features, relied on static AlphaFold structures, potentially missing dynamic interactions (e.g., conformational changes) critical to redox potential. It also misses most of the arrangement of the protein in 3D space. The `struct.burial_depth` computation, using convex hull methods, is inaccurate in incomplete or uncertain AlphaFold predictions (e.g., P73276). Model training faced challenges due to inherent model limitations: LinearRegression's linear assumptions, GPR's smoothness assumptions, SVR's noise sensitivity, and more, led to sub-par performance (Tables 4.4, 4.6). Feature scaling, combining distance-based (e.g., `struct.burial_depth`) and binary features (e.g., `struct.is_hipip`), likely skewed the importance of binary features. Moreover, the feature selection process, limited to $\min(50, n/2)$ features, may have excluded relevant features in low-feature datasets (e.g., SF4_bar at small radii), limiting generalization.

Computational restraints The computational cost of nested cross-validation (21 hours for radius-dependent datasets, Section E.9) and supercomputer queueing delays restricted hyperparameter exploration (e.g., SVR's `poly` kernel) and testing of larger radii, limiting scalability. These constraints highlight the trade-offs between computational efficiency and model optimization.

These limitations highlight the challenges of working with a diverse, computationally generated dataset and the trade-offs in model training, suggesting areas for improvement in data collection, structural modeling, and computational efficiency to enhance redox potential prediction accuracy.

5.6 Future Directions

This future work section attempts to address the identified limitations to enhance redox potential prediction accuracy and applicability in iron-sulfur proteins, and perhaps, other

protein families, too.

Dataset expansion Expanding the dataset with additional redox potential measurements, particularly for [3Fe-4S] and Fe3+ cofactors, through targeted literature searches or experimental validation, could improve model generalization. Trying to implement more crystal structures in the dataset instead of only AlphaFold predictions would reduce reliance on computational modeling and might mitigate biases, leading to more accurate predictions.

Feature transformation for binary indicators To mitigate the impact of feature scaling on scale-sensitive models like SVR, GPR, and KNN, future work could explore transforming binary features, such as `struct.is_hipip` and `struct.is_rieske`, into continuous or categorical representations. These binary indicators, critical for identifying high-potential [4Fe-4S] clusters (e.g., HiPIPs), may lose influence during standardization, skewing predictions for non-tree-based models. For instance, replacing `struct.is_hipip` with a continuous proxy like cofactor pocket hydrophobicity could preserve its predictive power while ensuring compatibility with scaling, potentially improving performance for models sensitive to feature distributions. This approach would complement the robustness of tree-based models (XGB, RF) and enhance generalization across diverse Fe-S protein datasets.

Enhanced feature extraction Feature extraction could be enhanced by including dynamic properties via molecular dynamics simulations (e.g., conformational flexibility) or quantum mechanical descriptors for iron-sulfur clusters (e.g., Fe-S bond lengths, cluster oxidation states) to capture electronic effects. Adapting Galuzzi et al.'s approach [Galuzzi et al., 2022] to extract physicochemical properties around key cluster atoms (e.g., sulfur atoms involved in redox changes) could enhance feature relevance.

Improved cofactor implantation Molecular dynamics simulations, using tailored AMBER or GROMACS force fields for iron-sulfur clusters, could refine AlphaFill and manual transplants, improving cofactor placement accuracy and reducing variability in features like `struct.burial_depth`. Developing such force fields would require addressing current limitations in modeling complex cluster interactions.

Advanced model development Fine-tuning hyperparameters for XGB and RF (e.g., learning rate, tree depth) and exploring unscaled features for KNN/SVR could improve performance. Tailoring feature selection for non-tree-based models could prevent the exclusion of relevant features in low-feature datasets. Exploring advanced ML models, such as deep learning or ensemble methods, may improve accuracy; however, a larger dataset is needed before Neural Networks can be implemented. One interesting approach presented by Blaabjerg et al. combines a graph representation of protein structure with the sequence data to predict variant effects [Blaabjerg et al., 2024]. One study by Jia et al. has already tried applying graph-based machine learning methods to redox potential prediction, showing promising results, although with lower accuracy than in this study [Jia et al., 2024].

Feature validation and applications Validating key SHAP features with experimental mutagenesis could confirm their roles in determining redox potential. Finally, applying these models to protein engineering, such as designing iron-sulfur proteins with tailored redox potentials for synthetic biology or bioenergetics, could extend the study's practical impact, leveraging insights into critical features like hydrophobicity and secondary structure.

These distances were chosen for thoroughness–the Fe-S cofactors having a diameter of around 4Å, and residues beyond 12Å being unlikely to affect the cluster's redox potential [Terranova, 2024], this range of radii would encompass, with a safe margin, all possible relevant interactions.

6

Conclusions

This study developed and evaluated machine learning models to predict redox potentials of iron-sulfur proteins based on structural and physicochemical features, using a diverse dataset of 168 entries from 130 proteins across four cofactor types ([4Fe-4S], [3Fe-4S], [2Fe-2S], Fe3+)—the largest systematic analysis of its kind. Compared to Galuzzi et al.'s 141-entry flavoprotein dataset with two cofactor types [Galuzzi et al., 2022], this study tackled greater structural and electrochemical complexity, though the minimal size increase underscores the need for further data expansion. A diverse dataset of 168 entries from 130 proteins across four cofactor types was compiled and put through a robust machine learning training process to assess the possibility of accurate redox potential prediction in iron-sulfur proteins, given the scarcity of experimental measures for both their structures and redox potentials.

Key findings include the superior performance of tree-based models, particularly Gradient Boosting (XGB) and Random Forest (RF). XGB achieved the lowest mean absolute error (MAE) across datasets (84.09 mV for radius-independent and 88.17 mV for radius-dependent data), while RF reached the best single performance of 61.49 mV MAE on the [2Fe-2S] dataset with bar features. Mann-Whitney U tests confirmed XGB's significant outperformance ($p < 0.05$) across datasets, though RF's competitive MAE on FES datasets suggests it may excel in specific contexts, particularly for [2Fe-2S] clusters with simpler coordination chemistry. Furthermore, XGB and RF's low MAE standard deviations (e.g., 8.47 mV for XGB on radius-independent data) underscored their stability, making them reliable for diverse iron-sulfur protein datasets. Despite promising results, the best MAE (61.49 mV for FES_bar) was 69% higher than Galuzzi et al.'s flavoprotein benchmark (36.4 mV), reflecting the greater structural complexity of iron-sulfur proteins (four cofactor types vs. two) and challenges like AlphaFold inaccuracies and pH distribution biases. While relative errors (6.83–7.65%) are slightly lower than Galuzzi et al.'s 7.70%, the broader redox potential range (up to 1196 mV vs. 472.5 mV) underscores the models' ability to handle greater variability and their robustness and potential for improvement. The superior performance on FES (61.49 mV) and SF4 (81.6 mV) datasets

compared to all cofactors (91.9 mV) likely stems from the underrepresentation of [3Fe-4S] (10 entries) and Fe3+ (6 entries) cofactors, which increased heterogeneity and reduced model generalization for the complete dataset. Addressing dataset and computational limitations, incorporating dynamic features, and exploring practical applications could further enhance prediction accuracy and advance our understanding of iron-sulfur protein electrochemistry.

SHAP analyses identified hydrophobicity (`seq.CTDC_HydrophobicityC3`, `Bar.prop.Hydrophobicity`), burial depth (`struct.burial_depth`), and cofactor type (`struct.is_hipip`) as critical predictors. Notably, `struct.is_hipip` reflects hydrophobic pockets in HiPIPs that stabilize the oxidized $[4Fe\text{-}4S]^{3+}$ state, elevating redox potentials (+50 to +450 mV), while burial depth and hydrophobicity modulate solvent access and local dielectric environments, key to electron transfer in iron-sulfur proteins. Potential pH bias in SF4 datasets, with 30/38 high pH entries ($\geq$ 7.6), likely amplified the importance of `pH` in SHAP analyses for SF4_bar, potentially skewing predictions toward pH-dependent effects. Ultimately, both local and global features influence predictions, underscoring the role of cofactor-specific and physicochemical properties. The study also highlighted key limitations: dataset heterogeneity, reliance on AlphaFold-predicted structures with cofactor implant inaccuracies, potential biases in pH distribution, and computational restrictions limiting hyperparameter optimization and larger radius testing.

Future work should focus on expanding the dataset with more experimentally resolved structures, particularly for underrepresented [3Fe-4S] and Fe3+ cofactors, to improve generalization. Incorporating dynamic (e.g., molecular dynamics) and quantum mechanical descriptors, alongside experimental validation of SHAP-identified features (e.g., hydrophobicity, burial depth), could further elucidate the structure–property relationships governing Fe-S protein redox behavior. Balancing pH distributions through targeted data collection or controlling pH in feature extraction would also mitigate biases in SF4 datasets. Additionally, exploring graph-based neural networks and leveraging optimized algorithms to reduce the 23-hour cross-validation time would enhance model accuracy and scalability.

Overall, this research demonstrates that machine learning, especially tree-based models, can provide valuable predictive insights into iron-sulfur protein redox potentials, paving the way for applications in protein engineering, synthetic biology, and biocatalyst design. With methodological refinements and expanded datasets, the accuracy and applicability of such computational predictions will continue to improve, deepening our understanding of these ancient and essential metalloproteins.

Bibliography

[Ahern et al., 2024] Ahern, K. R., Rajagopal, S., and Tan, W.-Y. (2024). Structure function – amino acids. In *Biochemistry Free For All* (LibreTexts). Accessed 2025-07-02.

[Arcinas et al., 2019] Arcinas, A. J., Maiocco, S. J., Elliott, S. J., Silakov, A., and Booker, S. J. (2019). Ferredoxins as interchangeable redox components in support of MiaB, a radical S-adenosylmethionine methylthiotransferase. *Protein Science*, 28(1):267–282.

[Banci et al., 1999] Banci, L., Bertini, I., Rosato, A., and Varani, G. (1999). Mitochondrial cytochromes c: a comparative analysis. *Journal of Biological Inorganic Chemistry*, 4(1):1–10.

[Bennett et al., 2023] Bennett, E. M., Murray, J. W., and Isalan, M. (2023). Engineering nitrogenases for synthetic nitrogen fixation: From pathway engineering to directed evolution. *BioDesign Research*, 5:0005.

[Berkovitch et al., 2004] Berkovitch, F., Nicolet, Y., Wan, J. T., Jarrett, J. T., and Drennan, C. L. (2004). Crystal structure of biotin synthase, an s-adenosylmethionine-dependent radical enzyme. *Science*, 303(5654):76–79.

[Birrell et al., 2016] Birrell, J. A., Laurich, C., Reijerse, E. J., Ogata, H., and Lubitz, W. (2016). Importance of hydrogen bonding in fine tuning the [2fe–2s] cluster redox potential of hydc from thermotoga maritima. *Biochemistry*, 55(31):4344–4355.

[Blaabjerg et al., 2024] Blaabjerg, L. M., Jonsson, N., Boomsma, W., Stein, A., and Lindorff-Larsen, K. (2024). SSEmb: A joint embedding of protein sequence and structure enables robust variant effect predictions. *Nature Communications*, 15:9646.

[Breiman, 2001] Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

[Cardenas-Rodriguez et al., 2018] Cardenas-Rodriguez, M., Chatzi, A., and Tokatlidis, K. (2018). Iron–sulfur clusters: from metals through mitochondria biogenesis to disease. *Journal of Biological Inorganic Chemistry*, 23(4):509–520.

[Chen and Guestrin, 2016] Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794.

[Cmglee, 2021] Cmglee (2021). Example of a reduction-oxidation reaction yielding sodium chloride with the "oil rig" mnemonic (redox_example.svg). Wikimedia Commons. Creative Commons Attribution-ShareAlike 4.0 International (CC BY-SA 4.0).

[Djaman et al., 2009] Djaman, O., Outten, F. W., and Rouault, T. A. (2009). Recent advances in iron-sulfur cluster biogenesis. *Chemistry & Biodiversity*, 6(6):1119–1134.

[Drucker et al., 1997] Drucker, H., Burges, C. J. C., Kaufman, L., Smola, A. J., and Vapnik, V. (1997). Support vector regression machines. *Advances in Neural Information Processing Systems*, 9:155–161.

[European Bioinformatics Institute, 2018] European Bioinformatics Institute (2018). CHEBI:30408 - iron-sulfur cluster.

[Frazzon and Dean, 2001] Frazzon, J. and Dean, D. R. (2001). Feedback regulation of iron–sulfur cluster biosynthesis. *Proceedings of the National Academy of Sciences*, 98(26):14751–14753.

[Galuzzi et al., 2022] Galuzzi, B. G., Mirarchi, A., Viganò, E. L., De Gioia, L., Damiani, C., and Arrigoni, F. (2022). Machine learning for efficient prediction of protein redox potential: The flavoproteins case. 62(19):4748–4759.

[Gambella et al., 2021] Gambella, C., Ghaddar, B., and Naoum-Sawaya, J. (2021). Optimization problems for machine learning: A survey. *European Journal of Operational Research*, 290(3):807–828.

[García-Guerrero et al., 2021] García-Guerrero, M. G., Chávez, M. A., de Anda, R., and Pérez-Rueda, E. (2021). Diversification of ferredoxins across living organisms. *Frontiers in Microbiology*, 12:653676.

[Guiza Beltran et al., 2024] Guiza Beltran, D., Wan, T., and Zhang, L. (2024). Whib-like proteins: Diversity of structure, function and mechanism. *Biochimica et Biophysica Acta (BBA) – Molecular Cell Research*, 1871(7):119787.

[Hagen, 2003] Hagen, W. R. (2003). Biochemical and spectroscopic characterization of iron-sulfur proteins. *Inorganic Chemistry*, 42(16):4946–4957.

[Hastie et al., 2009] Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, New York, NY, 2nd edition.

[Hekkelman et al., 2023] Hekkelman, M. L., De Vries, I., Joosten, R. P., and Perrakis, A. (2023). AlphaFill: enriching AlphaFold models with ligands and cofactors. 20(2):205–213.

[Imlay, 2006] Imlay, J. A. (2006). Iron–sulphur clusters and the problem with oxygen. *Molecular Microbiology*, 59(4):1073–1082.

[Jafari et al., 2022] Jafari, S., Tavares Santos, Y. A., Bergmann, J., Irani, M., and Ryde, U. (2022). Benchmark study of redox potential calculations for iron–sulfur clusters in proteins. *Inorganic Chemistry*, 61(16):5991–6007.

[Jia et al., 2024] Jia, L., Brémond, É., Zaida, L., Gaüzère, B., Tognetti, V., and Joubert, L. (2024). Predicting redox potentials by graph-based machine learning methods. *Journal of Computational Chemistry*, 45(28):2383–2396. Epub 2024 Jun 24.

[Jumper et al., 2021] Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., Bridgland, A., Meyer, C., Kohl, S. A. A., Ballard, A. J., Cowie, A., Romera-Paredes, B., Nikolov, S., Jain, R., Adler, J., Back, T., Petersen, S., Reiman, D., Clancy, E., Zielinski, M., Steinegger, M., Pacholska, M., Berghammer, T., Bodenstein, S., Silver, D., Vinyals, O., Senior, A. W., Kavukcuoglu, K., Kohli, P., and Hassabis, D. (2021). Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589.

[Krishtalik, 2003] Krishtalik, L. I. (2003). ph-dependent redox potential: how to use it correctly in the activation energy analysis. *Biochimica et Biophysica Acta*, 1604(1):13–21.

[Krol et al., 2010] Krol, J., Loedige, I., and Filipowicz, W. (2010). The widespread regulation of microrna biogenesis, function and decay. *Nature Reviews Genetics*, 11(9):597–610.

[Lundberg and Lee, 2017] Lundberg, S. M. and Lee, S.-I. (2017). A Unified Approach to Interpreting Model Predictions. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

[Meyer et al., 1983] Meyer, T. E., Przysiecki, C. T., Watkins, J. A., Bhattacharyya, A., Simondsen, R. P., Cusanovich, M. A., and Tollin, G. (1983). Correlation between rate constant for reduction and redox potential as a basis for systematic investigation of reaction mechanisms of electron transfer proteins. 80(22):6740–6744.

[Mirdita et al., 2022] Mirdita, M., Schütze, K., Moriwaki, Y., Heo, L., Ovchinnikov, S., and Steinegger, M. (2022). ColabFold: making protein folding accessible to all. 19(6):679–682. Publisher: Nature Publishing Group.

[Moreno et al., 1994] Moreno, C., Macedo, A. L., Moura, I., Moura, J. J., Moura, I., and LeGall, J. (1994). Redox properties of desulfovibrio gigas [fe3s4] and [fe4s4] ferredoxins and heterometal cubane-type clusters formed within the [fe3s4] core. square wave voltammetric studies. 53(3):219–234.

[National Center for Biotechnology Information, 2023a] National Center for Biotechnology Information (2023a). *Oxidation–Reduction Reactions*. National Library of Medicine (US), National Center for Biotechnology Information. Accessed 2025-07-02.

[National Center for Biotechnology Information, 2023b] National Center for Biotechnology Information (2023b). *Redox Signaling and Oxidative Stress*. National Library of Medicine (US), National Center for Biotechnology Information. Accessed 2025-07-02.

[OpenStax, 2019] OpenStax (N/A (approx. 2019)). *Chemistry 2e*. OpenStax. "Review of Redox Chemistry".

[Pang et al., 2011] Pang, A., Warren, M. J., and Pickersgill, R. W. (2011). Structure of PduT, a trimeric bacterial microcompartment protein with a 4fe–4s cluster-binding site. 67(2):91–96. Publisher: International Union of Crystallography.

