



UNIVERSITY COLLEGE LONDON

**Predicting and Characterising  
Zinc Metal Binding Sites in  
Proteins**

Sam M. Ireland

A thesis submitted to University College London  
for the degree of Doctor of Philosophy

September 2021

# Declaration of Authorship

I, SAM IRELAND, declare that this thesis titled, ‘Predicting and Characterising Zinc Metal Binding Sites in Proteins’ and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

---

Date:

---

# Impact Statement

This PhD has resulted in the creation of a dedicated database of zinc binding sites (ZincBindDB) and tools for easily accessing it, predictive models for predicting zinc binding sites in proteins (ZincBindPredict), and a novel Python library for analysing protein structures (atomium).

There was previously no such resource for collating zinc binding sites, and so the introduction of one facilitates research into zinc binding sites generally by giving researchers in this field a convenient, centralised repository of the sum of all knowledge of zinc binding sites for which there is structural information — where no such centralised repository existed before. This underpins and supports research into a diverse area of biology. The codebase is also open source and highly extensible, and could be expanded to support other metals as required.

Zinc binding is a property of about 10% of proteins, and is particularly important in certain enzyme reactions. Knowing whether a protein binds to zinc can offer insights into its function, and knowing precisely where it binds zinc can show the mechanism by which it carries out its intended function, as well as provide suggestions as to how pharmaceutical molecules might disrupt or enhance this function where required for medical interventions. The tools developed here make it easier to computationally assess whether a protein binds zinc, and where it does if so - aiding these endeavours.

The newly developed Python library atomium makes studying and analysing protein structures easier for Python programmers, specifically giving them the ability to study higher order assemblies of protein chains called biological assemblies, which was not possible with the existing BioPython library. This makes structural biology research more efficient.

# *Abstract*

Zinc is one of the most important biologically active metals. Ten per cent of the human genome is thought to encode a zinc binding protein and its uses encompass catalysis, structural stability, gene expression and immunity. Knowing whether a protein binds to zinc can offer insights into its function, and knowing precisely where it binds zinc can show the mechanism by which it carries out its intended function, as well as provide suggestions as to how pharmaceutical molecules might disrupt or enhance this function where required for medical interventions. At present, there is no specific resource devoted to identifying and presenting all currently known zinc binding sites. This PhD has resulted in the creation of ZincBind — a database of zinc binding sites (ZincBindDB), predictive models of zinc binding at the family level (ZincBindPredict) and a user-friendly, modern website frontend (ZincBindWeb). Both ZincBindDB and ZincBindPredict are also available as GraphQL APIs. The database of zinc binding sites currently contains 38,141 sites, and is automatically updated every week. The predictive models, trained using the Random Forest Machine Learning algorithm, all achieve an MCC  $\geq 0.88$ , recall  $\geq 0.93$  and precision  $\geq 0.91$  for the structural models (mean MCC = 0.97), while the sequence models have MCC  $\geq 0.64$ , recall  $\geq 0.80$  and precision  $\geq 0.83$  (mean MCC = 0.87), outperforming competing, previous predictive models.

# *Acknowledgements*

I am enormously grateful to the Wellcome Trust who, in addition to funding the entirety of this PhD and its associated costs, provided considerable assistance and understanding during the COVID pandemic with their prompt, targeted response.

I am greatly indebted to my supervisor, Professor Andrew Martin. Enormously helpful with questions, feedback and guidance at every stage of this project, while still granting me almost complete oversight of the direction and management of the project itself — I could not have asked for a friendlier, reliable and domain-knowledgable primary supervisor.

Likewise I am grateful to my second supervisor, Professor Stephen Perkins, and my thesis chair, Professor Christine Orengo, whose feedback and contributions at our many meetings were helpful to the eventual success of the project.

Prior to the start of the project I completed three rotations as part of this programme, and the help, guidance, and above all welcoming nature of those three supervisors — Professor Stephen Perkins (again), Professor Bonnie Wallace, and Professor Francesco Gervasio, to whom I partly owe the relative ease with which I was able to settle into PhD life — should be noted.

Indeed I would particularly like to single out the Postdoc Dr. Altin Sula, of Professor Bonnie Wallace's lab, whose patience, knowledge, mentorship and generosity of time was invaluable in making that second rotation the considerable success that it was — particularly given my relative lack of wet lab experience going into it.

More generally I am grateful to my fellow PhD Students and other colleagues at UCL and the Institute of Structural and Molecular Biology for their help, insights and comradeship.

Finally, I am indebted to the countless researchers and technicians who, over the past five decades, have painstakingly populated the Protein Data Bank with structures. Each of the tens of thousands of structures which a computational biologist such as myself considers for a fraction of a second was the work of months of highly skilled labour by teams of people who I will never meet or know, but by whose endeavours this project and the many like it are made possible. Thanks.

# Contents

<b>Declaration of Authorship</b>	<b>i</b>
<b>Impact Statement</b>	<b>ii</b>
<b>Abstract</b>	<b>iii</b>
<b>Acknowledgements</b>	<b>iv</b>
<b>List of Figures</b>	<b>viii</b>
<b>List of Tables</b>	<b>x</b>
<b>Abbreviations</b>	<b>xi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Zinc . . . . .	2
1.1.1 Cofactors . . . . .	2
1.1.2 Transition Metals . . . . .	5
1.1.3 Zinc's Unique Properties . . . . .	7
1.2 Properties of Zinc Binding Sites . . . . .	8
1.3 How do Zinc Binding Sites Work? . . . . .	14
1.4 The Protein Data Bank . . . . .	18
1.5 Conclusion . . . . .	21
<b>2 Computational Techniques</b>	<b>24</b>
2.1 Machine Learning . . . . .	24
2.1.1 Principles . . . . .	24
2.1.2 Supervised Learning Algorithms . . . . .	26
2.1.3 Evaluation . . . . .	32
2.2 Machine Learning to Predict Zinc Binding . . . . .	36
2.2.1 Prediction from Structure . . . . .	37
2.2.2 Prediction from Sequence . . . . .	42
2.3 Sequence Techniques . . . . .	46
2.4 Web Technologies . . . . .	47

---

2.4.1	Django . . . . .	47
2.4.2	GraphQL . . . . .	47
2.4.3	React . . . . .	48
2.5	Conclusion . . . . .	48
<b>3</b>	<b>atomium — The Python Structure Parser</b>	<b>50</b>
3.1	Rationale . . . . .	51
3.2	Data Structures . . . . .	52
3.2.1	Distance Optimisation . . . . .	56
3.3	Parsing . . . . .	57
3.3.1	File Contents and File Type Detection . . . . .	58
3.3.2	File Dictionaries . . . . .	58
3.3.3	Data Dictionary . . . . .	60
3.3.4	Model Generation . . . . .	61
3.4	Saving . . . . .	61
3.5	pdb2json . . . . .	62
<b>4</b>	<b>ZincBind — The Database of Zinc Binding Sites</b>	<b>65</b>
4.1	Data Generation . . . . .	66
4.1.1	Structure Inspection . . . . .	66
4.1.2	Equivalent Sites and Families . . . . .	71
4.2	The ZincBind Web Resource . . . . .	72
4.3	Data Analysis . . . . .	80
4.3.1	The Prevalence of Zinc . . . . .	80
4.3.2	Qualifying Atoms . . . . .	80
4.3.3	Zinc Binding Sites with High Representation . . . . .	82
4.3.4	Liganding Residues . . . . .	83
4.3.5	Co-active Binding Sites . . . . .	85
4.4	Conclusion . . . . .	86
<b>5</b>	<b>Predicting Zinc Binding Sites</b>	<b>88</b>
5.1	Approach . . . . .	88
5.2	Data Preparation . . . . .	90
5.2.1	Sequence Training Data . . . . .	91
5.2.2	Structure Training Data . . . . .	92
5.3	Model Training . . . . .	93
5.4	Model Performance . . . . .	95
5.5	Access to Models . . . . .	102
5.6	Conclusion . . . . .	103
<b>6</b>	<b>Conclusion</b>	<b>106</b>
6.1	Next Steps . . . . .	108
<b>A</b>	<b>Publications and Talks</b>	<b>111</b>
A.1	Published Papers . . . . .	111

---

A.1.1	ZincBind—the database of zinc binding sites . . . . .	111
A.1.2	atomium—a Python structure parser . . . . .	112
A.1.3	Zinbindpredict—Prediction of Zinc Binding Sites in Proteins	112
A.2	Talks and Presentations . . . . .	113
<b>B</b>	<b>atomium Documentation</b>	<b>115</b>
B.1	Loading Data . . . . .	115
B.2	Using Data . . . . .	116
B.2.1	Annotation . . . . .	116
B.2.2	Models and Assembly . . . . .	116
B.2.3	Model Contents . . . . .	118
B.3	Saving Data . . . . .	121
	<b>Bibliography</b>	<b>123</b>



# List of Figures

1.1	Stylistic representation of atomic orbitals . . . . .	4
1.2	Monodentate and bidentate binding . . . . .	17
2.1	Overview of the branches of machine learning. . . . .	25
2.2	Support Vector Machines . . . . .	28
2.3	Area Under Curve . . . . .	35
3.1	atomium classes. . . . .	55
3.2	atomium parsing. . . . .	59
3.3	atomium parsing. . . . .	62
4.1	The ZincBind build process. . . . .	70
4.2	An example of a ZincBind GraphQL API query. . . . .	73
4.3	The home page of ZincBind. . . . .	73
4.4	The advanced search interface. . . . .	74
4.5	The BLAST search interface. . . . .	75
4.6	An example of BLAST search results. . . . .	76
4.7	An example of paginated search results. . . . .	76
4.8	An example of a PDB page in ZincBind, showing basic information for that PDB. The three-dimensional structure view is further down the page. . . . .	77
4.9	An example of a binding site page in ZincBind. . . . .	78
4.10	Overview statistics for the dataset presented on the ZincBind data page. . . . .	79
4.11	A ZincBind group page, summarising the clustered zinc binding sites and providing links to individual site pages. . . . .	79
4.12	The relative frequency of omission reasons for zinc atoms that were ultimately not assigned to a binding site. . . . .	82
4.13	The distribution of binding sites among groups for the hundred most populated groups. . . . .	83
4.14	Correlation between residue hydrophobicity and their frequency as a secondary residue in zinc binding sites . . . . .	84
4.15	The distribution of liganding atom distances in all single-zinc sites. . . . .	86
5.1	Model Performance (MCC) as a function of training set size. . . . .	97
5.2	Learning curves for all 20 models. . . . .	98
5.3	MCC as a function of dataset size for 160 different models. . . . .	100

---

5.4	ZincBindPredict mutation request. . . . .	103
5.5	ZincBindPredict query request. . . . .	104
5.6	The prediction page on the ZincBind web interface. . . . .	104

# List of Tables

3.1	Examples of the filtering syntax that all atomium structures have by virtue of implementing the <code>StructureClass</code> metaclass. . . . .	55
4.1	Prevalence of various non-zinc metals in <code>ZincBind</code> . . . . .	81
5.1	Training set size. . . . .	93
5.2	Feature generation. . . . .	94
5.3	Structural model performance. . . . .	96
5.4	Sequence model performance . . . . .	97
5.5	Model Performance on datasets clustered at different similarity thresholds. . . . .	99
5.6	Predictive ability of using BLAST alone to predict zinc binding in protein sequences using homology alone. . . . .	101
5.7	Percentage of genome predicted to be zinc binding by <code>ZincBindPredict</code> for an assortment of bacterial genomes. . . . .	102

# Abbreviations

<b>ANN</b>	<b>A</b> rtificial <b>N</b> eural <b>N</b> etwork
<b>API</b>	<b>A</b> pplication <b>P</b> rogramming <b>I</b> nterface
<b>CIF</b>	<b>C</b> ystallographic <b>I</b> nformation <b>F</b> ile
<b>FN</b>	<b>F</b> alse <b>N</b> egative
<b>FP</b>	<b>F</b> alse <b>P</b> ositive
<b>JSON</b>	<b>J</b> ava <b>S</b> cript <b>O</b> bject <b>N</b> otation
<b>MCC</b>	<b>M</b> atthews <b>C</b> orrelation <b>C</b> oefficient
<b>mmCIF</b>	<b>M</b> acro <b>M</b> olecular <b>C</b> ystallographic <b>I</b> nformation <b>F</b> ile
<b>MMTF</b>	<b>M</b> acro <b>M</b> olecular <b>T</b> ransmission <b>F</b> ormat
<b>PDB</b>	<b>P</b> rotein <b>D</b> ata <b>B</b> ank
<b>RF</b>	<b>R</b> andom <b>F</b> orest
<b>SVM</b>	<b>S</b> upport <b>V</b> ector <b>M</b> achine
<b>TN</b>	<b>T</b> rue <b>N</b> egative
<b>TP</b>	<b>T</b> rue <b>P</b> ositive
<b>ZBS</b>	<b>Z</b> inc <b>B</b> inding <b>S</b> ite

# Chapter 1

## Introduction

Proteins are polymer chains of amino acid residues, and while the chemical diversity of the twenty canonical amino acids utilised in Biology can offer proteins a staggering variety of folds and functionality, there remain chemical processes that are not possible using only this chemical species. Consequently, many proteins use cofactors — chemicals which associate with the protein but are not generally covalently bound to them.

In order for a protein to induce this association to happen, it must present a region of its surface to which the cofactor in question will experience an attraction, and for which association will be thermodynamically favourable. The residues that make up this attractive region are called binding sites.

Metal atoms are a common cofactor. In this case the binding site is a metal binding site, and when the metal is zinc, it is a ‘zinc binding site’.

Clearly, zinc will only experience an attraction towards certain kinds of protein surfaces, and so proteins which need to attract a zinc cofactor will have to present surface residues capable of doing this. It should be possible to predict, therefore, whether a protein’s structure has a potential zinc binding site on it because only certain residues in certain geometric configurations will be capable of creating this high affinity for zinc. Furthermore, because structure is determined by amino acid sequence, it should in principle be possible to predict zinc binding capabilities from protein sequence.

This PhD project is an attempt to develop novel methods for doing so. Being able to predict whether a protein binds zinc from its sequence alone offers valuable insights into the function of the resultant protein, and allows entire genomes to be quickly searched for potential zinc binding proteins. Being able to predict zinc binding in a protein structure can reveal mechanistic information about how it carries out its function, as well as offering insights into how pathological zinc binding associated with that protein might be addressed. While there are previous methods for doing both of these things, this PhD project uses an entirely novel approach for doing so, backed by a comprehensive survey of all known zinc binding sites.

## **1.1 Zinc**

### **1.1.1 Cofactors**

Proteins are an expression of the information content of genetic material, and are the means by which the information content of that genetic material is effected. The translation process maps three-nucleotide codons to amino acids, building up proteins as polypeptide chains of some combination of twenty of these building blocks. These amino acids all contain the same atomic backbone, but each has its own unique side chain, and between them the twenty amino acid side chains cover a diverse range of chemical properties. There are hydrophobic and hydrophilic side chains, side chains of varied length, and varied electronic charge.