[Patel, 2022] Patel, M. (2022). Train/test split in machine learning – the right way. MachineLearningPlus. Accessed 2025-07-02; CCBY-SA license.

[Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[Prince and Adams, 2018] Prince, R. and Adams, M. (2018). Oxidation–reduction properties of the two fe4_s4 clusters in the ferredoxin from *Clostridium pasteurianum*. *Journal of Biological Chemistry*, 147:110–118.

[Rasmussen and Williams, 2006] Rasmussen, C. E. and Williams, C. K. I. (2006). *Gaussian Processes for Machine Learning*. MIT Press, Cambridge, MA.

[RCSB PDB, nd] RCSB PDB (n.d.). Methods for determining structure. PDB-101: Learn by RCSB PDB. Accessed 2025-07-02.

[Rees, 2002] Rees, D. C. (2002). Great metalloclusters in enzymology. *Annual Review of Biochemistry*, 71:221–246.

[Rousset et al., 1998] Rousset, M., Montet, Y., Guigliarelli, B., Forget, N., Asso, M., Bertrand, P., Fontecilla-Camps, J. C., and Hatchikian, E. C. (1998). [3fe-4s] to [4fe-4s] cluster conversion in *Desulfovibrio fructosovorans* [NiFe] hydrogenase by site-directed mutagenesis. 95(20):11625–11630.

[Stephens et al., 1996] Stephens, P. J., Jollie, D. R., and Warshel, A. (1996). Protein control of redox potentials of iron–sulfur proteins. *Chemical Reviews*, 96(7):2491–2514.

[Sticht and Rösch, 1998] Sticht, H. and Rösch, P. (1998). The structure of iron–sulfur proteins. *Progress in Biophysics and Molecular Biology*, 70(2):95–136.

[Syriopoulos et al., 2023] Syriopoulos, P. K., Kalampalikis, N. G., Kotsiantis, S. B., and Vrahatis, M. N. (2023). knn classification: a review. *Annals of Mathematics and Artificial Intelligence*, 93(1):43–75.

[Terranova, 2024] Terranova, U. (2024). Iron–sulfur peptides mimicking ferredoxin for an efficient electron transfer to hydrogenase. *ChemBioChem*, 25(20):e202400380.

[UniProt Consortium, 2008] UniProt Consortium (2008). The universal protein resource (uniprot). *Nucleic Acids Research*, 36(Database issue):D190–D195.

[Yan et al., 2016] Yan, L., Lu, Y., and Li, X. (2016). A density functional theory protocol for the calculation of redox potentials of copper complexes. *Physical Chemistry Chemical Physics*, 18(7):5529–5536. Publisher: The Royal Society of Chemistry.

[Zhong et al., 2023] Zhong, Q., Xiao, X., Qiu, Y., Xu, Z., Chen, C., Chong, B., Zhao, X., Hai, S., Li, S., An, Z., and Dai, L. (2023). Protein posttranslational modifications in health and diseases: Functions, regulatory mechanisms, and therapeutic implications. *MedComm*, 4(3):e261.

[Zou and Hastie, 2005] Zou, H. and Hastie, T. (2005). Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society, Series B*, 67(2):301–320.

[Österlund et al., 2010] Österlund, L. L., Berg, A., and Ekborg, M. (2010). Redox models in chemistry textbooks for the upper secondary school: Friend or foe? *Chemistry Education Research and Practice*, 11:182–192.

A

Computational Implementation Details

A.1 Virtual Environments

Table A.1: Virtual environments used in this project

Environment	**Application**	**Key Packages**
`protonation-env`	Assigning protonation states to mutant structure PDB files	`pdb2pqr`, `numpy`, `scipy`, `pandas`, `csvkit`
`ambertools_env`	Running `tleap` to generate topology files for energy minimization	`ambertools`, `numpy`, `scipy`, `pandas`, `matplotlib`
`parmed_env`	Converting AMBER file format to GROMACS	`parmed`, `numpy`, `pandas`
`gromacs_env`	Running energy minimization and converting output files to PDB format	`gromacs`
`alphafill-env`	Converting *.ent* files to *.pdb* format and transplanting cofactors via PyMOL API	`gemmi`, `numpy`, `pandas`
`redox-env`	Extracting features, training ML models, analyzing and visualizing results	`PyBioMed`, `BioPython`, `xgboost`, `scikit-learn`, `shap`, `numpy`, `pandas`, `matplotlib`, `seaborn`, `pillow`

A.2 Energy Minimization

Listing A.1: `tleapOrders`, tleap program commands input file

```
source leaprc.protein.ff19SB
source leaprc.water.opc
loadAmberParams frcmod.ionslm_126_opc

X = loadpdb your_name_pqr.pdb

check X
solvatebox X OPCBOX 10

addIonsRand X Na+ 0
addIonsRand X Cl- 0

saveamberparm X your_name.top your_name.crd
savepdb X your_name_tleap.pdb
quit
```

A.3 Feature Extraction

Table A.2: Column names from `tableAmm.txt`, and their descriptions.

Feature	**Description**
Sigla	Single-letter amino acid code
Name	Three-letter name of the amino acid
Volume	Molecular volume of the side chain ($Å^3$)
Log(Solubility (m at 20^oC))	Logarithm of solubility in mol/L at 20°C
Hydrophobicity	Hydrophobicity (empirical scale)
Isoelectric point	Isoelectric point (pI)
P(helix)	Propensity to form alpha-helix
nOats in side chain	Number of oxygen atoms in the side chain
nSats in side chain	Number of sulfur atoms in the side chain
nNats in side chain	Number of nitrogen atoms in the side chain
Steric hindrance	Steric hindrance due to side chain bulk
nH-bonds	Number of hydrogen bonds the side chain can form
Vol / log(solub)	Volume divided by solubility log
Vol / P(helix)	Volume divided by helix propensity
Vol / P(sheet)	Volume divided by sheet propensity
log(Solub) x Flex	Solubility log multiplied by flexibility
Steric hindrance x Flexibility	Steric hindrance multiplied by flexibility
Flexibility	Flexibility score
P(helix)+P(sheet)	Sum of helix and sheet propensities
log(Solub) x Hydrophobicity	Solubility log multiplied by hydrophobicity
Hydrophobicity x Flex.	Hydrophobicity multiplied by flexibility
V / (P(helix)+P(sheet))	Volume divided by total secondary structure propensity
Steric hindrance / P(helix)	Steric hindrance divided by helix propensity
Isoelectric point / P(sheet)	Isoelectric point divided by sheet propensity
P(sheet)	Propensity to form beta-sheet
Vol x isoelectric point	Volume multiplied by isoelectric point
Isoelectric point / P(helix)	Isoelectric point divided by helix propensity
P(helix) / P(sheet)	Helix propensity divided by sheet propensity
Steric hindrance / P(sheet)	Steric hindrance divided by sheet propensity
P(helix) x Flex.	Helix propensity multiplied by flexibility

Table A.3: `PyBioMed`'s `CTD` module descriptors and groups

Property	**Description**	**Group 1**	**Group 2**	**Group 3**
Hydrophobicity	Measures the tendency of amino acids to avoid water (important for folding and membrane interactions)	Polar	Neutral	Hydrophobic
Normalized van der Waals Volume	Size-related property, reflecting the physical volume of amino acid side chains	Small	Medium	Large
Polarity	Categorizes amino acids based on the polarity of their side chains	Nonpolar	Polar uncharged	Polar charged
Charge	Classifies amino acids by their electrical charge at physiological pH	Negative	Neutral	Positive
Secondary Structure Propensity	Tendency of amino acids to form -helices, -sheets, or turns	Helix-formers	Sheet-formers	Turn-formers
Solvent Accessibility	Reflects how exposed an amino acid is to solvent in folded proteins	Buried	Intermediate	Exposed
Polarizability	Measures the ability of amino acids to be polarized in an electric field (related to electron cloud flexibility)	Low	Medium	High

A.4 Feature subsets Directory Structure

```
feature_datasets/
├── all_cofactors_bar/
│   ├── features_r1_all_bar.csv
│   ├── features_r2_all_bar.csv
│   ├── ...
│   └── features_r16_all_bar.csv
├── all_cofactors_protein/
│   └── features_all_protein.csv
├── complete_dataset/
│   ├── features_r1_all.csv
│   ├── ...
│   └── features_r16_all.csv
├── FES_all/
│   ├── features_r1_FES.csv
│   ├── ...
│   └── features_r16_FES.csv
├── FES_bar/
│   ├── features_r1_FES_bar.csv
│   ├── ...
│   └── features_r16_FES_bar.csv
├── FES_protein/
│   └── features_FES_protein.csv
├── SF4_all/
│   ├── features_r1_all.csv
│   ├── ...
│   └── features_r16_all.csv
├── SF4_bar/
│   ├── features_r1_SF4_bar.csv
│   ├── ...
│   └── features_r16_SF4_bar.csv
└── SF4_protein/
    └── features_SF4_protein.csv
```

A.5 Model Training

A.6 Supercomputer Resources

Listing A.2: Computational Resources Requested From DelftBlue

```
1 --partition=compute
2 --ntasks-per-node=1
3 --cpus-per-task=16
4 --mem-per-cpu=3G
5 --time=30:00:00
```

Algorithm 1 Machine Learning Pipeline Ensure Redox Potential Prediction, Radius-Dependent Data

Require: Data directory (`data_dir`), output directory (`output_dir`), target column (`target_column`)
Ensure: Trained models, performance metrics, visualizations, and analysis report
Training Phase:
for each radius $\in [1, 16\,\text{Å}]$ **do**
    Load feature data and target values
    Preprocess data: remove zero-variance and highly correlated features, impute missing values, apply scaling If required
    for each model $\in$ {LinearRegression, ElasticNet, SVR, GPR, KNN, RandomForest, GradientBoosting} **do**
        Configure pipeline with scaling and feature selection as needed
        Perform nested cross-validation: 5-fold outer KFold (10 repeats), 3-fold inner KFold with grid search
        Train model, optimize hyperparameters, and save best model
        Evaluate performance using MAE, RMSE, R^2, Spearman, Pearson, Explained Variance
        Generate learning curves and prediction analysis plots
    end for
    Identify top 3 models by MAE and perform SHAP analysis Ensure feature importance
end for
Save results (JSON, CSV) and visualizations (PNG)
Analysis Phase:
Load saved results from `ml_results.json`
Identify top 5 models by MAE with associated metrics
Analyze feature importance evolution across radii
Assess model stability using CV standard deviation and prediction range
Perform statistical significance test between model pairs
Evaluate radius trends with linear regression
Generate detailed comparison plots (MAE, R^2, optimal radius, distribution)
Compare with literature benchmarks
Generate and save analysis report

Model	**Scaling**	**Feature Selection**	**Hyperparameters**
Linear Regression	Yes	Yes	None
Elastic Net	Yes	Yes	$\alpha \in \{0.01, 0.1, 1.0, 10.0, 100.0\}$, $l1\_ratio \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$
SVR	Yes	Yes	$C \in \{0.1, 1, 10\}$, $\gamma \in \{\text{scale}, \text{auto}, 0.001, 0.01, 0.1, 1\}$, $\epsilon \in \{0.01, 0.1, 0.2\}$, kernel = rbf
GPR	Yes	Yes	kernel $\in \{\text{Constant} \times \text{RBF} + \text{White}, \text{Constant} \times \text{RBF}, \text{RBF} + \text{White}\}$, $\alpha \in \{10^{-10}, 10^{-8}, 10^{-6}, 10^{-4}\}$
KNN	Yes	Yes	$n\_neighbors \in \{3, 5, 7, 9, 11, 15\}$, weights $\in \{\text{uniform}, \text{distance}\}$, metric $\in \{\text{euclidean}, \text{manhattan}, \text{minkowski}\}$
Random Forest	No	No	$n\_estimators \in \{100, 200\}$, $max\_depth \in \{5, 7, 10\}$, $min\_samples\_split \in \{2, 5, 10\}$, $min\_samples\_leaf \in \{1, 2, 4\}$, $max\_features \in \{\text{sqrt}, \text{log2}\}$
XGB	No	No	$n\_estimators \in \{100, 200\}$, $learning\_rate \in \{0.05, 0.1\}$, $max\_depth \in \{3, 5\}$, $min\_samples\_split \in \{2, 5\}$, $min\_samples\_leaf \in \{1, 2, 4\}$, $subsample \in \{0.8, 1.0\}$

Table A.4: **Machine learning models and their configurations used in the redox potential prediction pipeline**. α in Elastic Net and GPR controls regularization strength, penalizing large coefficients to prevent overfitting (higher values increase penalty). `kernel` in SVR and GPR defines the transformation (e.g., 'rbf' [radial basis function] for nonlinear patterns, 'Constant × RBF' for Gaussian processes). γ in SVR adjusts the influence radius of training points (smaller values broaden it). ϵ in SVR sets the margin of tolerance for errors. `n_neighbors` in KNN specifies the number of nearest points considered, with `weights` ('uniform' or 'distance') and `metric` ('euclidean', 'manhattan', 'minkowski') affecting distance weighting. `max_depth` and `min_samples_split` in Random Forest and XGB limit tree depth and node splits to control complexity, while `subsample` in XGB fractions the training data per tree.

B

Additional Figures and Tables

B.1 Dataset Statistics

Table B.1: Mean, median, and standard deviation of the residue counts and maximum diameter for all proteins in the dataset, per cofactor type.

Cofactor	Residue Count			Max Dimension (Å)		
	Mean	Median	Std Dev	Mean	Median	Std Dev
[2Fe-2S]	163	159	74	62	55	19
[4Fe-4S]	138	81	140	43	37	17
[3Fe-4S]	169	105	122	49	38	20
Fe3+	53	54	1	27	27	1

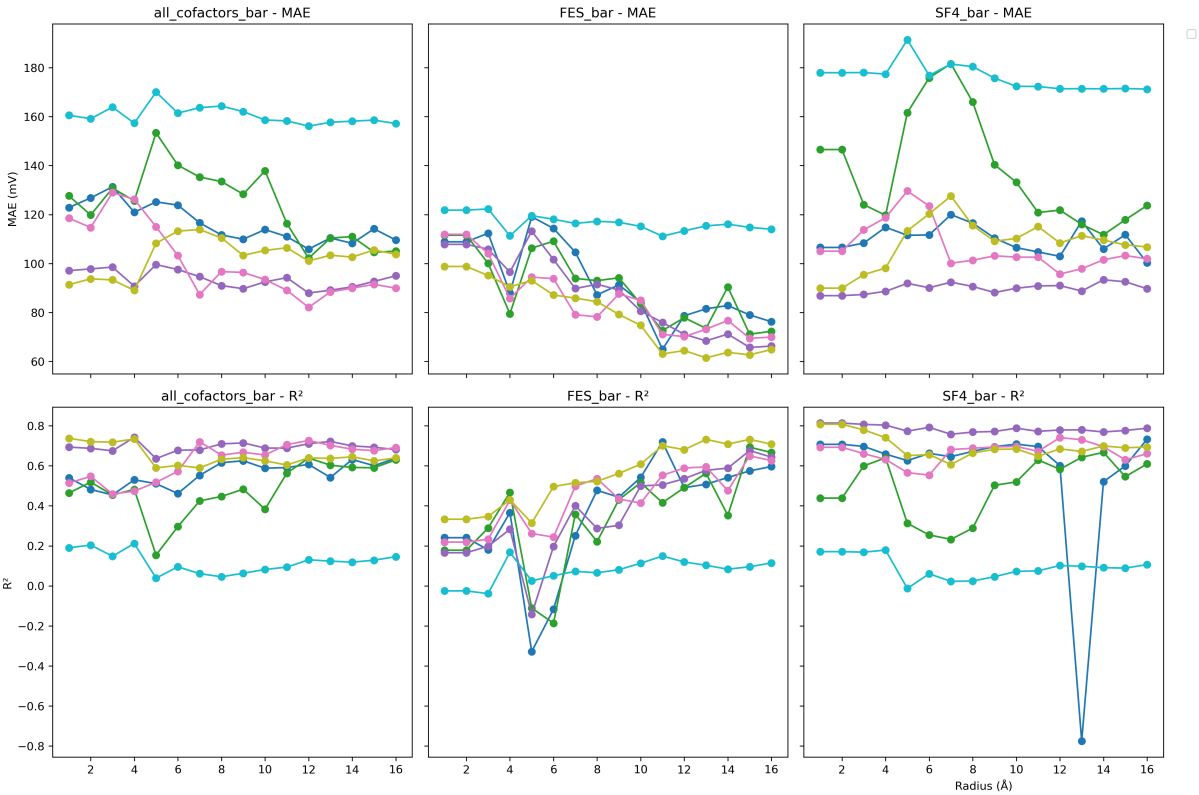

Figure B.1: Performance trends showing MAE vs. radius and R^2 vs. radius for three radius-dependent datasets all_cofactors_bar, FES_bar, and SF4_bar. Models are colored according to Figure 4.11 (. FES_bar clearly shows an improving performance with increasing radius for all models, while trends are less obvious in the other datasets.

Table B.2: Detailed statistics of iron-sulfur protein cofactors. Redox potential (Em) is in mV and burial depth is in Å.

		Em (mV)			Burial depth (Å)		
Cofactor	**Count**	**Mean**	**Std**	**Median**	**Mean**	**Std**	**Median**
[2Fe-2S]	60	-267.05	199.59	-343.0	6.00	3.55	5.14
[4Fe-4S]	92	-322.20	297.16	-403.5	8.30	4.88	6.24
[3Fe-4S]	10	-201.40	188.57	-200.0	10.19	7.51	7.18
Fe3+	6	-42.50	30.25	-52.0	4.25	0.04	4.24

B.2 Performance Trends

B.3 Feature Retention Analysis

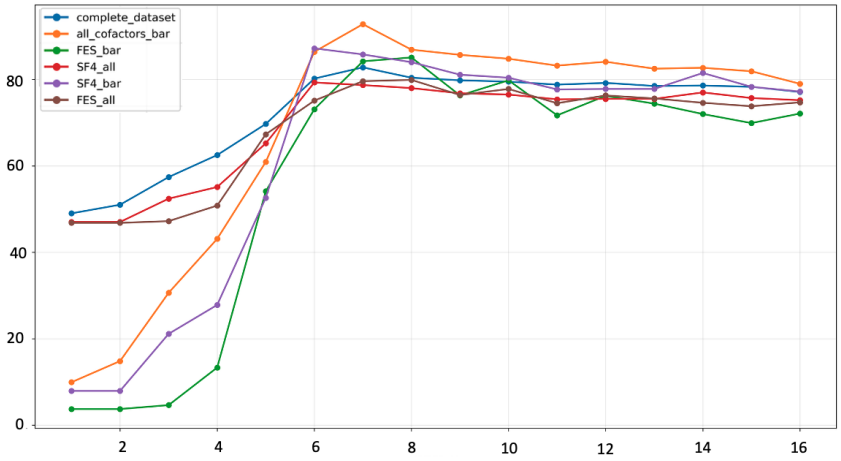

Figure B.2: Features retained per radius after preprocessing. The y-axis shows the percentage of features retained, and the x-axis shows the radius in Angstrom. We see that all datasets above a radius of 5Å retain approximately the same proportion of features.

B.4 Recurrent Features Summary

Table B.3 lists all recurrent features identified by SHAP analysis across the top 3 model-dataset combinations for all cofactors, FES, and SF4 datasets, as detailed in Tables 4.12, 4.13, and 4.14. The table includes each feature's biological relevance and the datasets and subsets (all features, bar, protein) where they are important, providing a comprehensive reference for the feature importance insights discussed in Section 5.4.

Table B.3: Summary of Recurrent Features, Their Biological Relevance, and Datasets Where Important

Feature	**Biological Relevance**	**Datasets (Subset)**
`struct.is_hipip`	Indicates HiPIP structure, stabilizing oxidized [4Fe-4S] clusters via hydrophobic environments, raising redox potential	All (All, Bar, Protein), SF4 (All, Bar, Protein)
`seq.CTDC_ _PolarityC1`	Proportion of polar amino acids, modulating dielectric environment and electron transfer efficiency	All (All)
`Bar.prop.nNats in side chain`	Nitrogen atoms in bar region side chains form hydrogen bonds or electrostatic interactions, affecting cofactor redox state	All (Bar)
`Protein.mean. Steric hindrance`	Average steric hindrance constrains cofactor accessibility, influencing electron transfer pathways	All (Protein)
`Protein.mean. Isoelectric point / P(sheet)`	Ratio of isoelectric point to sheet propensity reflects charge distribution and structural constraints, modulating electrostatic environment	All (Protein)
`Bar.mean. Isoelectric point`	Average isoelectric point in bar region affects local charge distribution, critical for [2Fe-2S] redox stability	FES (All)
`Bar.mean.Vol / P(helix)`	Ratio of residue volume to helix propensity in bar region influences packing density, affecting [2Fe-2S] cofactor stability	FES (Bar)
`Bar.mean.P(helix) x Flex.`	Product of helix propensity and flexibility in bar region modulates [2Fe-2S] cofactor accessibility via dynamic structural effects	FES (Bar)
`Protein.mean.nNats in side chain`	Fraction of nitrogen atoms in protein side chains contributes to global electrostatic interactions, influencing [2Fe-2S] redox potentials	FES (Protein)
`seq.CTDC_ _HydrophobicityC3`	Fraction of hydrophobic amino acids stabilizes [4Fe-4S] clusters by reducing solvent exposure, raising redox potential	SF4 (All)

Continued on next page

Table B.3 – continued from previous page

Feature	Biological Relevance	Datasets (Subset)
`struct. burial_depth`	Deeper cofactor burial reduces solvent accessibility, increasing [4Fe-4S] redox potential by stabilizing the reduced state	SF4 (All, Bar)
`Protein.mean. P(sheet)`	Higher sheet propensity enhances structural rigidity, modulating dielectric environment and electron transfer in [4Fe-4S] proteins	SF4 (All)
`Protein.mean. P(helix) x Flex.`	Product of helix propensity and flexibility reflects dynamic structural effects, influencing [4Fe-4S] cofactor accessibility	SF4 (All)
`pH`	Environmental pH affects residue protonation near [4Fe-4S] cofactor, directly influencing redox potential	SF4 (Bar)
`Bar.prop. Hydrophobicity`	Total hydrophobicity in bar region stabilizes [4Fe-4S] clusters, enhancing redox potential, similar to `seq.CTDC__HydrophobicityC3`	SF4 (Bar)

C

Heatmaps

C.1 Model Heatmaps

All model-specific heatmaps, except for XGB (already given in Figure 4.14) for performance on radius-dependent datasets, are provided in this Appendix. For performance on radius-independent heatmaps, see Figure 4.9. For visualization purposes, all heatmaps use the same range for the color mapping. The y-axis displays the dataset subset, and the x-axis displays the radius from the cofactor barycenter considered for the features, ranging from 1Å to 16Å going from left to right in increments of 1Å

Figure C.1: Linear Regression-specific heatmap for radius-dependent datasets

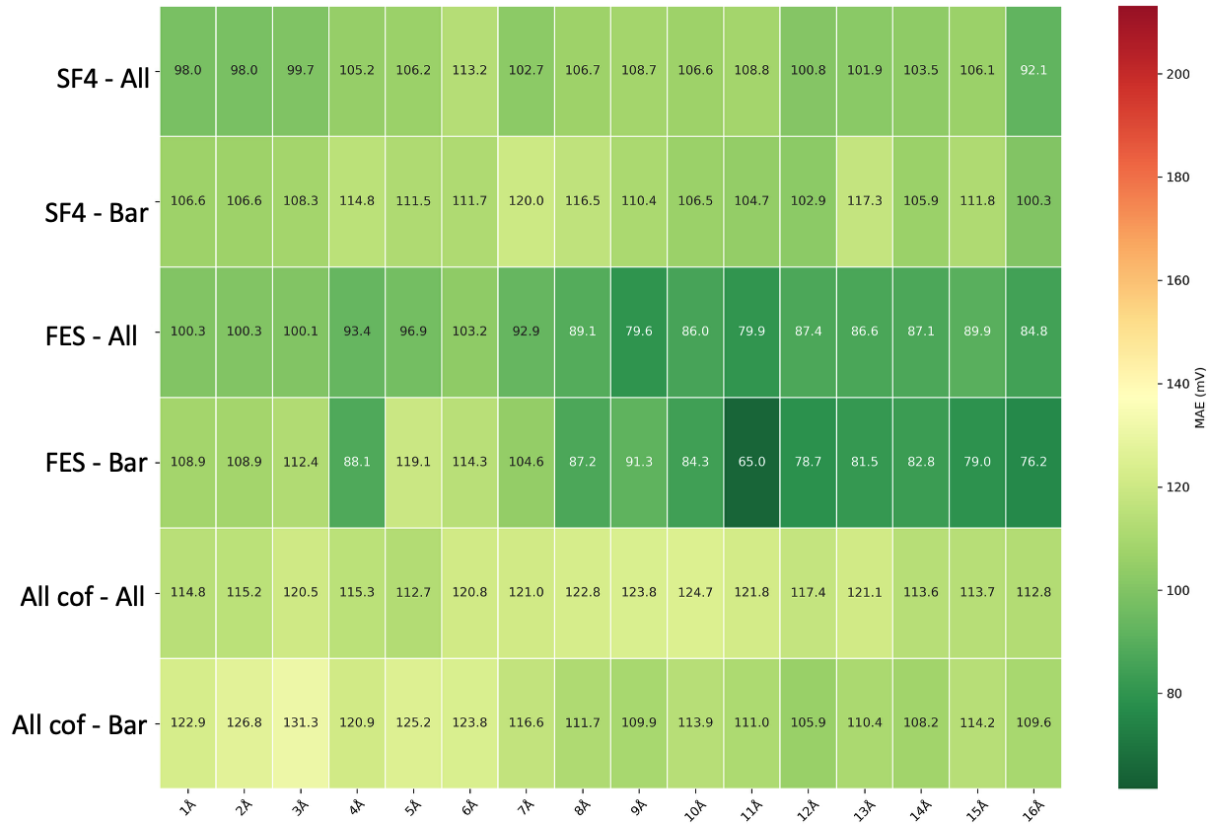

Figure C.2: Elastic Net's performance heatmap on all radius-dependent datasets.

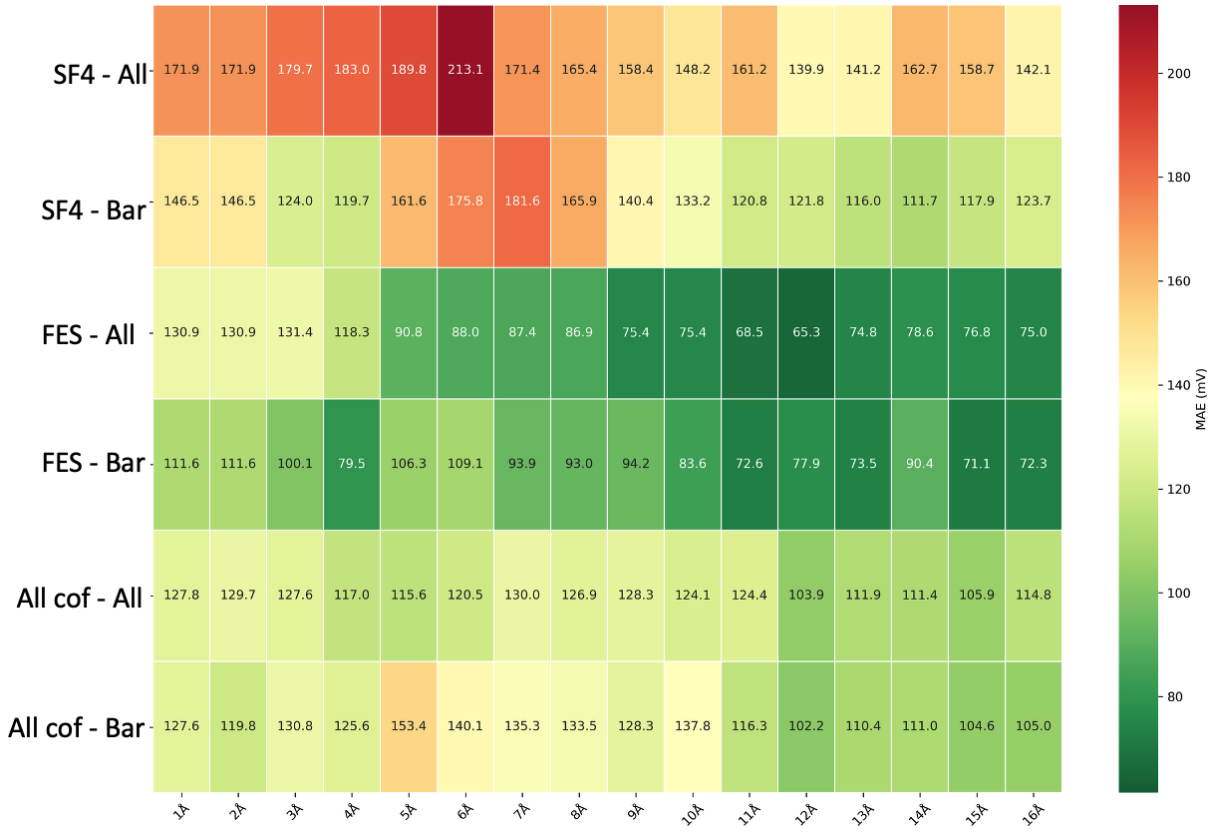

Figure C.3: Gaussian Process Regressor-specific heatmap for radius-dependent datasets.

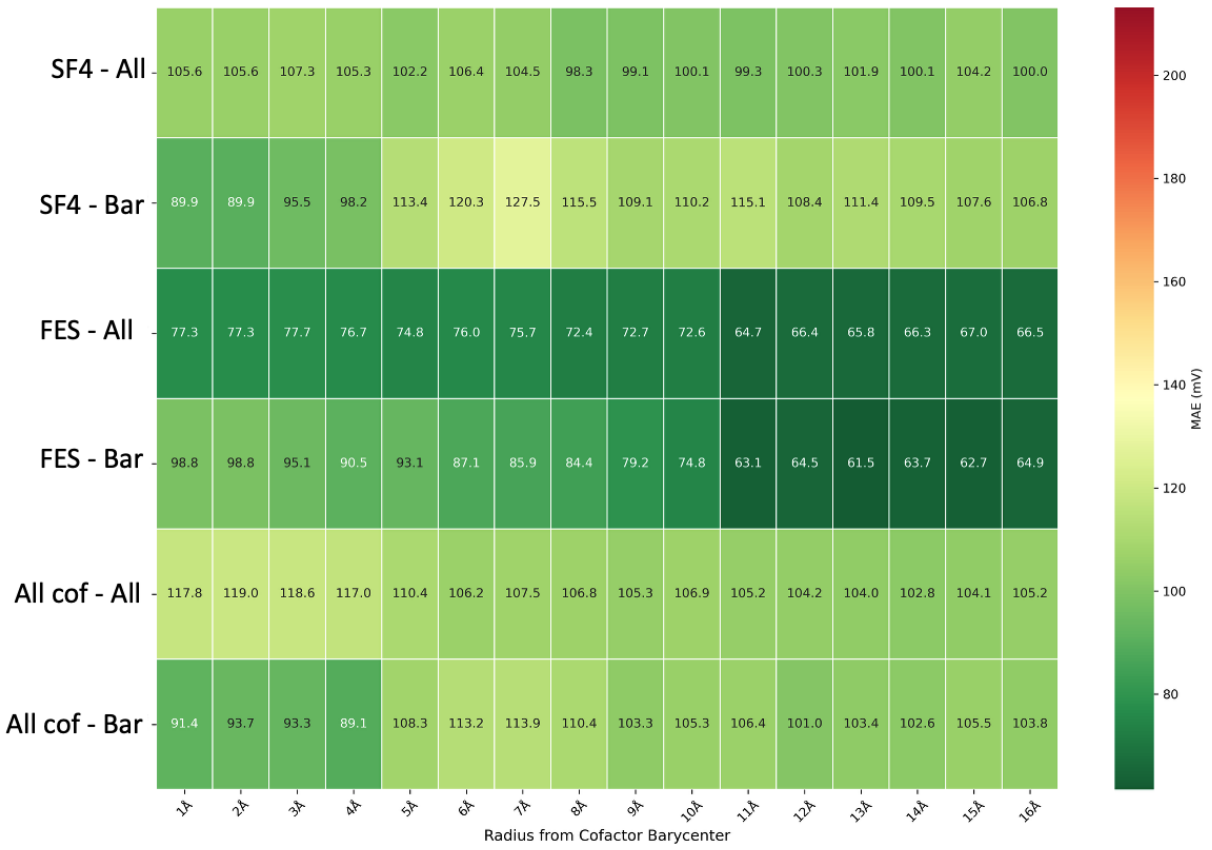

Figure C.5: Random Forest-specific heatmap for radius-dependent datasets.

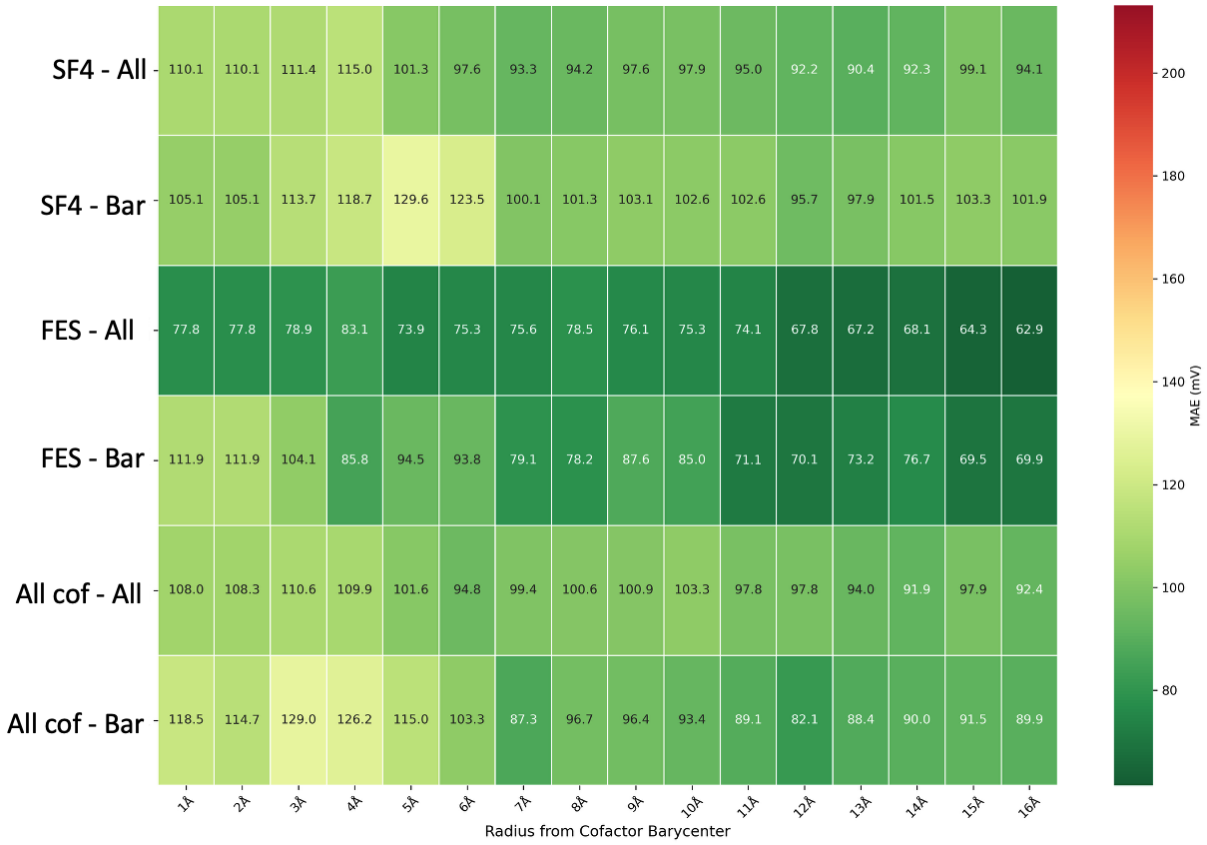

Figure C.4: K Nearest Neighbors Regressor-specific heatmap for radius-dependent datasets.