However, while this ‘polypeptide-space’ is certainly vast, the different possible combinations and orientations of the twenty amino acid side chains still represent a small subset of total chemical space, and there are limits to what proteins can achieve using only these chemical species. The principal atoms involved (carbon, nitrogen, oxygen and sulphur) are all clustered at the upper right corner of the periodic table, all have broadly similar electronic properties, and are all similar in size.

Other regions of the periodic table are not available to evolution by the direct means of encoding them with DNA — there is no codon that will cause them to

be incorporated directly — but these other chemical species can be incorporated through other means. The three-dimensional structure of a protein is determined by primary structure, so the genetic code can create a protein which presents a region of its surface that the ‘desired’ (in the evolutionary sense) chemical species will be attracted to and will associate with the protein through non-covalent means (assuming it is present in the protein’s environment). These are ‘cofactors’.

While there are many organic, molecular cofactors in use, metal ions — particularly transition metal ions — have particular properties that make them very useful as cofactors. To understand why this is, it will be necessary to briefly review the electronic structure of atoms, show how transition metals are distinguished by their particular electronic structure, and then look at how zinc’s properties are unusual by the standards of other transition metals.

Atoms’ electrons are arranged into energy levels, which are in turn sub-divided into orbitals - regions of space in which electrons will usually be found. For example, atoms can hold two electrons in the first, lowest energy level, which has a single s-orbital. The second energy level can hold eight electrons, with one slightly lower energy s-orbital and three slightly higher energy p-orbitals. This describes the electron arrangement of the first ten elements.

The third energy level is slightly more complicated. It has three sub-levels (an s-orbital, three p-orbitals, and five d-orbitals) but the d-orbitals are actually in a higher energy state than the s-orbital of the *fourth* energy level, which ‘fills up’ first. The order in which the orbitals fill up is therefore 1s2s2p3s3p4s3d4p and so on (see Figure 1.1). An atom with thirteen electrons (i.e. aluminium) will have its thirteenth electron in a 3p orbital, an atom with nineteen electrons (potassium) will have its nineteenth electron in a 4s orbital, and only once an atom has to ‘add’ a twenty-first electron (scandium) will the 3d orbitals be used.

Atoms which use these d orbitals — the transition metals — have unusual properties not encountered in previous atoms as a consequence. And it is precisely these properties that make them so useful to proteins, and why evolution has driven proteins to acquire them.

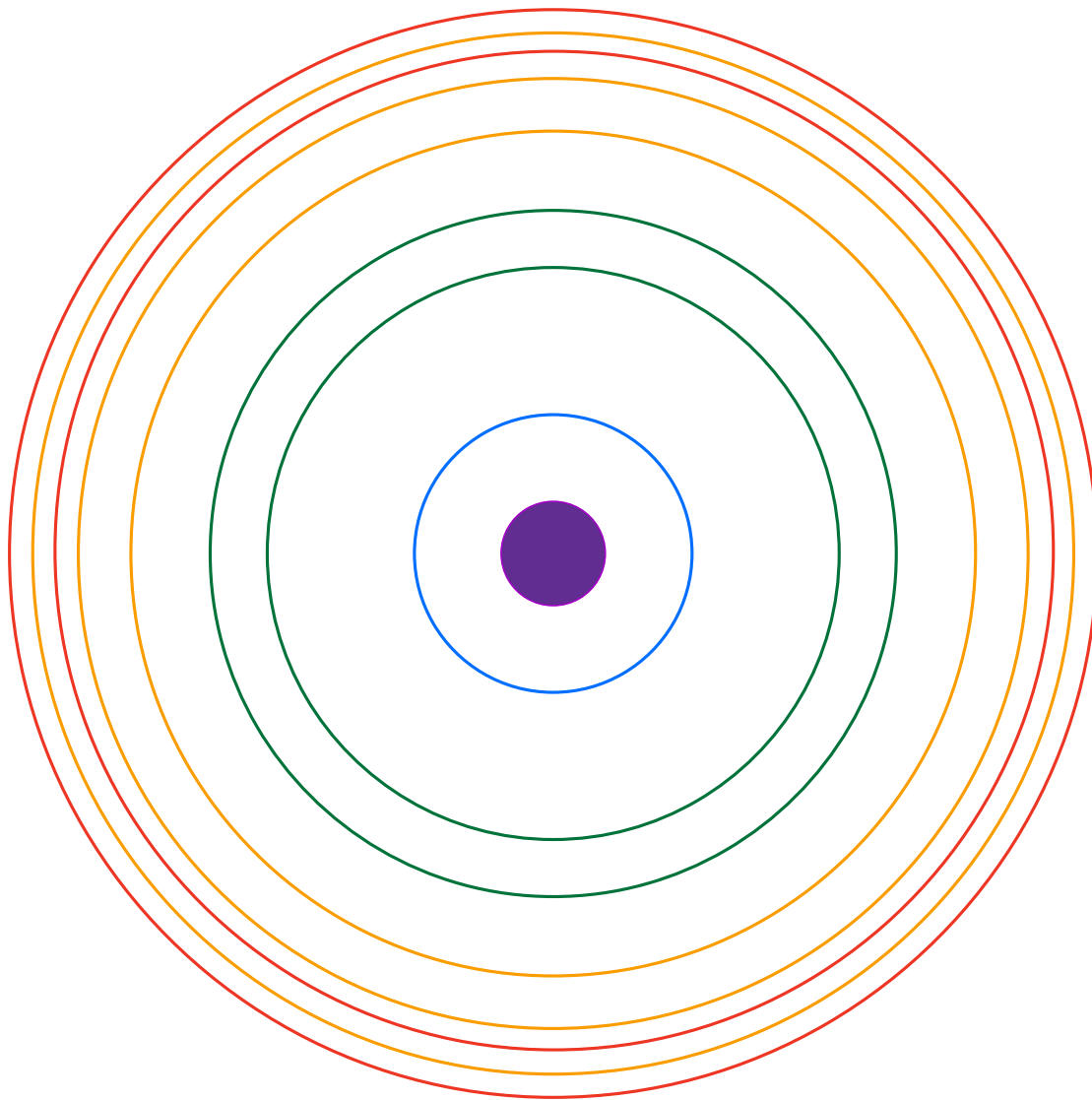


FIGURE 1.1: Stylistic representation of atomic orbitals, showing levels 1 (blue), 2 (green), 3 (orange) and 4 (red). The difference in energy levels becomes less pronounced further out, to the point where difference in orbitals within these levels becomes greater than those between them by level 4. It is for this reason that the highest energy orbitals of level 3 (3d) are higher in energy than the lowest energy orbitals of level 4 (4s). The distances shown here are not to scale with the actual energy differences.



### 1.1.2 Transition Metals

Transition metals are often employed as cofactors by proteins, particularly in catalysis. As they occupy a different region of electron configuration space than the organic non-metal atoms that make up the biological amino acids (for reasons just outlined), they can provide functionality that would otherwise be unavailable.

Atoms ‘prefer’ to have full energy levels and sub-energy levels — in the sense that if they are in an environment which draws electrons away from them (an oxidising environment) it will be energetically favourable to lose whatever number of electrons causes them to fall back to the last filled level, emptying the highest energy level. Likewise, if in an electron-rich environment in which electrons are ‘pushed’ to the atom (a reducing environment), they will prefer to take only the precise number of electrons which fills the current energy level.