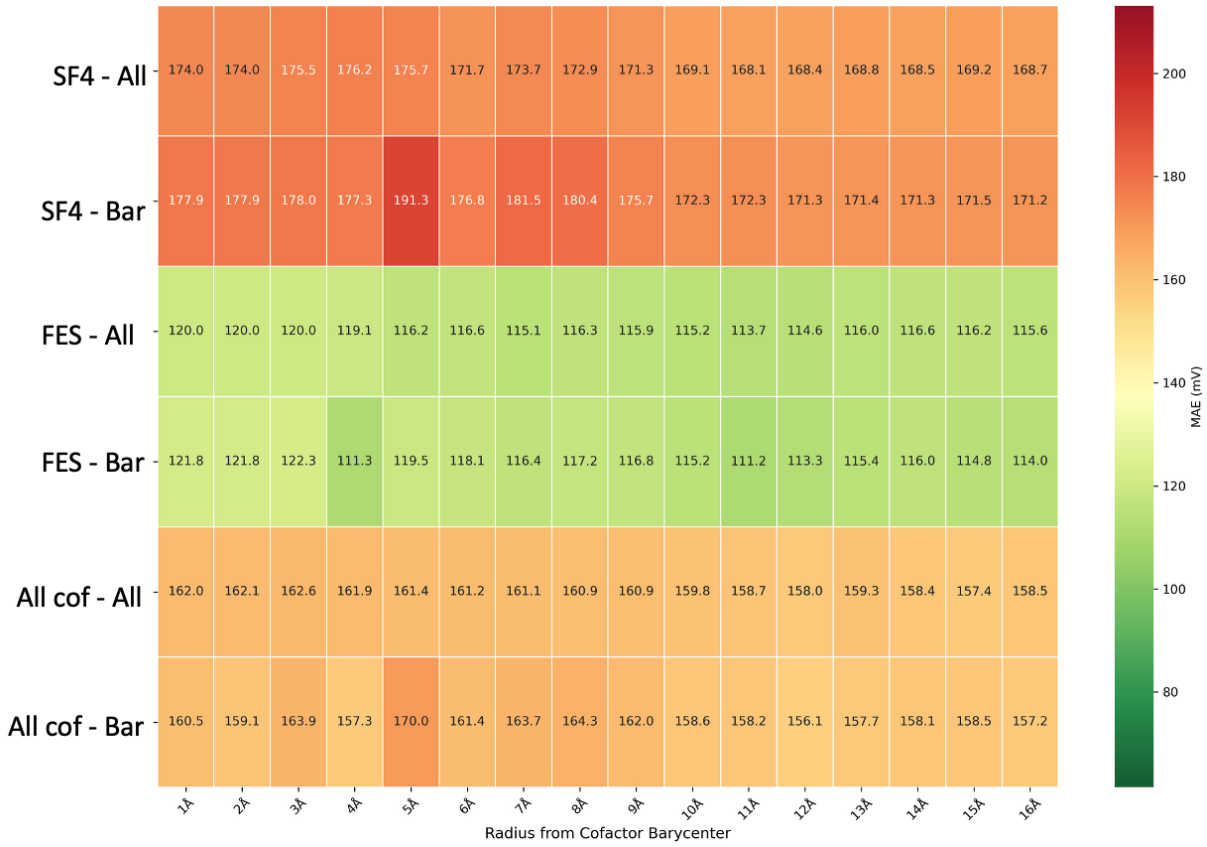

Figure C.6: Support Vector Regressor-specific heatmap for radius-dependent datasets.

D

SHAP Plots

In this appendix, the SHAP violin plots are provided for the top model (radius-independent datasets) or the top model-radius combination (radius-dependent datasets). In each plot, the top 20 features are displayed in descending order of importance on the y-axis, the distribution of the values is displayed on the x-axis, and the color represents the directional impact of the features, with red indicating a high feature value and blue representing a low feature value.

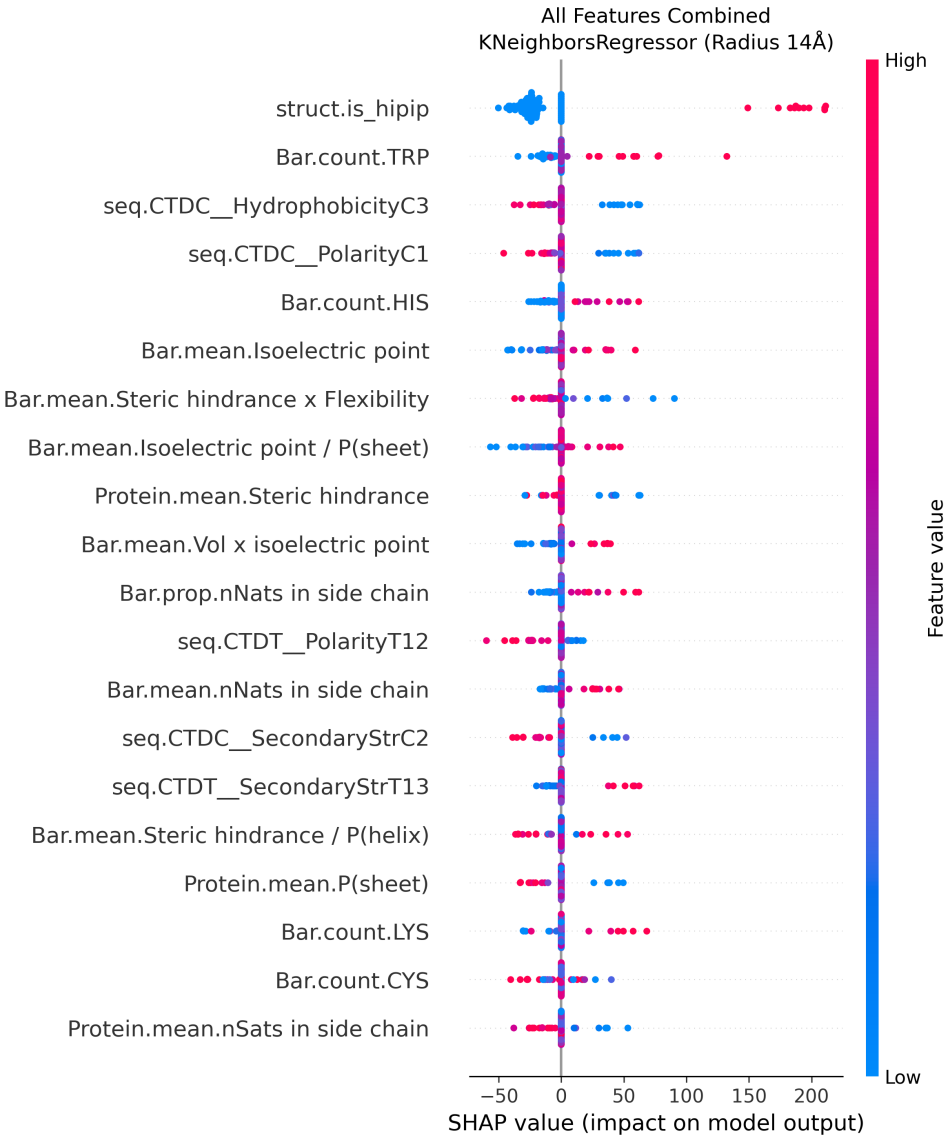

Figure D.1: SHAP violin plot for the complete dataset. The best model here was KNN for a radius of 14Å

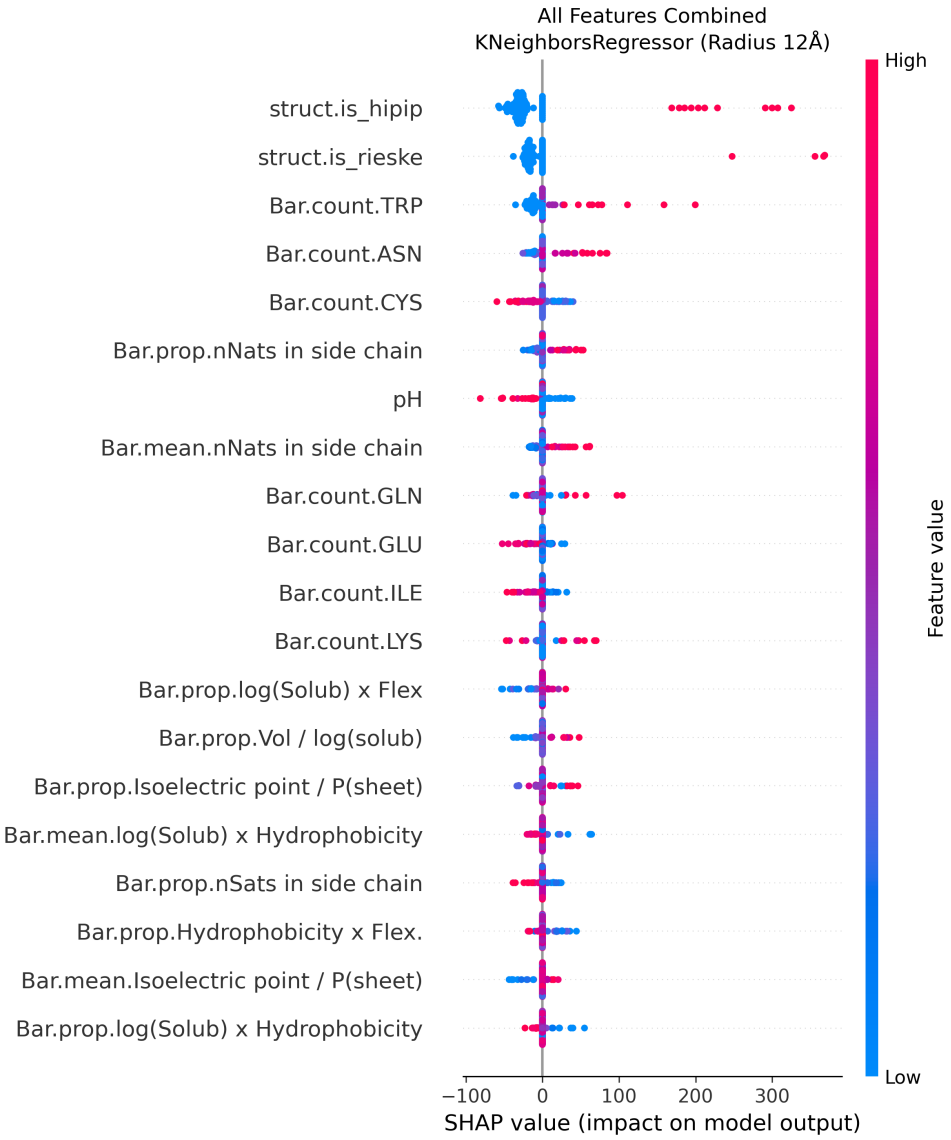

Figure D.2: SHAP violin plot for dataset all_cofactors_bar. The best model here was KNN for a radius of 12Å

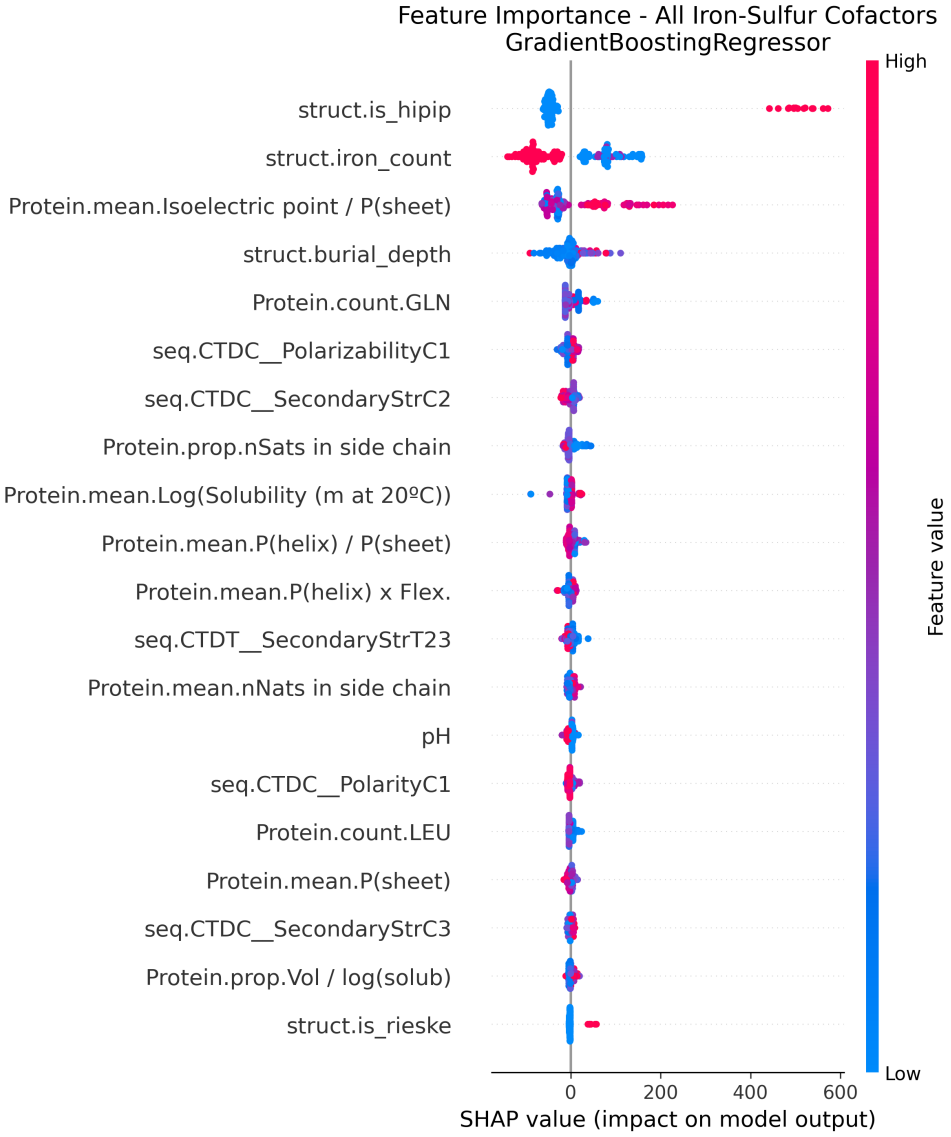

Figure D.3: SHAP violin plot for dataset all_cofactors_protein. The best model here was XGB.

Figure D.4: SHAP violin plot for dataset FES_all. The best model here was KNN for a radius of 16Å.