For elements before scandium, this is straightforward: energy levels are far apart, and there is generally only one clear point to fall back to, or advance to. Sodium for example, has a single 3s electron, so when in an oxidising environment it can relinquish this one electron, and become  $\text{Na}^+$ . It has a single oxidation state.

Once d-orbitals start being filled however, this becomes less straightforward. Rather unintuitively, when it comes to *losing* electrons, different rules govern which energy levels are the first to be unfilled — the 4s electrons can be considered the highest energy, and these will be lost first. Therefore most atoms with unfilled d-orbitals can adopt the 2+ oxidation state. The d-orbitals are rather close in energy level, so if there is further oxidative pressure, these can be lost too. This property of having multiple stable oxidation states is one of the chief characteristics of transition metals. It is partly for this reason that transition metals are so widely employed by evolution as protein cofactors, particularly in proteins which catalyse reactions — enzymes. A common mechanism for catalysing a reaction is to stabilise an otherwise very thermodynamically unfavourable intermediate that might have a strong charge associated — by using a chemical species which can have one oxidation state in its native state, but which will happily switch to a more oxidised form, the excess charge from the reaction intermediate can be temporarily ‘stored’ in the cofactor, stabilising the intermediate.

As cations, the transition metals also act as efficient Lewis acids — they can accept lone pairs of electrons. This is also useful in catalysis as it can stabilise intermediate structures by withdrawing excessive negative charge from them. This achieves a similar effect, but does so using a different mechanism. The metal atom is not being oxidised, however.

Another curious property of atoms with partially filled d-orbitals is the so-called breaking of degeneracy. In a single, uncharged atom of such elements, the five d-orbitals are said to be degenerate, in the sense of having identical energy levels. This is also true when oxidised, but when these charged ions interact with a solvent molecule or other ligands, some d-orbitals end up closer than others to the orbitals of the incoming ligand, entering a higher energy state than the others. The previously uniform d-orbitals split into two slightly different energy levels, and the previous degeneracy is said to have broken.

This confers some unique properties on these elements, most visibly that of colour. The electrons will prefer to occupy the lower energy d-orbitals, but the energy gap between them is small enough as to be in the visible light range, so light passing through a solution of these ions will have the corresponding wavelength absorbed, with the energy used to ‘promote’ the electrons and with the light not absorbed determining the solution’s colour.

It also means that different ligand geometries are associated with different changes in energy, meaning that certain geometries end up being preferred over others. This has implications for proteins, which must fold in such a way that the liganding residues adopt the correct geometry.

Unsurprisingly then, many proteins have evolved to take advantage of these useful properties of transition metals by presenting binding sites on their surface that will acquire one of them. This is generally done by bringing residues with available lone pairs into close proximity to each other, in an arrangement that matches the dimensions and orbital geometry of the desired metal, meaning that the geometric properties of these binding sites tend to match the particular properties of the metal atom they are adapted to. Each metal has its own particular properties arising from its electron configuration, and zinc’s are some of the most atypical.

### 1.1.3 Zinc's Unique Properties

Though a transition metal in the sense that it has ten d-electrons but no 4p electrons, it is an unusual one, to the point where it is often not even classified as one<sup>1</sup>. Many of the properties of transition metals derive from *partially filled* d-orbitals, which zinc does not have. The five fully filled d-orbitals are relatively stable, meaning it can only really be oxidised to 2+ by losing its 4s electrons, so it has just one oxidation state. Electrons cannot be 'promoted' from one d-orbital to another in solution as all spaces in the orbitals are already filled, so it has no colour or spectroscopic activity when in solution. At first glance, zinc would appear to be rather unremarkable metal, unworthy of much consideration by evolution.

However, the data shows otherwise. About 10% of all proteins in the human genome are zinc proteins with a zinc binding site [1] — making it the second most abundant such metal after iron. It is also the only metal found in all classes of enzyme (hydrolases, oxidoreductases, lyases, transferases, ligases and isomerases) [2]. Clearly, evolution has found zinc very useful.

In fact it is precisely that unremarkableness that has made zinc so attractive. Having just one oxidation state may mean that it cannot hold onto an electron mid-reaction, but it also means that the ion will retain its electronic properties in a wide range of reducing environments, and its status as a particularly good Lewis acid means it is still very useful in catalysis.

Its lack of spectroscopic activity may make zinc solutions colourless, but the same arrangement of d-orbitals that causes this also means that there is no energetic penalty for zinc coming out of solution (where it is octahedrally coordinated) and into the binding site because such a transition is accompanied by a rearrangement of the orbitals geometrically. In the case of metals with incomplete d-orbitals, more of the orbitals are forced into higher energy states. For zinc, there is no energetic penalty in moving from the octahedral geometry associated with solvation by water, to the (typically) tetrahedral geometry of protein binding. This equivalence

---

<sup>1</sup>It is not particularly worthwhile to debate whether zinc is or is not a transition metal objectively — transition metals are an artificial category invented to serve a particular purpose, and the edges of that category are blurred. By some definitions it is, but by other definitions it is not. What really matters are its objective physical/chemical properties, not the name of the category humans have assigned it to based on those properties.

in ‘Ligand Field Stabilisation Energy’ gives zinc an advantage from the protein’s perspective [3].

As a somewhat unusual metal cofactor, the corresponding zinc binding sites that proteins form around them also have unique properties, distinct from those of other cofactors, and other transition metals. These will be explored next.

## 1.2 Properties of Zinc Binding Sites

What do zinc binding sites ‘look like’ from a structural point of view? What are their properties at the atomic scale?

These questions were addressed by researchers from some of the very first crystal structures, and continue to be a topic of research today. The properties of zinc binding sites that distinguish them from other spatially proximate clusterings of residues are ultimately what must be used to predict them, so the advances in this field are crucial to any zinc binding prediction model. The history of our understanding of these properties will be reviewed here.

By the middle of the 1980s, a number of structures had been produced and already a few general themes that would recur over and over again were observed. Catalytic sites (those which are involved in the catalysis of some reaction) tended to have three protein ligands and one water ligand, whereas structural sites (those not involved in catalysis, but which stabilise local regions of the protein) generally had four protein residues, for example. One of the first reviews in this area to examine the structures obtained so far identified coordination by sulphur and nitrogen, and tetrahedral geometry, as the two defining characteristics of zinc binding sites [4]. With the benefit of hindsight we can see that this is a slight simplification, but it shows that a consensus about ‘typical features’ was beginning to form. Another early review would go on to list most of the basic properties of zinc binding sites we now know — their division into structural and catalytic sites, the near ubiquity of water in catalytic sites, the much stronger preference for tetrahedral geometry in structural sites than in catalytic sites, and the preference for histidine, cysteine,

aspartate and glutamate residues [2]. These properties would be repeated and elaborated upon by a number of similar reviews in this period [5–7].

The more structures that were available, the more detailed the inferences that could be made became. A series of reviews looked at typical atom geometries in cysteine binding [8] and histidine binding [9], and while both papers looked at metal binding generally rather than zinc, in both cases the observations were found to be largely metal-agnostic. It was shown that cysteine generally coordinated the metal such that the Zn—S—C $\beta$ —C $\alpha$  torsional angle is either 90° or 180°, and that histidine residues generally coordinate the metal via their NE2 nitrogen atom, with the metal lying in the histidyl plane.

Another structural characteristic of zinc binding sites (and indeed metal binding sites generally) was the ‘hydrophobic contrast’ that seemed to exist around them [10, 11]. Briefly, this is the observation that metal atoms in a protein tend to be surrounded by a shell of hydrophilic atoms (as might be expected), but that these were in turn surrounded by unusually hydrophobic atoms (measured against the average ‘background’ hydrophobicity of the other residues). This could be implemented as a function of coordinates, which would be at a maximum when centered in such a concentric sphere. It was shown that maxima of this function would cluster around metal binding sites.

Whilst it might be expected that zinc binding sites would have some kinds of structural consensus, it is not immediately obvious that they should have any kind of predictable sequence patterns. However in 1989 it was shown, albeit from a relatively small dataset as existed at that time, that catalytic zinc binding sites seemed to have characteristic “short and long spacer sequences” [12]. That is, the three residues that make up catalytic binding sites seem to be made up of two residues separated by a small stretch of amino acids (around five) and a third residue that is much more distal on the sequence. They proposed that the first two residues were acting as a nucleus, a proto-site that is stabilised by the third residue. This pattern was confirmed by later studies, with the short spacer being found to be generally 2 to 7 residues long [13].

This was not however the *first* sequence motif associated with zinc binding sites - a few years previously a curious pattern of residues in a transcription factor

sequence led to the discovery of a new class of zinc protein, and the creation of an entirely new field of genome engineering.

In 1985 it was observed that a particular transcription factor in the model organism *Xenopus laevis* was a zinc protein, and moreover that it seemed to be made of globular subunits of roughly equal mass, each with a repeating C...C...H...H motif [14]. They hypothesised that perhaps this particular transcription factor was essentially a string of zinc binding domains — referred to as ‘fingers’ — each stabilised by a zinc atom at the base. The very regular repeating pattern of thirty residues, each with these four residues in the same location, was key in uncovering these ‘zinc fingers’. While not initially thought to be widespread, in the months and years that followed, more examples of these genome-interacting proteins with similar sequence motifs were identified [15]. Finally, in 1989, the structure of the original zinc finger was solved, largely confirming the initial speculations about the structure of this protein, and the role of zinc within it — the domain was a finger-like projection which could bind DNA bases at the tip, and which was stabilised by a C2H2 zinc binding site at the base.

The field continued to grow. By 1998 there were thought to be at least 500 zinc finger proteins, and possibly as many as 1% of all mammalian genes [16]. Much of this field is concerned with the ‘fingertip’ end of zinc fingers, which actually implements the DNA-recognising properties, though for the purposes of this project it is the base that is of more interest. There are a few points to take away from this brief digression. The first point, which will be returned to in the section on zinc binding site prediction, is that the zinc fingers were discovered because of a characteristic *sequence* pattern, emphasising that such prediction is both possible and potentially offers many insights. The second is that until this point, much of the interest in zinc binding sites was in catalytic zinc binding sites; there were fewer structural sites available to study, and most reviews from this period addressed them last, if at all. From this point on however, many more became available to study, and it became clear that they were responsible for much of zinc’s ubiquity across the proteome.

Most of the early work on characterising the typical properties of zinc binding sites was done by manually surveying the crystal structures available and drawing

conclusions from them. But as the 1990s wore on and the number of such structures available reached the hundreds, this became increasingly impractical. Instead, from this point onwards, the sites would have to be collated algorithmically from the Protein Data Bank [17].

The first such database survey was in 1998. They found 387 zinc proteins, and after excluding some on quality and experimental technique factors, they were left with 111 [18]. Some of their findings were in line with what was already known: the three residue/one water combination for catalytic sites, the preference for tetrahedral geometry, histidine's preference for one nitrogen over another (by a ratio of 70:30, they found). They also identified a new, almost ubiquitous, motif across zinc binding sites which they call the 'elec-His-zinc' motif whereby zinc is liganded by a histidine which in turn forms a hydrogen bond with an electron donor of some kind.

This was followed by other reviews [19–23], showing similar results: that zinc-ligand distances are similar to those found in small molecules, and further sub-categorising the ever increasing number of zinc finger proteins.

Increasingly, the modes of classification were becoming more sophisticated too. Initially, two kinds of binding site were recognised: catalytic sites, which catalysed some metabolic reaction, and structural sites, which did not and just stabilised structure. Later came the recognition that some sites required the action of multiple metals in a single functional unit — either multiple zinc atoms, or a zinc atom in conjunction with another metal atom. From 2001, a fourth kind was recognised, interface binding sites [24]. It had been known since the early days of insulin research, of course, that in some cases zinc atoms could stabilise polymeric structures, but by this point there was enough data to classify these as their own category, with their own properties — that cysteine is very underrepresented in them, that stabilisation is often done by beta sheets rather than alpha helices (common in other zinc binding sites), and even evidence that these sites could also have catalytic roles in addition to their role in quaternary structure.

A more in-depth look at the differences between catalytic and structural sites even seemed to offer a means of automatically classifying zinc binding sites [25], though generally during this period this was a task done manually. They confirmed

that catalytic zinc binding sites tended to have more unusual atom distances and geometries (imposing a ‘strain’ upon them that we will return to in the following section), and looked in detail at catalytic sites’ disdain for cysteine residues. They found that a relatively simple method of assuming a site is structural if it has more than one cysteine residue *and* no water, and assuming it is catalytic otherwise, assigned the correct label in 90% of cases in their dataset. They also looked at the possibility of using atom distances to classify zinc binding sites, though it must be remembered that when looking at differences in atom distance on the order of single Ångströms, structure with poor resolution should be removed from the dataset if you are to make sensible inferences.

Atom distances themselves have been a major focus of research. A typical example is a 2007 review of 994 structures [26], showing that histidine approaches zinc more closely than cysteine (2.11 Å versus 2.32 Å on average) with water distances generally somewhere between those two figures. The vast majority of metal liganding atoms are within 3 Å of the metal [27]. It has been speculated that some non-zinc metal-ligand distances are longer than might otherwise be expected because of the ‘Jahn-Teller effect’ [28]. This phenomenon, like the Ligand Field Stabilisation Energy that makes zinc so attractive in the first place, results from the different energy levels of the d-orbitals in these metals which distorts them and lengthens the bond. This does not affect zinc binding, but does affect other metals.

The properties of zinc binding sites are not just limited to the metal itself and the residues that coordinate it. It is increasingly recognised that the ‘secondary shell’ around the primary residues are of great importance, by providing a network of stabilising hydrogen bonds around the liganding residues to keep them in their precise orientation [24]. This concept has been extended to the idea of a ‘Minimal Functional Site’ [29], proposed as the smallest unit of metal binding that should be used when classifying and studying the phenomenon. The authors in this case used all residues within 5 Å of a liganding residue, but more important is the general idea that the residues around the coordinating residues, which are often part of secondary structure, are crucial to understanding the metal binding itself.

There have also been reviews into the efficacy of these database reviews at all, and the limits of what can be inferred from them. A 2005 review [30] suggests



that while purely spatial properties, such as the location of the metal ion and its liganding residue identities, can be provided reliably by crystallography, often the precise coordination geometry does not agree with spectroscopic data, and should be treated with caution. They also warn against trying to infer redox state from crystallography, though this does not particularly apply to zinc. A later review [31] pointed out that reported atom distances increase as a function of resolution, and stresses the need to take this into account. They criticise a number of earlier studies for treating unlikely geometries or electronic configuration as ‘real’, or for treating zinc salts and crystallographic artefacts as biologically relevant zinc binding sites.

Finally, it is worth examining the numerous secondary databases that have been created over the past decade and a half relating to zinc binding sites, and which offer a web-accessible means of viewing the properties of zinc (though usually metals generally) binding sites.