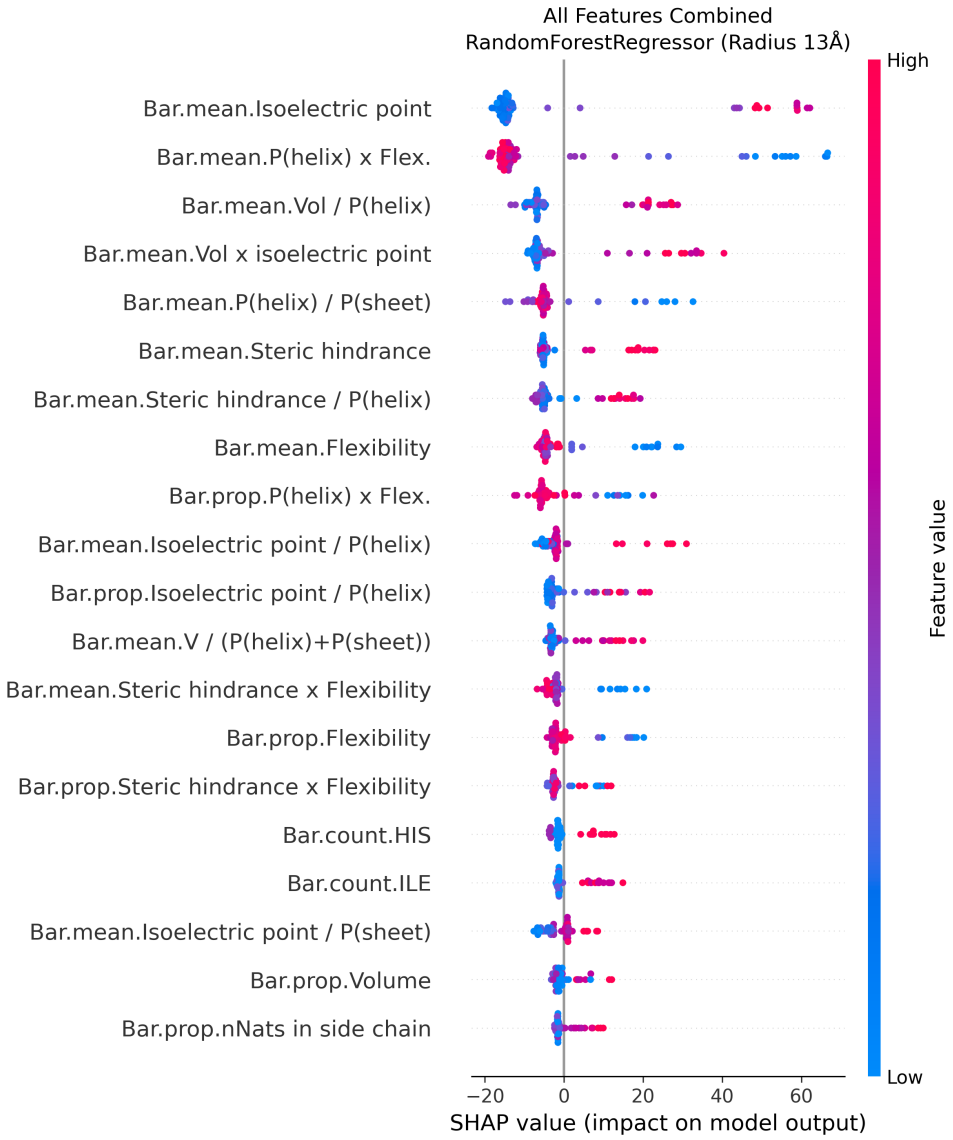

Figure D.5: SHAP violin plot for dataset FES_bar. The best model here was RF for a radius of 13Å

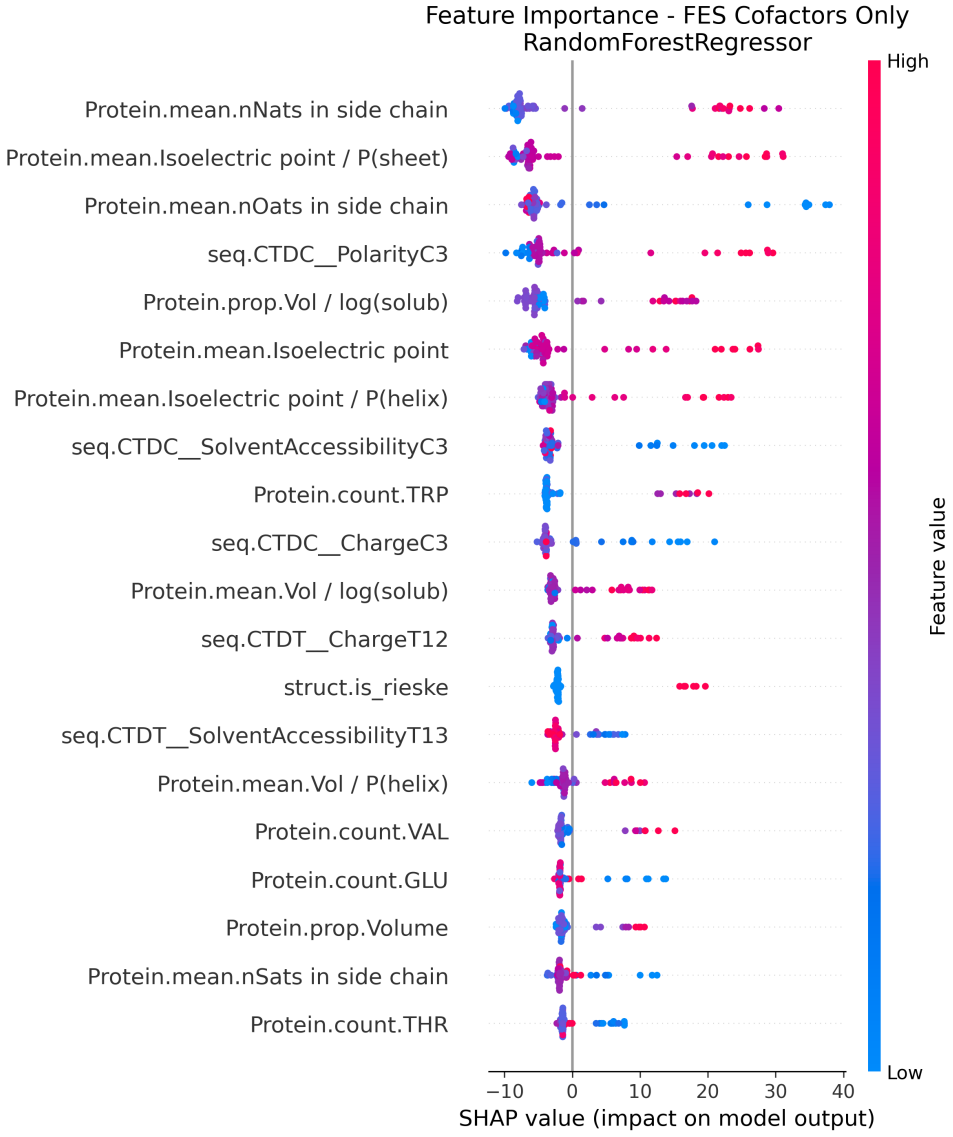

Figure D.6: SHAP violin plot for dataset FES_protein. The best model here was RF.

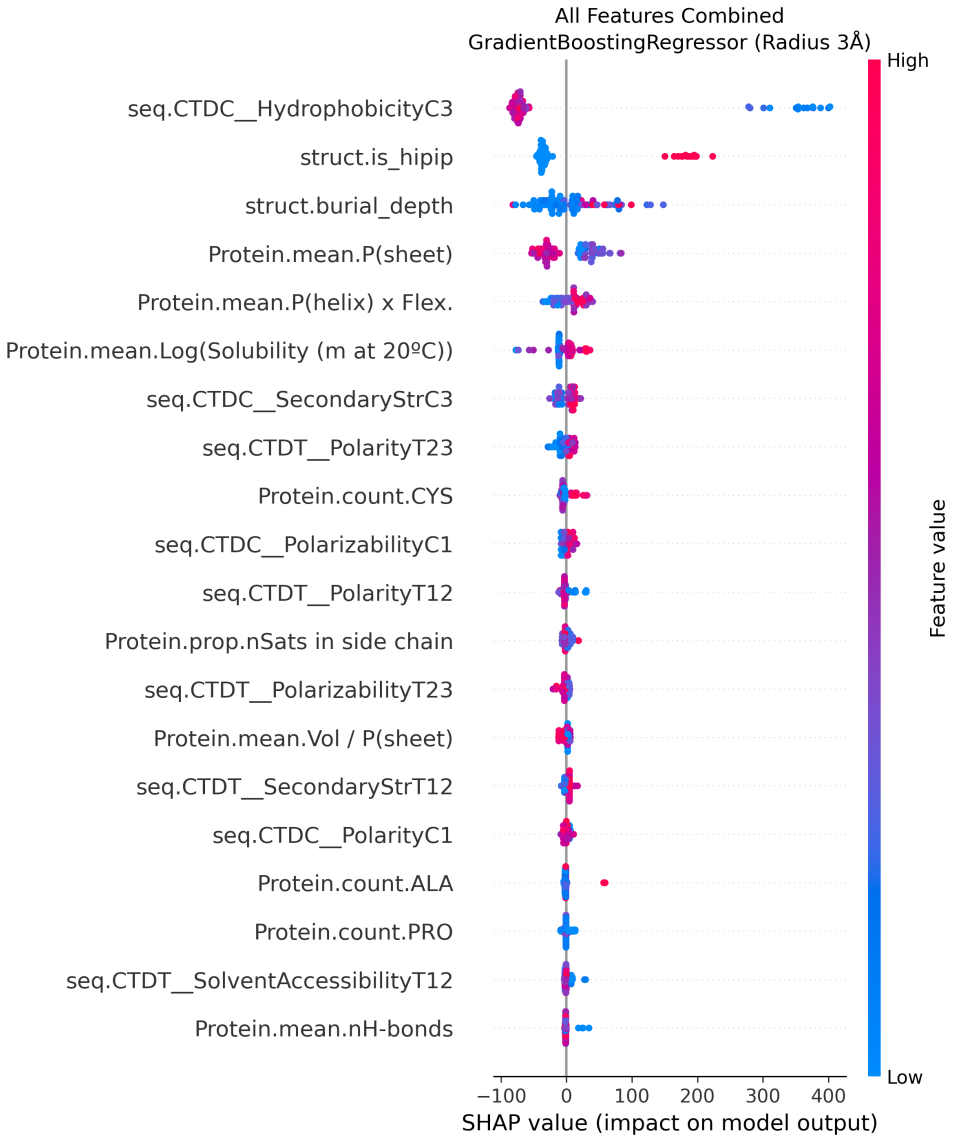

Figure D.7: SHAP violin plot for dataset SF4_all. The best model here was XGB for a radius of 3Å.

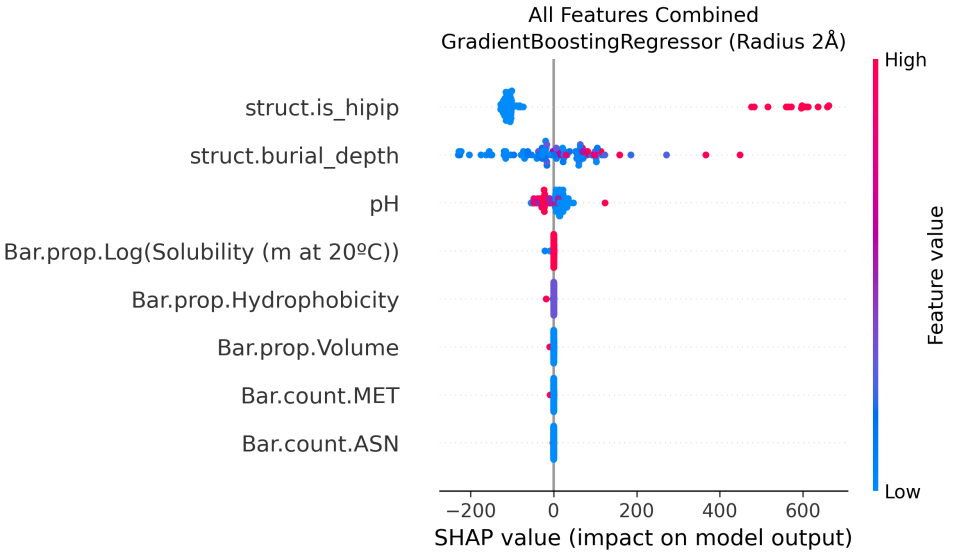

Figure D.8: SHAP violin plot for dataset SF4_bar. The second-best model here was XGB for a radius of 2Å. This figure is provided to show its similarity to the top performance on this dataset, of XGB at radius 1Å, shown in Figure 4.18.

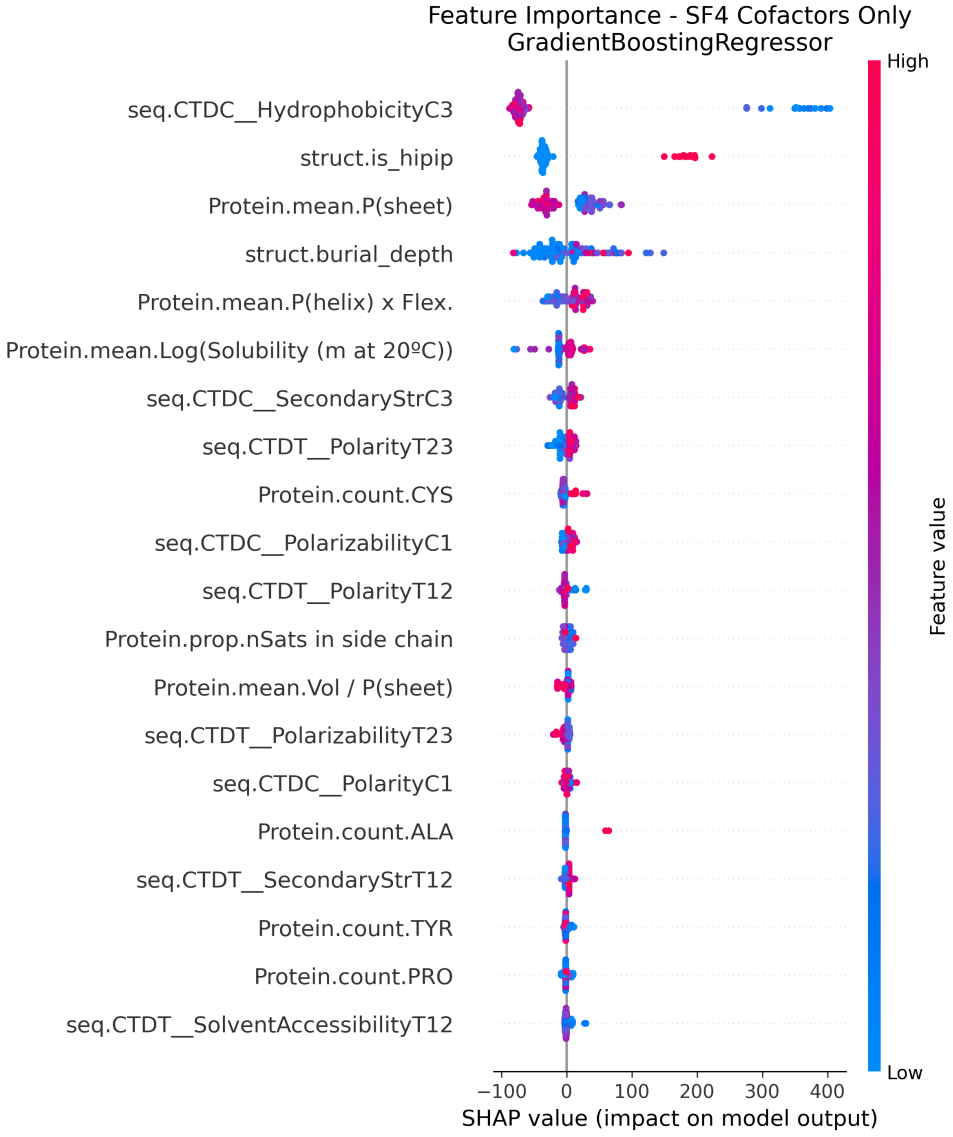

Figure D.9: SHAP violin plot for dataset SF4_protein. The best model here was XGB.

E

Detailed Methodology

This appendix outlines the detailed computational methodology employed to predict redox potentials of iron-sulfur proteins, adding information on script functions and input/output, naming conventions, and directory structures to allow better reproducibility and complement Chapter 3.

E.1 Data Collection

E.1.1 Data Collection Format

The data for this project was recorded in the following formats:

- An Excel sheet containing information on the proteins, their cofactors, and corresponding redox potentials. There is then one row per cofactor/redox potential.
- A folder containing the 3D protein structures from AlphaFold, as *.pdb* files..
- A folder containing the 3D protein structures from crystallography, if available, in the default *.ent* file format.

The Excel sheet is filled with the variables described in table E.1. Some extra notes to accompany this table:

- **Mutations** were all substitutions, written using single-letter amino acid codes, in the format `{original amino acid}+{location in sequence}+{replacement amino acid}`. If there are multiple mutations in the same protein, they are joined with an underscore.
  - For example, mutation entry `D13G_V14G` represents an Aspartic Acid in position 13 and a Valine in position 14, each being substituted by a Guanine.
- **Classification** examples include *electron transport* and *oxidoreductase*.
- **Cofactors** are written as: [Fe3+], [2Fe-2S], [3Fe-4S], [4Fe-4S].
- **Family** is based on the **cofactor**. Among others, we have:

  - Rubredoxins, HiPIPs, Rieske proteins, Nitrogenases
  - [3Fe-4S]Ferredoxins, [4Fe-4S]Ferredoxins
  - [4Fe-4S]/[3Fe-4S]Ferredoxins, with one 4-Iron and one 3-Iron cofactor
  - 2[4Fe-4S]Ferredoxins, with two 4-Iron cofactors
  - 2[4Fe-4S]/[3Fe-4S]Ferredoxins, with two 4-Iron cofactors and one 3-Iron cofactor
  - and so on...
- **Which cofactor** gives us a way to differentiate the cofactors in the case of 2[4Fe-4S]/[3Fe-4S]Ferredoxins and 2[4Fe-4S]Ferredoxins.
  - The **proximal** cofactor is more buried and has a lower (more negative) redox potential. It is closer to the electron donor in the electron transport chain.
  - The **distal** cofactor is less buried, has a higher redox potential, and is closer to the electron acceptor.
  - The **intermediate** cofactor is always the [3Fe-4S] cofactor in 2[4Fe-4S][3Fe-4S]Ferredoxins, and, as the name indicates, is usually located in between the two [4Fe-4S] cofactors, mediating the electron transport.

Table E.1: Data collection variables and descriptions

Variable	**Description**
UniProt ID	Protein's unique identifier from UniProt database
PDB	PDB ID for the protein crystal structure, if any
Name	Name of the protein as given in UniProt
Organism	Organism to which the protein belongs
Organism (alternative name)	Alternative/old name for the organism, if any
Strain	Organism strain, as mentioned in UniProt or on paper, if any
Mutation	'no' or the mutation
Classification	Main function of the protein
Family	Type of iron-sulfur protein
Cofactor	Type of iron-sulfur cluster
Which cofactor	If multiple cofactors in the same protein: 'proximal,' 'distal,' or 'intermediate'
pH	pH at which the redox potential was measured
Em	Redox potential
Reference – redox potential	DOI for redox potential paper
Reference – structure	DOI for crystal structure paper, if any
Resolution (Angstrom)	Crystal structure resolution, if any
Notes	Anything of relevance about the entry to note

E.1.2 Finding Redox Values and protein 3D Strctures

UniProt database search I conducted the first phase of the search for iron-sulfur cluster proteins in UniProt, a freely accessible database of protein sequences and other functional information. Two filters were applied, joined by an "AND" operator:

1. *Function / Cofactor / ChEBI term*: "30408"
2. *Function / Biophysicochemical properties / Redox potential*: "mV"

The first filter identifies proteins with a specific chemical compound as their cofactor, given by the ChEBI term. ChEBI, short for Chemical Entities of Biological Interest, is a sort of dictionary of molecular entities, more specifically focused on "small" chemical compounds, a category which iron-sulfur clusters fall under. The term used in this search, **30408**, is for iron-sulfur clusters, described as "a unit comprising two or more iron atoms and bridging sulfur ligands," [European Bioinformatics Institute, 2018]. This term does not include rubredoxins, because at this stage of my research, I did not yet know they existed; all my data for rubredoxins was subsequently extracted directly from scientific papers. Applying only this first filter yielded over 12,000 results, so to refine it more, I applied the second filter, which looks for the keyword "mV" in the redox potential property of a protein entry, which efficiently isolates the entries for which a redox potential is noted in UniProt. The final search query is then:

```
(cc_cofactor_chebi:‘‘CHEBI:30408’’) AND (cc_bpcp_redox_potential:mV)
```

Then, I moved my search to Scopus.

Scopus database search I applied the following two conditions, joined by an "AND" operator:

- Within the keywords, abstract, and title: "redox" AND "iron-sulfur" AND "mV"
- Within the keywords, abstract, and title: "ferredoxin*" OR "rubredoxin*" OR "rieske" OR "hipip*"

The asterisk represents any string of characters, to expand our search. The resulting search query is:

```
(TITLE-ABS-KEY( iron-sulfur ) AND TITLE-ABS-KEY( redox ) AND
TITLE-ABS-KEY( mv ) AND (TITLE-ABS-KEY( ferredoxin* ) OR TITLE-ABS-KEY(
rubredoxin* ) OR TITLE-ABS-KEY( rieske ) OR TITLE-ABS-KEY( hipip* )))
```

which yields 202 documents. Then, I checked the abstract to see if the "mV" values are indeed referring to redox potentials of the iron-sulfur proteins. If these conditions were fulfilled, I looked inside the paper and filled the Excel datasheet accordingly.

Downloading structures To find the proteins on UniProt, I searched for the organism name and protein name as given in the papers. It was usually pretty straightforward to find the right protein, and the 3D structure could subsequently be downloaded from the "Structure" section, and any other missing information on the Excel sheet could be recorded. To validate the structures, I:

- Looked to see if most of the AlphaFold structure was colored dark blue, indicating a high confidence prediction score.

- Compared the AlphaFold structure with the crystallography or NMR structure, if available, to see if they were similar.

Structure files were organized and named according to the following convention:

- **Wild types:** File name `{UniProt ID}_WT_Structure.pdb`, within a folder `FeS_WildType_structures`
- **Mutants:** File name `{UniProt ID}_{Mutation}_Structure.pdb`, within a folder `FeS_mutant_structures_uniminimized`

Saving references All the papers I read and extracted information from throughout this project were saved on Zotero, tagged with the UniProt ID(s) of the protein(s) they mention, and/or with other meaningful tags (like "crystal structure" or "ML").

E.2 Mutant Structures

During the data collection step, some mutant entries were recorded containing 1 to 3 amino acid substitutions. Unfortunately, none of these entries have crystal structures available, and AlphaFold is not sensitive enough to these small sequence changes to provide accurate mutant predictions. Therefore, to obtain the structures of mutant entries, mutations were first induced in the wild-type AlphaFold PDB files, followed by energy minimization to ensure physically realistic conformations.

E.2.1 Tools and Packages for Energy Minimization

A combination of specialized software tools and Python packages is used to carry out energy minimization on the mutated protein structures. Each tool contributed to specific stages in the pipeline, from structural manipulation to physical optimization. Table E.2 summarizes the sequential steps and corresponding software tools used in this process.

To ensure reproducibility and avoid conflicts between software dependencies, separate virtual environments were created for different stages of the project. A virtual environment is an isolated workspace that contains its own installation of Python and libraries, enabling users to install and switch between different versions of software without conflicts. The environments were managed using `mamba`, a faster alternative to `Conda` when it comes to resolving dependencies. More information is provided in Table A.1, which summarizes the main virtual environments created for this project, their respective roles, and the key packages installed in each.

Detailed implementation of this pipeline, including code-level execution and file handling, is described in the following subsection.

E.2.2 Energy Minimization Detailed Pipeline

This subsection provides a technical description of the energy minimization pipeline used for the mutant protein structures. Each stage builds upon the tools introduced in Section E.2.1, implementing them via Python scripting or shell commands to automate a reproducible workflow.

Table E.2: Software tools and packages used for energy minimization

Step	**Tool / Package**	**Role in Project**
1. Mutation induction	`PyMOL (API)`	Used via its Python API to induce site-specific amino acid substitutions in AlphaFold-predicted protein structures. This allowed structural modifications directly in the PDB files.
2. Protonation assignment	`PDB2PQR`	Assigned protonation states based on pH-specific considerations. Generated PQR files needed for subsequent charge and electrostatics calculations.
3. Charge calculation	Base Python, `pandas`	Used to parse the PQR files and calculate the net charge of each protein, which determined the number of ions needed for system neutralization.
4. System preparation	`AmberTools / tleap`	Used to solvate the protein, add counterions, add missing Hydrogen atoms, and prepare the system using the AMBER force field. Output included topology files and coordinate files in AMBER format.
5. File format conversion	`ParmEd`	Converted AMBER-formatted topology and coordinate files into formats compatible with GROMACS (.gro and .top), ensuring interoperability between software tools.
6. Energy minimization	`GROMACS`	Performed the actual energy minimization using a steepest descent algorithm.
7. File format conversion	Base Python, `subprocess`	Used to convert minimized structures (.gro files) back to PDB format.
8. Structure extraction	Base Python	Used to parse the PDB files and remove the water atoms and ions around the minimized structure.
9. Reassign missing chain IDs	Base Python	Used to parse the PDB files and reassign amino acids in the minimized structure to chain 'A' for analysis.

Step 0: Mutation indices First, as discussed in section 2.3, the AlphaFold structure contains the entire, immature protein, while the protein sequence used by researchers typically has already undergone all post-translational modifications. This means that the mutation index extracted from the papers and indicated in the "Mutations" column of the Excel dataset is not correct in the AlphaFold PDB file. To address this issue, I created a new column on the Excel table, named "Mutations_AF," which contains, like the original "Mutations" column, either 'no' or the mutation at the corrected index. To find this index in the full AlphaFold sequence, I looked at every mutant protein's entry on UniProt, in the *PTM/Processing* section, and found the position of the "Chain" in the sequence. The index of the mutation in AlphaFold is then the index of the start of the chain plus the mutation index from the paper, minus 1 (to avoid counting the first residue of the chain

twice). An example is given in Figure E.1.

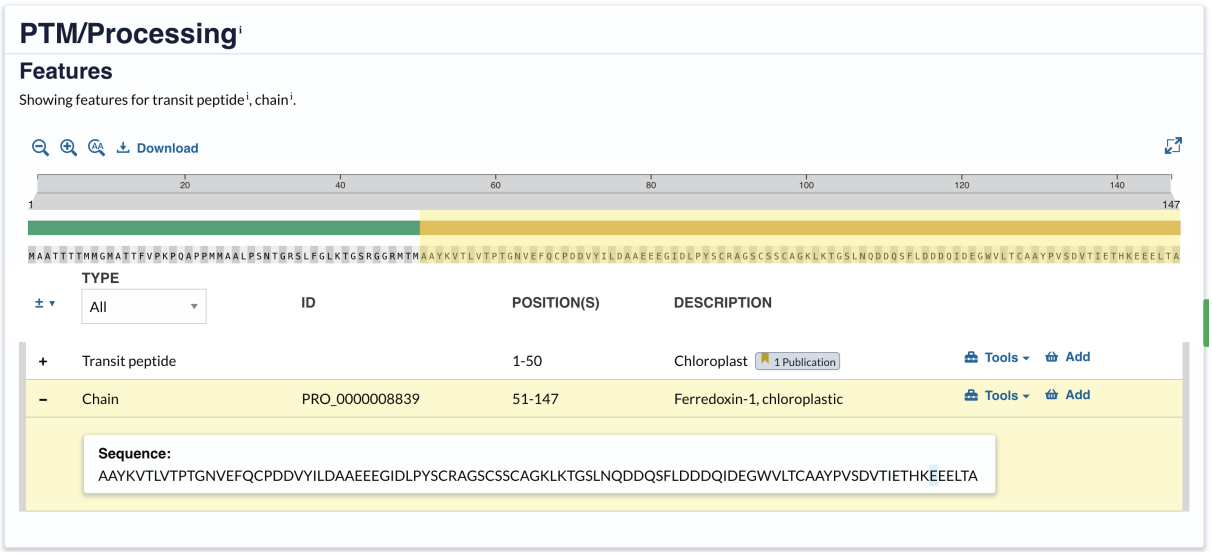

Figure E.1: Screenshot of entry P00221's UniProt page in the *PTM/Processing* section. Aliverti et al. induced a mutation at position 92 in the mature protein, which is position 142 (51+92-1) (highlighted here in blue) in the Alphafold sequence.

Step 1: Inducing mutations All mutations were induced using the script `mutate_proteins.py`, which takes as input:

- A folder containing the wild-type protein PDB files
- The Excel dataset, with "Mutations_AF" column

And gives as output:

- A folder containing the mutant PDB files, all named `{UniProt ID}_{Mutation}.pdb`
- A CSV file that logs whether the entry is skipped because it is a wild-type, or if a mutation is applied. For the latter, it records how many mutations, if they were successful, and logs the errors if not.

Step 2: Assigning protonation states Protonation states are assigned using `pdb2pqr`, installed in virtual environment `protonation-env`, with the script `assign_protonation.sh`, which is written in bash instead of Python for easier CSV parsing. It takes as input:

- The folder containing mutant PDB files, output of **Step 1**.
- The Excel dataset, with pH values

And gives as output:

- Protonated structure files, in *.pqr* format
- A CSV file that logs status information on the protonation assignment (success, or failure and error)

Step 3: Calculating protein charge The charge of the protonated structures is calculated in the script `generate_solvation_parameters.py`, which takes as input:

- The folder containing the protonated structure files, output of **Step 2**.

And gives as output:

- The solvation parameters: a CSV file with the charge of the protein, and the number of Na+ or Cl- ions needed to neutralize the structure.

Step 4.a: `tleap` preparation The `tleap` program command files are generated for each mutant with the script `generate_tleap_inputs.py`, which takes as input:

- The folder containing the protonated structure files, output of **Step 2**.
- The solvation parameter CSV file, output of **Step 3**
- A template file for tleap commands, presented in Listings A.1

And gives as output, within a root folder `tleap_runs`, subfolders for each mutant, containing:

- The mutant structure, stripped of its hydrogen atoms and saved as a *.pdb* file.
- A customized `tleap` commands file, with a notable parameter of **10Å** of padding around the structure.

The hydrogens were removed from the structure because, when running the `tleap` commands, if the parametrization is not successful, it is often because of unrecognized hydrogen atoms. To avoid this issue altogether, they are removed before converting the file from PQR to PDB format. In **Step 4.b**, `tleap` will add them back according to its force field.

Step 4.b: System preparation The `tleap` commands are executed using AMBER, installed in the `ambertools_env` virtual environment, via the script `run_tleap_all.py`, which takes as input:

- The folder `tleap_runs`, output of **Step 4.a**.

And gives as output, in the corresponding structure subfolder within the `tleap_runs` folder:

- The solvated system files (protein structure with hydrogens, in a water box with ions)
- Log files capturing the `tleap` output

Step 5: AMBER to GROMACS file conversion The AMBER-format output files from **Step 4.b** are converted to formats supported by GROMACS using `ParmEd`, installed in the `parmed_env` virtual environment, via the script `convert_to_gromacs.py`, which takes as input:

- The AMBER topology (.top) and coordinate (.crd) files in the folder `tleap_runs`, output of **Step 4.b**.

And gives as output, still in the structure subfolders:

- GROMACS-compatible topology (.top) and structure (.gro) files, with suffix "`_gro`" to differentiate them from the AMBER topology files.

Step 6: Energy minimization Finally, the energy minimization is run using GROMACS, installed in the `gromacs_env` virtual environment, via the script `run_minimization.py`, which takes as input:

- The GROMACS topology (.top) and coordinate (.crd) files in the folder `tleap_runs`, output of **Step 5**.
- A minimization parameters file (minim.mdp).

And gives as output, still in the structure subfolders:

- Energy minimized structures, with suffix "`_minim`".
- Log files documenting the minimization process

Step 7: File conversion from GROMACS to PDB The energy minimization outputs a lot of files, but we are only interested in the file ending in "_minim.gro" which contains the final coordinates of our system after minimization. To use these for the rest of the report, we first need to convert them to PDB format, using the script `prepare_alphafill_inputs.py`, which takes as input:

- The directory containing the minimized structures.

And gives as output, in a new folder:

- PDB files of the minimized structures.

Step 8: Structure extraction The next step is to clean up the PDB file by removing all water atoms and ions added for minimization, leaving behind only the minimized structure. This is done with the `remove_water_box.py` script, which takes as input:

- The directory containing the minimized structure PDB files.

And gives as output, in a new folder:

- Cleaned up PDB files of the minimized structures.

Step 9: Reassigning chain IDs The energy minimization removed the chain IDs, which led to errors during the feature extraction step when the code would try to get the sequence of the protein. This was fixed by assigning all amino acids to chain 'A,' which is correct since all the structures involved are monomers. The script used is `add_chain_ids.py` script, which takes as input:

- The directory containing the minimized structure PDB files.

And gives as output, in the same folder:

- The PDB files with chain IDs.

E.3 Multimers

Some of the proteins in this dataset are only biologically relevant as multimers. Multimers are protein complexes that are composed of multiple protein subunits, monomers, which can be identical or different, and are non-covalently bound. In the data collected for this project, none of the papers mentioned whether the protein for which they measured the redox potential is in its monomeric or multimeric form. The large majority of the proteins in my dataset exist as monomers, but some of them, UniProt states, interact as homodimers, simply writing "Homodimer" in the *Interaction* section of the protein's page. Cross-referencing this with PDB was also not helpful, as the "Biological assembly

1," which should be the biologically relevant structure, would not be differentiated from the asymmetric unit. They would either both be homodimers, or both monomers.
To address this confusion, it was decided that:

- If a protein complex binds one cofactor per subunit, and that subunit is modeled correctly in AlphaFold, then only that subunit with its cofactor will be used in the dataset.
- If a protein complex binds one cofactor per subunit, and the subunit is not accurately modeled by AlphaFold, then the crystal structure of the complex will be used in the dataset instead, if available.
- If a protein complex binds one cofactor in the whole complex, then the AlphaFold structure(s) for the subunit(s) will be multimerized before implanting the cofactor and using it in the dataset.

Multimer assembly One entry in particular, UniProt ID P0DUV7, exists as a homotrimer. Its UniProt page's *Interaction* section states that this protein "homotrimerizes to form a pseudohexamer with a large central pore, which is probably the binding site for the [4Fe-4S] center" [Pang et al., 2011]. To obtain this structure, I used ColabFold, a faster and more user-friendly platform for protein structure prediction using AlphaFold2 and AlphaFold-Multimer [Mirdita et al., 2022]. More specifically, I used the `AlphaFold2_mmseqs2` Google Colab notebook available at ColabFold's Github (`https://github.com/sokrypton/ColabFold`). In the query sequence, I pasted the complete protein sequence as found on UniProt, 3 times, separated by a colon each time. I named the job appropriately, and then clicked `Runtime` / `Run all`. No other settings were changed. The resulting homotrimer is shown in Figure E.2. We can see from the right structure that most of it is colored dark blue, indicating that AlphaFold is highly confident in its prediction, which validates our structure. From these images, we can clearly see the "large central pore," which UniProt mentioned is likely the binding side of the cofactor.

Crystal structures Throughout the data collection, if a big discrepancy was found between the AlphaFold structure and the available crystal or NMR structure, the latter would be used instead. This happened in a few cases, where UniProt stated the protein interacts as a dimer, and the monomer predicted by AlphaFold would look nothing like the experimentally found homodimer structure. The following protein entries had their AlphaFold structure replaced by a crystal or NMR structure in the final dataset: P9WF43 (NMR), P00260 (crystal), Q9NZ45 (crystal), and Q8N5K1 (crystal).

E.4 Cofactor Implantation

Once all the protein structure files are downloaded, prepped, and converted to PDB format, they are ready to be implanted with their corresponding cofactor. For mutant structures, this step is done after the energy minimization rather than before due to the complexity and lack of defined force fields for Iron-Sulfur clusters in AMBER, which would make minimizing them extremely difficult.

In this section, a detailed implementation of the pipeline used to insert cofactors into the protein structures is described.