- MESPEUS was one of the first — a general-purpose database of metal binding sites generally, which focused on providing basic geometric information [32], and which allowed basic surveys of atom-length patterns to be carried out.
- ZifBASE had a more specific focus — this was a database of zinc fingers specifically, and provided much more than structural information from the PDB: sequence information, DNA targets, and physiochemical properties like isoelectric point were also provided [33].
- Probably the most important resource in this area is MetalPDB [34], a database of metal binding sites, similar to MESPEUS, but with more information and a more sophisticated means of classifying them. They divide their sites into ‘equistructural sites’ - (those which have binding sites of similar structure) and within those ‘equivalent sites’ (equistructural sites with the same metal). They place a great focus on the secondary shell, as part of the ‘Minimal Functional Site’ concept they introduced earlier, and have recently updated the resource with a greater focus on apo-structures [35]. They have also provided a number of related resources, such as MetalS3

[36], a tool for searching the database via structure pattern rather than keyword, and tools for classifying sites based on similarity [37]. An issue with MetalPDB however is its reliance on asymmetric units in the PDB files they parse. Often the raw coordinates of such files represent only the repeating unit of the crystal structure, and the actual ‘real’ biological protein must be assembled from that data using transformation matrices contained with the files. MetalPDB (and most other resources) do not do this, which makes some of their ‘binding sites’ rather unrealistic.

Another impression one gets immediately when surveying these resources is that with the exception of MetalPDB, all of these resources have ceased to exist — their URLs go nowhere, or their sites are broken beyond reasonable use and obviously not maintained. This is a problem generally with academic software, in which the creation of resources is funded by initial grants. However, because they do not generate revenue themselves, they need grant funding of some kind in perpetuity if they are to remain online — which they generally do not have. The solution to this problem is not obvious, and in any case beyond the scope of this PhD, though the burden that this maintenance poses for a lab or organisation can be lightened somewhat by automating as much of the process as possible (where appropriate).

Other resources are focused more on validating or describing a given metal binding site that it is presented with, such as the coordination geometry classifier FindGeo [38], or the validation tool CheckMyMetal [39], which uses known properties of metal binding sites to determine if a proposed binding site is realistic.

### **1.3 How do Zinc Binding Sites Work?**

In parallel with an increasingly clear understanding of the properties of zinc binding sites, theoretical and analytical research has been performed on the mechanisms and physics underpinning zinc binding. These two areas of research are complimentary and interconnected, with the properties in large part explained by the physics of the binding sites, and insights into the mechanisms derived from experimental observations.

One of the earliest concepts to be introduced in this area was that of the ‘entatic state’ of metals in biology [40], in the 1960s. Essentially this was the idea that metal atoms in proteins have different geometries and properties than they do in smaller molecules and which they would ‘prefer’ energetically, because the protein backbone is able to impose restraints upon it. It is a kind of ‘strain’ that the protein’s overall conformation imposes on the metal and which gives the metals any unique properties they may possess which are not found outside of biology. For example, it may cause an otherwise unfavourable metal oxidation state to be favoured over another, as can be seen with iron.

This concept was later elaborated upon to show that this strain should be thought of as a two-way process [41]. The authors note that “[They] described the valuable strains induced by the protein in the metal... but they could equally well have pointed to the strain induced in the protein by the metal to it. The overall idea was that of catalytic perfection induced by strain” — a strain which makes the enzyme “energetically poised for catalysis” [42].

This is a concept that applies to metalloproteins, but once it started to become apparent just how widely used zinc was in the 1980s, attention began to turn to answering the question of why zinc in particular was of such apparent use to proteins.

One property noted early on was that zinc atoms have a full *d*-shell, making it slightly unusual among transition metals [42]. This means that when ionised, zinc will lose its two 4*s* electrons to become 2+, but there are no other oxidation states that it can realistically adopt — it is ‘redox inert’. This means that the protein can rely upon the metal retaining its oxidation state regardless of the oxidation environment it finds itself in, a stability whose utility is clear.

Another property of this full *d*-shell that would be elaborated upon later was the absence of a phenomenon known as ‘ligand field stabilisation energy’ [3, 43–45], as previously outlined. Briefly, this refers to the fact that atoms with incomplete *d*-shells, like most other transition metals, have preferred coordination geometries and experience energetic penalties if coordination angles deviate from this. If the *d*-shell is full however, no one geometry is preferred over another, which offers the protein much more flexibility.

Another property of zinc that is revisited time and time again in the literature is zinc's role as a Lewis Acid; it will readily accept electron pairs from electron pair donors [46]. It has been noted that while other metals are superior Lewis acids from a purely chemical standpoint, zinc will more readily form tetrahedral complexes [47], making it frequently the Lewis acid of choice.

This effective electron-pair-accepting that zinc exhibits is important because it is useful wherever zinc is used in catalysis. One of the first zinc enzymes to have its precise mechanism of action investigated was carboxypeptidase A, a digestive enzyme. It was suggested that the water molecule that coordinated zinc as one of its ligands was being 'activated' by zinc — its electron pairs were being withdrawn by the metal, making it less able to retain its hydrogens and becoming the more reactive hydroxide. Some structural evidence to support this came from the crystal structure of carboxypeptidase A in complex with a substrate that was known to be hard for the enzyme to digest for reasons at that time unknown. The crystal structure revealed that the substrate was inadvertently looping around and coordinating the zinc atom, blocking the water molecule and accounting for the much slower rate of hydrolysis [48]. Another means by which zinc can aid in catalysis is by stabilising any negatively charged intermediate in the reaction by offering a temporary repository for their electrons, as is the case with carbonic anhydrase [49].

Indeed, multiple papers have emphasised this 'primed' property of catalytic zinc sites. They have more distorted geometry than the more typically tetrahedral structural sites [19], the liganding atoms tend to be further from the metal [25], and they are generally more thermodynamically unfavourable when modelled quantum mechanically [50].

The means by which protein residues interact with the metal can also have important consequences. Aspartic acid and glutamic acid are both relatively common zinc binding residues, each doing so via their side chain carboxylate groups. They can either contribute one liganding oxygen atom ('monodentate' binding) or both (bidentate binding) (see Figure 1.2). While early theoretical models suggested that the different modes were determined merely by the other residue identities, with more negative other residues favouring mere monodentate binding [51], it

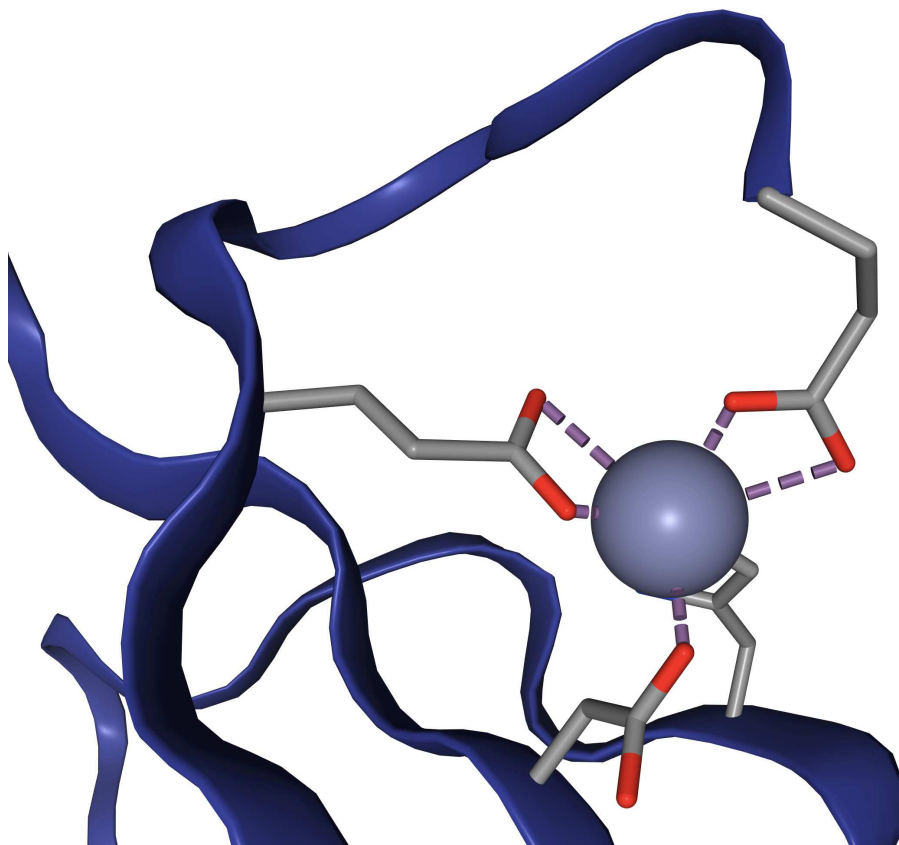


FIGURE 1.2: A zinc binding site demonstrating both bidentate binding via two glutamate residues, and monodentate binding via one aspartate residue. This visualisation is taken from ZincBind (binding site 6VDA-1), and is a site in PDB 6VDA, a *Thermotoga maritima* transport protein [53].

has more recently been suggesting that a residue may use both. The so-called ‘carboxylate shift’ is theorised to occur when an acidic side chain in a catalytic zinc binding site switches from monodentate binding to bidentate binding so as to keep the number of coordinating atoms around the zinc constant, and thereby reducing the activation energy of substrates entering and leaving the complex [52].

The reason for the preference for cysteine in structural sites has also received attention. It has been proposed that cysteine residues transfer more charge to the zinc atom, reducing the potency of its 2+ charge and making it less reactive [25].

Finally, it is worth bearing in mind that zinc does not become ‘part’ of the protein as part of the translation process, but binds to the fully translated apo-protein, in a process whose mechanisms are also crucial to understand. It is possible to compare the apo- and holo-forms of zinc binding proteins using sequence identity. For every zinc binding protein structure you have, you can search the Protein Data

Bank for proteins with a highly similar structure that does not have zinc in the same position. This is precisely what a 2005 review did [54] in a bid to investigate the mechanisms by which zinc binds proteins in the first place. After looking at 210 apo/holo pairs, they found that often, at least part of the coordination shell is in place before the metal is in place. This is consistent with the ‘nucleus’ theory of the short and long spacer sequences, outlined above. Furthermore, they found that in just 40% of cases was there any rearrangement of side chains upon metal binding, and in just 14% of cases did this involve backbone rearrangement. This mode of zinc binding is encouraging for efforts to predict zinc binding from apo-structure, as we will see in the next chapter.

This zinc binding should also not be thought of as permanent. Metalloproteins are in equilibrium with free metal + apoprotein, and the zinc binding will have an on and off rate like any other complex. While this project is largely focused on proteins where the zinc is essentially a permanent feature of the protein because it is essential for some function, it has been pointed out that these are merely proteins where the equilibrium is far to the right, and that all metal proteins lie on a continuum in this regard, with zinc transporters (for example) having an equilibrium further to the left [55]. This equilibrium constant is determined by the geometry of the binding site, and the secondary shell around the liganding residues seem to play a very large role in fine-tuning this affinity by exercising careful control over the precise positioning of these side chains [56].

## 1.4 The Protein Data Bank

The ultimate data source for almost all the data used in this project is the Protein Data Bank (PDB) [57]. This is a Bioinformatics Database jointly administered by the Research Collaboratory for Structural Bioinformatics (RCSB), the Protein Data Bank in Europe (PDBe) and Protein Data Bank Japan (PDBj) — each a member of the overarching Worldwide Protein Data Bank (wwPDB). These organisations accept depositions of solved protein structures from researchers around the world, releasing them as PDB structure files with four character IDs. There are over 160,000 structures available in the PDB in 2021.

These ‘solved structures’ contain the three-dimensional coordinates of the atoms that make up a macromolecular structure — typically a protein or protein complex, often with small molecules bound, but there are nucleic acid structures in the Data Bank too, and increasingly large ribosomal structures. The structure files also contain meta-data about the experiment which produced the raw data from which the coordinates were calculated, details of any refinement done to those coordinates, and annotations of the names and entity relationships of the molecules present (which residues belong to which polymer, which small compounds have binding sites in which polymer, which molecules are instances of the same biological entity, etc.).

These three-dimensional coordinates are obtained by isolating the protein in question, performing some kind of experimental, analytical technique on the sample, and inferring the locations of the atoms from that data.

- By far the most common technique used is X-ray Diffraction. The sample is crystallised such that the structure arranges itself in a single, solid repeating lattice, and then X-rays are passed through this structure. This beam of electromagnetic radiation is diffracted by the atoms of the structure and, because the proteins are all aligned, this diffraction will create a discrete ‘diffraction pattern’ of spots on the photoreceptive surface behind the sample. Specifically it is electron density which scatters the X-rays, and it is this which is detected. This pattern of spots can be used to generate an inferred three dimensional map of electron density by using Fourier Transforms, from which the location of the atoms themselves can be ‘fit’.
- Nuclear Magnetic Resonance (NMR) spectroscopy is another technique used to generate atomic coordinates. Here the sample is placed into a very strong magnetic field, which causes the quantum spin of the  $^1\text{H}$ ,  $^{13}\text{C}$ , and  $^{15}\text{N}$  nuclei to align with this field. Pulses of radio waves are then passed through the sample, which temporarily aligns the spin of the atoms with this electromagnetic radiation. When they relax back into alignment with the magnetic field, they emit radio waves which are detected, and which can be used to infer the proximity of the different atom types in the sample. This contains

enough information to build up a map of atom connectivity, and from this atomic coordinates.

- An increasingly prevalent technique is Cryo-Electron Microscopy (Cryo-EM). Rather than holding proteins in place by crystallising them into a repeating lattice and inferring their coordinates using X-Rays, they are ‘flash frozen’ by rapidly lowering the sample’s temperature, and then probed with beams of electrons.

Despite the increasing rate of submission of Cryo-EM structures, the majority of structures considered in this project have still been solved with X-Ray Crystallography.

The field of structural biology is diverse, but there are a number of concepts from this field and the protein structures mentioned above which are of particular importance to this project.

- Resolution. Typically given in Ångströms (0.1 nm), this is a single value metric of the quality of a protein structure. It is essentially a measure of how ‘detailed’ the electron density map is. High resolution structures (confusingly these are those with lower values, around 1 Ångstroms) might detect the contours of individual atoms within a protein residue for example, whereas lower resolution structures (higher values, above, say, 3 Ångströms) might merely have residue-shaped blobs. In analyses where the coordinates of atoms need to be relied upon with reasonable precision, a common early step on the pipeline is to discard structures with resolutions below some threshold.
- R-values. Another single-value that represents the quality of a protein created from X-Ray Diffraction. Part of the process of generating an electron density map from a scattering pattern is an iterative process whereby the researcher creates a putative set of coordinates, calculates what pattern of scattering would theoretically correspond to those coordinates if they were accurate, compares this with the actual scattering map, and refines the coordinates based on the difference. This difference between the actual scattering



pattern, and the scattering pattern that would produce a given set of coordinates, is the R-value. A perfect match would be 0, but typical values are around 0.2. A distinction is usually made between these R-values, and ‘free’ R-values, in which around 10% of the observations are removed before refinement, and then the refinement is performed on the remaining 90%, with the free R-value a measure of how well the new simulated diffraction pattern predicts the remaining 10%.