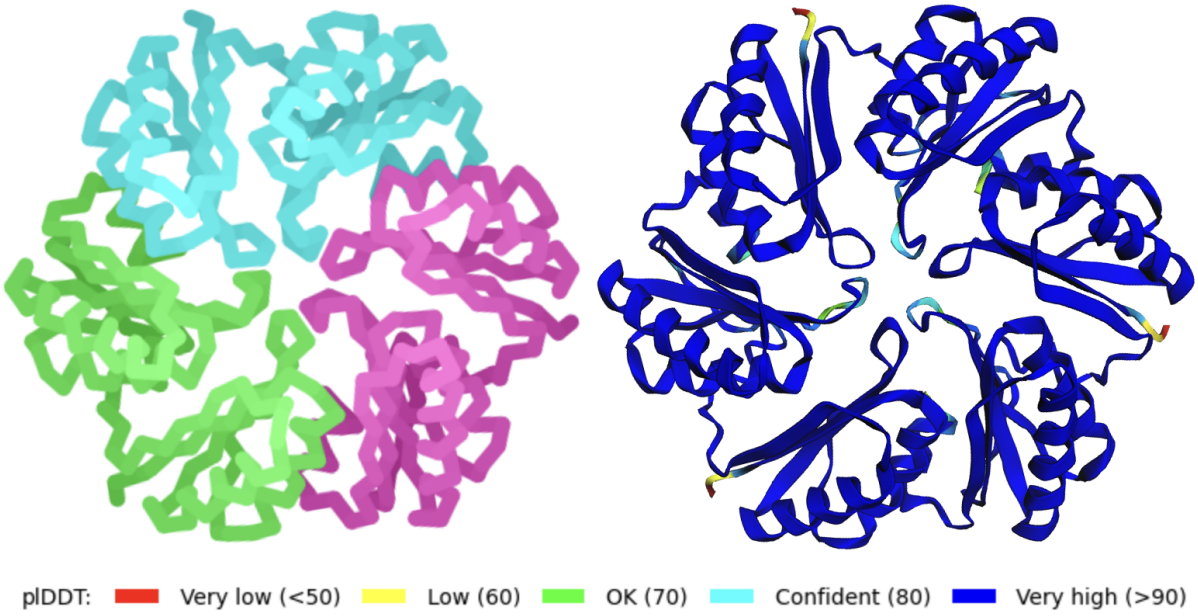

Figure E.2: Results of ColabFold's prediction on homotrimerizing entry P0DUV7. The left image is the structure, colored by chain, showing clearly the trimerization. The right image is colored by AlphaFold's prediction confidence level, according to the legend below the images.

E.4.1 AlphaFill transplants

All predicted proteins in the AlphaFold database lack coordinates for small molecules, ligands, and cofactors. The AlphaFill database is available at https://alphafill.eu/.

AlphaFill interface The AlphaFill database offers two options: "Show an entry based on Uniprot ID," where a UniProt ID can be entered in a field, and "Create your own AlphaFill model," which takes a structure PDB file as input. If structures with over 25% similarity in the sequence are found, AlphaFill returns the enriched structure. It is possible to look at results for 25%, 30%, 40%, 50%, 60%, and 70% identity. Of course, each time, only the structure at the highest identity possible (meaning, with results) is considered.

Downloading AlphaFilled structures As no API was found, I looked up each individual entry in my dataset manually. The entries that were not already present in the database were uploaded in the second field I mentioned. All output structures were downloaded into the same folder as *.cif* files, the default AlphaFill extension. However, not all of these structures were enriched with their corresponding ligands and cofactors: some were empty, while others had too many chemical compounds added.

(Overly) Enriched structures In the latter case, AlphaFill gives the option of selecting which compounds to keep in the final structure, so I would select just the compounds I knew are present in the protein, as stated in UniProt (usually just the Fe-S cofactor, but sometimes, also a Zinc ion or other small molecule), before downloading the structure. In the cases where only one compound is left on the structure, AlphaFill gives the option to "optimize" it using an energy minimization technique called YASARA [Hekkelman et al.,

2023]. If this optimization was available, and if it did not run into an error, I would download the optimized structure.

Empty structures Structures with no added compounds were downloaded with a "`_empty`" suffix in the file name. I downloaded all the structures, regardless of whether they were enriched by AlphaFill or not, to ensure consistency and completeness in my dataset.

To prepare these structures for the rest of the project, they needed to be converted from *.cif* to *.pdb* format. This is done using `gemmi`, a package which I downloaded into my `alphafill-env` virtual environment, and done via the Python script `convert_cif_to_pdb.py`. All structures were initially saved in a folder named `alphafill_cif`, and the converted files were saved into a new folder `alphafill_pdb`.

E.4.2 "Manual" transplants

After the initial search on AlphaFill, more than half the structures in my dataset were still empty. This meant another method of cofactor implantation was necessary. I called this part of my project the "manual" transplants, though the process is sometimes automated to transplant to multiple proteins at once. All transplants were done through Python scripts, using the PyMOL API. There are four phases to this part of the project:

1. Cofactor transplants into wild-type protein structures from their crystal or NMR structures (backbone alignment).
2. Cofactor transplants into mutant protein structures from their wild-type counterparts (backbone alignment).
3. Cofacor transplants into wild-type proteins from structurally similar protein donors (backbone and local alignment).
4. Cofactor transplant exceptions.

Transplants from crystal or NMR structures

For this phase of the cofactor transplants, I moved all the downloaded crystal or NMR structures to a folder called `crystal_donors`. I then converted them to PDB format from their native *.ent* format using the Python script `convert_ent_to_pdb.py`, and saved the converted files to a folder `crystal_donors_pdb`. The transplant itself is executed using the Python script `transplant_cofactors_from_crystal.py`. The donor file is `crystal_donors_pdb` and the target file is `alphafill_pdb`. This script parses the file names in each folder and tries to find a match for the target file name in the donor folder, if the target file has a "`_empty`" suffix. If a donor is found, it uses backbone alignment to line up both structures, then extracts the cofactor from the donor and implants it into the target, and saves the resulting structure. At every step, error handling logs if an element is not found by the code, and moves on to the next file in the folder. Since the donors concerned only wild-type proteins, the outputs were saved with a "`_transplanted`" suffix into a folder named `alphafilled_wild_types`. For completeness and for use in the next phase of this transplant step, the wild-type structures that had been correctly completed by AlphaFill were also manually copied into this folder.

Transplants from wild-type counterparts

Transplants into minimized mutant structures from their wild-type counterparts are done with the script `transplant_cofactors_mutants.py`, which uses similar logic as the `transplant_cofactors_from_crystal.py` script. For the donors, it takes the `alphafilled_wild_type` folder, and the target folder is the `input_structures` folder initially created for AlphaFill, containing the wild-type protein structures and the minimized mutant structures. The script parses the file names and looks specifically for those with "minimized" in the name. For each file found that fulfills this condition, it finds the corresponding wild-type donor in the donor folder and aligns the two structures via backbone alignment, then transplants the cofactor. The resulting structures are saved in a folder `alphafilled_mutants_manual` with naming convention `{UniProt ID}_{mutation}_transplanted.pdb`.

Transplants from structurally similar donors

After both previous steps were done, many protein structures still remained without a cofactor. For these, structurally similar protein donors had to be visually inferred to be their donors. This was done in PyMOL, where I would load the target protein, and then load a few proteins to try as donors. I would start with proteins from the same family, as those usually have similar structures. As such, all rubredoxins were filled by another rubredoxin (the first of which was filled by its crystal donor), all HiPIPs were filled by another HiPIP, and so on. The script used to execute the transplant itself is `manual_cofactor_transplant.py`, which takes a target and donor file as input, and has a setting `USE_GLOBAL_ALIGNMENT`. If set to `True`, the target and donor are aligned by their backbones. If set to `False`, it takes specific residues to align in both structures, for a local alignment. In a protein's UniProt page, the "Function" section has a "Features" subsection which lists the position (residue numbers) of the binding sites for the ligands and cofactors of the protein. When these were given, which was almost always the case, local alignment was always used. If they were not, then global alignment was used. The cofactors to extract are also passed to `COFACTOR_RESN` as the 3-letter identifiers used by the PDB. The setup for this script is shown in Listing E.1. It is important to note that when the donor structure has multiple of the same cofactor, if selected, they will all be copied into the target structure. So, for example, if a target just needs one SF4 cofactor, the donor also ideally only has one SF4 cofactor.

Listing E.1: Configuration and first lines of the `manual_cofactor_transplant.py` script

```
# === User Config ===
# Input files
DONOR_FILE = "donor_file_path"
TARGET_FILE = "target_file_path"
OUTPUT_FILE = "output_file_path"

# Toggle for alignment strategy
USE_LOCAL_ALIGNMENT = True  # Set to False to use backbone alignment

# Only used if USE_LOCAL_ALIGNMENT is True
DONOR_SELECTION = "resi 9+12+15+19"
TARGET_SELECTION = "resi 38+41+50+54"

# Cofactor residue names to extract
COFACTOR_RESN = ["SF4"]  # ["FES", "SF4", "FE", "FE2", "F3S"]

```

```
# Log file path
LOG_FILE = "CSV_log_file_path"

# === Start PyMOL
cmd.reinitialize()
cmd.load(DONOR_FILE, "donor")
cmd.load(TARGET_FILE, "target")

# For the CSV log file
alignment_type = "Local" if USE_LOCAL_ALIGNMENT else "Global"
```

Transplant exceptions

For some structures, it was not as straightforward to implant a cofactor. Namely, the homotrimerized entry P0DUV7 had no clear donor and required modeling and coloring of the coordinating residues in PyMOL (using the commands `show sticks` and `color`) to manually implant the cofactor directly in the PyMOL application and visually assert its position.

Another entry, P00209, was initially filled by its crystal structure with a [3Fe-4S] cofactor, but one paper had inserted an extra Iron atom into the cofactor to transform it into a [4Fe-4S] cofactor for which they calculated the redox potential [Moreno et al., 1994]. For this entry, the bare structure file had to be duplicated to then manually implant a [4Fe-4S] cofactor. The two files are named `P00209_F3S.pdb` and `P00209_SF4.pdb`.

Lastly, entry P18187_P238C's mutation caused the intermediate [3Fe-4S] cofactor to turn into a [4Fe-4S] cofactor [Rousset et al., 1998]. So, while the initial minimized structure had two [4Fe-4S] cofactors and one [3Fe-4S] cofactor implanted into it, like with the other mutated P18187 entries, this one had to be changed manually by deleting the [3Fe-4S] cofactor directly in the PyMOL application, and choosing a new donor for the [4Fe-4S] cofactor, implanting it via `manual_cofactor_transplant.py`.

E.4.3 Validation and cleanup

For every cofactor transplant, the PyMOL-calculated root mean square deviation (RMSD) was recorded in a separate log file corresponding to each transplant script. Every structure was verified visually in PyMOL to see if the correct and right number of cofactors were implanted in each structure. For structures where the transplant RMSD was larger than 2Å, if possible, I would show the coordinating residues on the structure as sticks and try to manually rearrange the cofactor's position to fit better.

Once all the protein structures are complete and validated, they are ready for the next step of the project, the feature extraction.

E.5 Feature Extraction

My feature extraction code was adapted from the methodology of Galuzzi et al (2022). My approach extends their framework to iron-sulfur proteins by incorporating iron-sulfur cluster-specific structural features while maintaining their core concept of radius-based local feature extraction on top of protein-wide feature extraction. For the local feature extraction, while Galuzzi et al.'s code calculates features both within a certain radius of the cofactor's barycenter and within a certain radius of each atom in the cofactor, my

code only looks at the former, since iron-sulfur clusters are a lot more densely packed and small than the FAD or FMN cofactors found in flavoproteins.

The features in this project were extracted for distances ranging from 1Åto 16Åfrom the cofactor barycenter, in steps of 1, which then yields 16 datasets per run of the feature extraction script. These distances were chosen for thoroughness–the Fe-S cofactors having a diameter of around 4Å, and residues beyond 12Å being unlikely to affect the cluster's redox potential [Terranova, 2024], this range of radii would encompass, with a safe margin, all possible relevant interactions.

The feature extraction is done via the script `extract_features.py`, which imports functions from the supporting scripts `utils.py` and `utils_structure.py`. All scripts are run within the `redox-env` virtual environment (see Table A.1).

The following subsections describe the feature extraction process in detail.

E.5.1 Feature Categories

Molecular descriptors considering only the region around the cofactor are subsequently referred to as "Bar region features" or simply "Bar features" and are prefixed in the extracted dataset with "`Bar.`", while those considering the whole protein are prefixed with "`Protein.`". These prefixes only apply to the first two of the four feature categories generated by the feature extraction pipeline, as described below:

Amino acid counts First, the amino acids within each region (Bar region and whole protein) are counted. These features are prefixed with "`.count`", and since there exist 20 amino acids, this step of the feature extraction yields **40 features** (20 counts for each region). The amino acids are written with their 3-letter codes, yielding feature names such as: `Bar.count.ALA` (number of alanines within the specified radius around the cofactor barycenter) and `Protein.count.VAL` (number of valines in the entire protein).

Properties and means Values of 28 different physicochemical properties for all amino acids (described in Table A.2) are then calculated for each region. This table, `tableAmm.txt`, is also taken from Galuzzi et al. The final dimensions of the table are 28×20, or rather 30×20 including the amino acid single-letter and 3-letter codes. The features prefixed by "`.prop`" are the sum of the property values for all the amino acids in the region considered, and the features prefixed by "`.mean`" are the property value divided by the total number of amino acids in that region. This extraction step yields 28×4 features, so **112 features**.
The final feature names are, for example: `Bar.mean.Hydrophobicity` (average hydrophobicity value in the Bar region considered) and `Protein.prop.P(helix)` (tendency of the amino acids in the entire protein to adopt a helix secondary structure).

Structure features The features referred to as "Structure features" include **4 features** whose names are prefixed by "`struct.`":

1. `burial_depth`: calculated as the minimum distance from the cluster barycenter to the protein surface, modeled as a convex hull using `SciPy`'s `ConvexHull` function, in Angstrom.
2. `iron_count`: the number of iron atoms in the cofactor.

3. `is_hipip`: a binary indicator of whether the protein is a HiPIP (1 if yes, 0 otherwise).
4. `is_rieske`: a binary indicator of whether the protein is a Rieske protein (1 if yes, 0 otherwise).

For the binary indicators to be assigned, two lists of Rieske and HiPIP UniProt IDs are passed to the feature extraction script.

Sequence features These features are prefixed by "`seq.`" and are generated using `PyBioMed`'s Composition-Transition-Distribution (`CTD`) module. This module calculates descriptors for seven physicochemical properties of amino acids, classifying each amino acid in one of 3 predefined groups, based on literature-derived scales. These properties, groups, and their descriptions can be found in Table A.3. Unlike Galuzzi et al., who only use the `CalculateC` (calculate composition) function that returns the fraction of amino acids in each group, I also employed the `CalculateT` (calculate transition) function. This function returns the normalized counts, or frequency, of the transition between amino acids of different groups along the sequence, for each property. Respectively, both these functions return 7 properties $\times$ 3 groups $= 21$ features, and 7 properties $\times$ 3 transitions $= 21$ features, so $2 \times 21 =$ **42 features** in total.
Examples of feature names in this category include: `seq.CTDT__HydrophobicityT12` (frequency of transitions between groups 1 and 2 qua hydrophobicity) and
`seq.CTDC__SolventAccessibilityC2` (fraction of amino acids in group 2 qua solvent accessibility).

All of these molecular descriptors together, plus the pH value at which the redox potential was measured, give **199 features** for each cofactor.

E.5.2 Feature Extraction Pipeline

The following paragraphs will detail the pipeline of the feature extraction script, `extract_features.py`.

All validated protein structures with their implanted cofactors were copied into a folder `final_structure_dataset`. I cleaned the file names using Python to remove all the suffixes, leaving the file names as `{UniProt ID}_{mutation, if any}.pdb`. The script used is `clean_structure_file_names.py`, which has two implementations: one where files with the same name are overwritten, and one without overwriting. This folder serves as the input for the feature extraction script.
The script then, for each radius listed:

1. Iterates over all the structures in the folder; for each structure, it calls the `get_all_FeS_clusters` function from `utils_structure.py` to identify all the Fe-S clusters in the protein.
2. Calculates the burial depth of the cofactors found.
    - In the case that there are two [4Fe-4S] cofactors, it labels them either "proximal" or "distal" according to burial depth–the proximal cluster being more buried than the distal one.
    - If there are two [4Fe-4S] cofactors and one [3Fe-4S] cofactor, the same convention is followed, only the [3Fe-4S] cofactor is then labeled "intermediate."

- It handles the special case of entry P18187_P238C, where the [3Fe-4S] cofactor was turned into a [4Fe-4S] cofactor by the mutation, by measuring the pair-wise distance between the 3 cofactors, and labeling the middle one "intermediate."

3. Labels all the cofactors found as `{UniProt ID}_{mutation, if any}_{Cofactor PDB code}_{label (proximal, distal, intermediate), if any}`. This is the cofactor ID.
4. For each cofactor ID, runs the `extract_features` function, which:
   (a) Parses the structure PDB file to extract the sequence.
   (b) Extracts CTD features.
   (c) Counts amino acids.
   (d) Computes properties and their means according to `TableAmm.txt`.
   (e) Adds the structural features.
5. Returns the feature CSV files, named `features_r{radius}_final_{datetime stamp}.csv`, as well as log files.
6. Returns metadata files for each radius with columns `structure_id`, `cofactor_id`, `cofactor_type`, `burial_depth`, and `radius`.

Before the pH and redox potential can be added to the features, one of the metadata files is used to add a `cofactor_id` column to the Excel data sheet, by joining the "UniProt" and "AF_mutation" columns in the latter with the `structure_id` column in the former. The folder containing the feature files and the updated Excel sheet are then passed to the script `merge_with_ph_em.py`, which matches the `cofactor_id` column in the feature files to the one in the Excel sheet to enrich the feature files with the redox potential and pH values. These files are returned in a separate folder, with a `_with_ph_em` suffix.

E.5.3 Dataset Division

To test the impact of different combinations of cofactors and features in the dataset, it was decided to divide the feature datasets into subsets, using the create_feature_subsets.py script. Separate datasets were then created for the 3 cofactor groups: all cofactors, only [4Fe-4S], only [2Fe-2S]. These groups were chosen since the [4Fe-4S] and [2Fe-2S] cofactors represent the largest subsets of the collected data. Then, for each cofactor group, 3 feature sets were created, containing: all features, only bar features, and only protein features. The structure features (with "`struct.`" prefix) and the pH and Em columns were kept in all the data subsets. The subset CSV files are named: `features_{radius, if relevant}_{cofactor (sub)set}_{feature (sub)set}.csv`. The resulting directory structure is shown in Section A.4.

Dividing the feature files into different subsets marks the last stage of the feature extraction phase, and the data is now ready to be run through the Machine Learning training pipeline, the implementation of which will be described in detail in the following section.

E.6 Model Training Pipeline

This section describes the methodology employed for training machine learning models to predict the redox potential of iron-sulfur proteins, based on the computational framework

adapted from Galuzzi et al. for flavoproteins [Galuzzi et al., 2022]. The training pipeline, implemented in Python, integrates data preprocessing, nested cross-validation, model training, performance evaluation, and feature importance analysis. The pipeline was designed to systematically evaluate multiple machine learning models across a range of spatial radii (1 Å to 16 Å) around iron-sulfur cofactors, capturing microenvironmental features influencing redox potential. The pipeline was implemented using the Scikit-learn library [Pedregosa et al., 2011], with additional feature importance analysis conducted using SHAP (SHapley Additive exPlanations) [Lundberg and Lee, 2017] when available. The following subsections detail each component of the pipeline.

E.6.1 Data Preprocessing

The input features are loaded from CSV files containing the extracted molecular descriptors and the target variable (redox potential, E_m, in millivolts). Preprocessing steps included:

- **Feature Selection**: Non-numeric columns are excluded, and features with zero variance across samples are removed using `VarianceThreshold`. Highly correlated features (Pearson correlation > 0.95) are also eliminated to reduce redundancy.
- **Missing Value Imputation**: Missing values are filled with the median of the respective feature.
- **Scaling**: For models requiring normalized inputs, features are standardized using `StandardScaler` to ensure zero mean and unit variance.

A crucial detail is that the preprocessing is performed within each cross-validation fold to prevent data leakage, ensuring parameters are derived solely from training data.

E.6.2 Nested Cross-Validation (CV)

A nested cross-validation strategy is used for hyperparameter optimization and performance evaluation. It is repeated 10 times with varying random seeds to assess generalization performance. The outer loop employs 5-fold KFold cross-validation, while the inner loop uses 3-fold KFold cross-validation and `GridSearchCV` to try all hyperparameter combinations and optimize them based on *negative* mean absolute error (MAE), since lower MAE is better. Since there is one new model instantiated and trained every outer CV fold, and the entire nested CV is repeated 10 times, the final model performance metrics are the average over $5 \times 10 = 50$ models. This structure ensures unbiased performance estimates by isolating hyperparameter tuning from final evaluation.

For feature data containing radius-specific features, this nested CV is performed for each combination of radius and model, using the ML training script `ml_training.py`. For the 3 feature data CSV files that are radius-independent (all cofactors, only SF4, and only FES "protein features only" data), this nested CV is performed for each model-dataset combination, in the `ml_training_protein.py` script.

E.6.3 Model Selection and Training

Seven machine learning models are trained, as mentioned in section 2.5: Linear Regression, Elastic Net, Support Vector Regression (SVR), Gaussian Process Regression (GPR), K-Nearest Neighbors (KNN), Random Forest, and Extreme Gradient Boosting(XGB).

Each model is integrated into a `Scikit-learn` Pipeline, incorporating preprocessing steps (scaling and feature selection) as needed. Table E.3 summarizes the models, their hyperparameter grids, and preprocessing requirements.

Feature selection is implemented within each outer CV fold to avoid overfitting by passing too many features to the models. For models needing feature selection (Linear Regression, Elastic Net, SVR, GPR, KNN), f-regression scores are used to measure the linear dependency between each feature and the target variable, in the end selecting only the top $\min(50, n/2)$ features, where n is the number of features in the training set. This means that for all training sets for these models needing feature selection, no more than 50 features are kept. The other two models, Random Forest and XGB, do not need feature selection as they already have built-in feature selection mechanisms through their training process, and are generally more robust to irrelevant features.

The best model for each radius/dataset is saved as a pickle file for further analysis after all iterations of the nested CV. Then, SHAP values are computed and graphs are generated for the top 3 models (for radius-independent datasets) or the top 3 model-radius combination (for the radius-dependent datasets). These values are subsequently analyzed using the script `find_recurrent_shap_features.py` as described in Subsection E.8.6.

E.7 Performance Evaluation and Interpretation

Model performance is assessed using multiple metrics: Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), R^2 score, Spearman and Pearson correlations, and Explained Variance. These metrics are computed for each fold and repeat, with means and standard deviations reported to quantify performance stability. Learning curves and prediction analysis plots (scatter, error distribution, and residual plots) are generated for each model and radius to visualize training dynamics and prediction quality.

For feature importance, SHAP analysis is performed on the top 3 performing models (based on MAE) across all radii. SHAP values are calculated using `TreeExplainer` for tree-based models (Random Forest, Gradient Boosting) and `KernelExplainer` for the others, analyzing feature contributions separately for protein-wide, local (Bar region), and all features. Summary plots and feature importance rankings are saved for the top 20 features per group.

E.8 Results Analysis and Visualization

The analysis and visualization of the iron-sulfur protein dataset and its machine learning (ML) predictions are conducted using a suite of custom Python scripts, tailored to handle both radius-dependent and radius-independent data. This section outlines the methodologies employed to process, analyze, and visualize the results, ensuring a comprehensive evaluation of dataset characteristics and model performance.

E.8.1 Dataset Statistical Analysis

Dataset statistics are computed using the `dataset_statistics.py` script, which analyzes structural and experimental data from PDB files, feature CSV files, and metadata Excel files. The script calculates protein size metrics (residue counts, 3D dimensions, and convex hull volumes) using the BioPython PDB parser. Cofactor type distributions, protein

family proportions, burial depths, redox potential statistics, pH values, mutation frequencies, and protein coverage are also plotted in various forms. Statistical measures (mean, median, standard deviation) are calculated , and visualizations (histograms, boxplots, and pie charts) are generated using Matplotlib and Seaborn, saved as high-resolution PNG files in a designated output directory. Detailed statistics are exported as CSV and JSON files for further analysis.

E.8.2 ML Training Result Interpretation

Dedicated scripts, `interpretation_protein.py` and `interpretation.py`, were developed to interpret and visualize ML training results from both radius-independent (processed with `ml_training_protein.py`) and radius-dependent (processed with `ml_training.py`) datasets, respectively. The script takes as input the complete output folder from the ML training script, loads the results from `ml_results.json`, and performs the following analyses:

- **Top Model Identification**: The top five models are ranked by mean absolute error (MAE), with performance metrics (MAE, R^2, and standard deviations) reported for each radius or overall, depending on the dataset type.
- **Model Stability Analysis**: Stability is evaluated for radius-dependent data by computing cross-validation standard deviation and approximate prediction ranges across radii, visualized in dual-panel plots using `Matplotlib`.
- **Comprehensive Visualization**: A four-panel plot is created for radius-dependent data, displaying MAE and R^2 versus radius, optimal radius per model, and MAE distribution across all radii, excluding LinearRegression to avoid skewing color scales.
- **Performance Comparison**: Model performance on the radius-independent data is visualized for MAE and R^2 with side-by-side bar charts, with and without Linear Regression to avoid skewing scales.
- **Literature Comparison**: Model performance is benchmarked against a literature MAE of 36 mV for flavoproteins [Galuzzi et al., 2022], calculating percentage improvement.

A text-based report (`analysis_report.txt`) is generated, summarizing top models and performance per radius. Results are stored in an output directory named after the input directory and "`_interpretation`" suffix, as PNG visualizations.

E.8.3 Mann-Whitney Tests

Mann-Whitney U rank tests were performed using the `mann_whitney_tests.py` script, after copying all ML result CSV files, `ml_output_summary.csv`, from each dataset into an `input` directory, with naming convention `{dataset}_ml_output_summary.csv` (e.g., `complete_dataset_ml_output_summary.csv` and `FES_bar_ml_output_summary.csv`). Results are saved in a .txt file, `mann_whitney_results.txt`) in the input folder.

E.8.4 Comparative Heatmap Generation

Comparative heatmaps are generated using `compare_datasets_heatmaps.py` for radius-independent data and `generate_heatmaps_complete.py` for radius-dependent data, addressing limitations in the ML training scripts' visualizations (e.g., LinearRegression skew-

ing color scales and lack of model-specific heatmaps). For radius-independent data, `compare_datasets_heatmaps.py` creates heatmaps with dataset types (All cofactors, FES, SF4) on the x-axis and models on the y-axis, using MAE, RMSE, and R^2 metrics. For radius-dependent data, `generate_heatmaps_complete.py` produces dataset-specific heatmaps (radii on x-axis, models on y-axis) and model-specific heatmaps (datasets on y-axis, radii on x-axis), excluding LinearRegression from color range calculations to ensure consistency across all radii. Visualizations are saved as PNG files in designated output directories, leveraging Seaborn and Matplotlib for heatmap rendering.

E.8.5 Aggregated Results Analysis

To consolidate and compare ML results across multiple datasets, two dedicated scripts were developed: `aggregate_radius_dependent.py` for radius-dependent data and `aggregate_radius_` for radius-independent data. These scripts aggregate results from multiple output folders, each containing `ml_results.json` files, and generate comprehensive visualizations and reports.

For radius-dependent data, `aggregate_radius_dependent.py` processes results from six datasets: `complete_dataset` (all cofactors with both radius-dependent and radius-independent features), `all_cofactors_bar` (all cofactors, radius-dependent only), `FES_all`, `SF4_all`, `FES_bar`, and `SF4_bar`. The script performs the following:

- **Data Aggregation**: Collects performance metrics (MAE, R^2, and standard deviations) across all datasets and radii, excluding LinearRegression to avoid skewing analyses.
- **Stability and Trend Analysis**: Computes mean MAE standard deviations per model and radius for stability, and performs linear regression on MAE versus radius to assess performance trends, reporting average slopes and p-values.
- **Visualizations**: Generates stability heatmaps (with and without LinearRegression), MAE distribution collages in 2x3 (with `complete_dataset`, `FES_all`, `SF4_all` in the first row; `all_cofactors_bar`, `FES_bar`, `SF4_bar` in the second row) and 3x2 layouts, and performance trend collages (MAE and R^2 versus radius) in 2x3 and 3x2 layouts for report and appendix datasets. A separate legend image is saved for model color mapping.
- **Reporting**: Outputs a text report (`aggregated_report.txt`) summarizing optimal model performance per dataset and radius, stability metrics, and trend analyses.

Visualizations are saved as high-resolution PNG files in an `aggregated_interpretation` directory, with a log file (`aggregation_log.txt`) capturing processing details.

For radius-independent data, `aggregate_radius_independent.py` aggregates results from datasets such as `all_cofactors_protein`, `FES_protein`, and `SF4_protein`, which use only radius-independent features. The script performs the following:

- **Data Aggregation**: Collects performance metrics (MAE, R^2, and standard deviations) from `ml_results.json` files across all relevant datasets.
- **Top Model Selection**: Identifies the top five models by MAE, reporting performance metrics for each dataset.
- **Reporting**: Generates a text report (`aggregated_report.txt`) summarizing the top models and their performance, saved in an `aggregated_interpretation` directory, with a log file (`aggregation_log.txt`) for processing details.

Both scripts utilize Matplotlib and Seaborn for visualization and Pandas for data manipulation, ensuring consistent output formats and compatibility with the ML training pipeline.

E.8.6 SHAP Graph Analyses

To elucidate the key molecular and structural features influencing the performance of machine learning models for different datasets, SHAP (SHapley Additive exPlanations) values were computed to quantify feature importance for the top 3 models or model-radius combinations, selected based on their predictive performance measured by MAE, for each dataset subsets. The ML training scripts output these SHAP values as CSV files with two columns (features, SHAP importance score) and as graphs. Since there are 9 dataset subsets and 3 graphs for each, 27 graphs are generated for the entire dataset.

A Python script, `generate_shap_table.py`, was developed to automate the identification of recurrent features and generate a structured output. The script, implemented using the `pandas` library, processes the ML output SHAP CSV files for one specific dataset subset, stored in a directory. The script extracts the top N features (default $N = 5$) based on SHAP importance values from each of the CSV files. Recurrent features are defined as those appearing in at least M combinations (default $M = 3$), with both N and M configurable to allow flexibility in analysis (e.g., top 10 features or recurrence in at least 2 combinations). The script outputs a CSV file in the `output/` directory, with columns for Feature, Description, and binary indicators (or) for recurrence in all the CSV files. Feature descriptions are maintained in a dictionary within the script, mapping each feature to its physicochemical or structural significance (e.g., `seq.CTDC__HydrophobicityC3` as "Fraction of amino acids in hydrophobicity group 3 (hydrophobic)").

I ran this script 9 times to obtain all my data for the tables in Section 4.6. I focused on the top 5 features in all the SHAP CSV files, and filtered out those appearing in at least 2 of the 3 files. I do this for the subsets of all features, only bar features, and only protein features, for each of the cofactor subsets. So, the features that are filtered out by the script are added to a table (one table per cofactor subset), and a tick in the feature subset column marks whether that feature was recurrent in the feature subset dataset.

E.8.7 Results Availability

Results are systematically stored in structured output directories. Training outputs include JSON files (`ml_results.json`), CSV summaries (`ml_results_summary.csv`), and serialized best models as pickle files. Grid search parameters are saved as JSON (`grid_search_parameters.json`, `best_parameters_summary.json`). Visualizations encompass performance plots, SHAP summaries, feature importance evolution, model stability analyses, comprehensive comparisons, heatmaps, and aggregated collages, all saved as high-resolution PNG files. Analysis and aggregation reports are saved as `analysis_report.txt` and `aggregated_report.txt`, ensuring accessibility for further review.

E.9 Supercomputer Usage

All the ML training scripts were run using DelftBlue, the TU Delft's supercomputer. I first recreated my `redox-env` virtual environment on DelftBlue, according to DelftBlue's

documentation (https://doc.dhpc.tudelft.nl/delftblue/howtos/conda/), and using the Windows/Linux-adapted `portable_environment.yml` file I generated from my `redox-env`. The resources I asked of DelftBlue are shown in Listing E.2. Running the ML training loop on all scripts took about 23 hours.

Listing E.2: Computational Resources Requested From DelftBlue

```
--partition=compute
--ntasks-per-node=1
--cpus-per-task=16
--mem-per-cpu=3G
--time=24:00:00
```

E.10 Code Availability

The custom Python scripts developed for this study are available to support reproducibility and transparency. These scripts implement the statistical analysis, ML training result interpretation, and heatmap generation described in Sections E.8. The code is hosted on a public GitHub repository at https://github.com/yourusername/iron-sulfur-ml-analysis, under an open-source license (e.g., MIT License). Detailed documentation, including dependencies (e.g., BioPython, Matplotlib, Seaborn, scikit-learn), accompanies the repository. For any updates or issues, interested parties may contact the corresponding author at your.email@example.com.

Model	**Scaling**	**Feature Selection**	**Hyperparameters**
Linear Regression	Yes	Yes	None
Elastic Net	Yes	Yes	$\alpha \in \{0.01, 0.1, 1.0, 10.0, 100.0\}$, $l1\_ratio \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$
SVR	Yes	Yes	$C \in \{0.1, 1, 10\}$, $\gamma \in \{\text{scale}, \text{auto}, 0.001, 0.01, 0.1, 1\}$, $\epsilon \in \{0.01, 0.1, 0.2\}$, kernel = rbf
GPR	Yes	Yes	kernel $\in \{\text{Constant} \times \text{RBF} + \text{White}, \text{Constant} \times \text{RBF}, \text{RBF} + \text{White}\}$, $\alpha \in \{10^{-10}, 10^{-8}, 10^{-6}, 10^{-4}\}$
KNN	Yes	Yes	$n\_neighbors \in \{3, 5, 7, 9, 11, 15\}$, weights $\in \{\text{uniform}, \text{distance}\}$, metric $\in \{\text{euclidean}, \text{manhattan}, \text{minkowski}\}$
Random Forest	No	No	$n\_estimators \in \{100, 200\}$, $max\_depth \in \{5, 7, 10\}$, $min\_samples\_split \in \{2, 5, 10\}$, $min\_samples\_leaf \in \{1, 2, 4\}$, $max\_features \in \{\text{sqrt}, \text{log2}\}$
XGB	No	No	$n\_estimators \in \{100, 200\}$, $learning\_rate \in \{0.05, 0.1\}$, $max\_depth \in \{3, 5\}$, $min\_samples\_split \in \{2, 5\}$, $min\_samples\_leaf \in \{1, 2, 4\}$, $subsample \in \{0.8, 1.0\}$

Table E.3: **Machine learning models and their configurations used in the redox potential prediction pipeline**. α in Elastic Net and GPR controls regularization strength, penalizing large coefficients to prevent overfitting (higher values increase penalty). `kernel` in SVR and GPR defines the transformation (e.g., 'rbf' [radial basis function] for nonlinear patterns, 'Constant × RBF' for Gaussian processes). γ in SVR adjusts the influence radius of training points (smaller values broaden it). ϵ in SVR sets the margin of tolerance for errors. `n_neighbors` in KNN specifies the number of nearest points considered, with `weights` ('uniform' or 'distance') and `metric` ('euclidean', 'manhattan', 'minkowski') affecting distance weighting. `max_depth` and `min_samples_split` in Random Forest and XGB limit tree depth and node splits to control complexity, while `subsample` in XGB fractions the training data per tree.