- **Temperature factors.** The above metrics apply to an entire structure, but the quality of a set of coordinates might be better in some regions than others. Temperature factors represent this localised quality, as each atom has its own temperature factor. They are a measure of the uncertainty over the atom’s position in space, with higher temperature factors indicating greater uncertainty (representing either statistical uncertainty regarding where the atom is likely to be, or actual variation in the atom’s location due to excitation — hence the name ‘temperature factors’). This uncertainty can be a single value ‘isotropic’ temperature factor (applying in all directions) or ‘anisotropic’ (directional uncertainty).
- **Biological Assemblies.** The raw coordinates of PDB structures as given contain the ‘Asymmetric unit’, which is the repeating unit of the solved structure. This may not be the actual biologically relevant entity, so a number of assemblies are generated computationally using this base unit, and their biological feasibilities calculated from metrics such as buried surface area and the interaction strength of associated surfaces. These possible assemblies are given in the metadata of the PDB files as transformation matrices, along with the associated software-calculated metrics.

## 1.5 Conclusion

Zinc is a crucial metal for a wide variety of functions in biological systems, and across a wide spectrum of organisms. Its chemical properties — particularly its lack of redox activity which gives it an essentially permanent 2+ charge in physiological environments regardless of the reducing/oxidising environment it finds itself

in — make it uniquely useful to proteins in carrying out functions that they would be unable to do with either the chemistry of the twenty canonical amino acids (whose atoms are largely electron donors, not acceptors), or other transition metals (whose redox state can alter depending on the environment). Chiefly those functions are the catalysis of reactions as a key component of enzymes, or as a stabiliser of local protein structure — either a region of a single chain, or at the interface between two chains.

While zinc has other occasional roles, such as as a cellular messenger, these two primary roles require the zinc ion to be tightly and permanently bound to the protein in question. This requires the protein to fold in such a way that it brings together liganding residues for zinc that the metal will have a strong affinity for, due to the geometry and chemical identity of the atoms in question. These zinc binding sites have been examined in every greater detail as the number of zinc protein structures has increased, and certain key properties of the binding sites have been observed repeatedly at both the sequence and structure level.

Some of these characteristic features are the overwhelming use of histidine and cysteine, and to a slightly lesser extent the carboxylate residues aspartate and glutamate (with cysteine being more common in structural sites and the carboxylate residues slightly more common in catalytic sites), the near ubiquity of tetrahedral geometry of the liganding atoms, the presence of three liganding residues plus water in catalytic sites and four residues in structural sites, and the typical liganding distances.

Physically, the zinc binding sites seem to rely heavily on a network of stabilising hydrogen bonds holding the primary liganding residues in place, from the secondary residues around them. Binding between the zinc atom and the binding site does not usually rearrange the backbone of the protein significantly. Zinc largely facilitates catalysis by stabilising the negative charges of any positively charged reaction intermediates, making their existence less thermodynamically unfavourable for the short amount of time that they exist.

The existence of these typical features and well understood mechanisms of action is crucial in trying to predict zinc binding sites from structure or sequence, as it means that there are particular properties to be looked for. The next chapter will

examine the methods for making these predictors, and review previous attempts at predicting zinc binding using these techniques.

# Chapter 2

## Computational Techniques

In this chapter, the pre-existing algorithms and other computational techniques used to predict Zinc Binding Sites will be outlined.

It will not provide details of specific implementations used in this project, or any novel techniques developed — those will be presented in later chapters. It is instead meant to give a general mathematical background to these techniques, and familiarise the reader with the terms that will be used in later chapters.

### 2.1 Machine Learning

#### 2.1.1 Principles

This project ultimately created a predictor — something which can take either a protein structure or a protein sequence, and determine where, if anywhere, are zinc binding regions.

Machine learning is the use of existing data to create predictive models that can take new data and make inferences about it. It ‘learns’ the characteristics of the existing dataset and uses this to predict certain things about the new data. This can be ‘unsupervised’, where the model identifies the internal structure of the data and creates clusters from it, or it can be ‘supervised’, where the existing data is labelled with some correct output, and the model learns how to predict

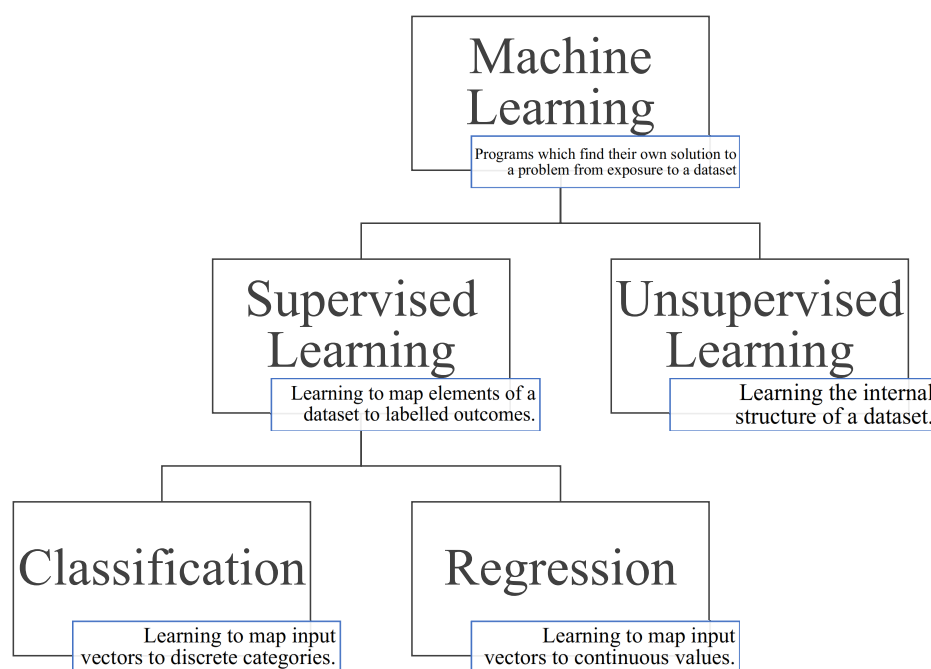


FIGURE 2.1: Overview of the branches of machine learning.

the output based on the input characteristics. If the output is a discrete category, this is classification; if it is a continuous value, it is regression. This is illustrated in Figure 2.1.

This project uses supervised learning, because the dataset being used to train the models already have the locations of zinc binding sites labelled.

When the inputs are passed to the model, they are done so as a set of ‘features’. These are numerical characteristics that the model actually uses to learn what kinds of input belong in which category. These are usually passed to the model as a vector of numbers and classification is just the mapping of vectors to discrete classes. Machine learning algorithms are ultimately mathematical functions, which cannot take abstract concepts such as ‘zinc binding residues’ as inputs. Anything which they accept must be turned into some numerical representation by making numerical measurements of it — this is its ‘representation’. When preparing a dataset, the researcher must decide which properties to use (feature selection) and how to encode them numerically (the representation).

Once the dataset has been processed, models must be ‘trained’ — a process which varies from algorithm to algorithm, but which is the process by which the algorithm changes its internal model to perform best on the problem it is given. It is given a dataset to learn from, which in the case of supervised learning is a list of examples of the type of object under consideration, each labelled with the outcome that the algorithm must try to learn. Training here, in the general sense, is the process by which the algorithm learns to develop rules that use the information given for each object to predict the output.

The simplest approach would be to give the model the entire dataset and let it learn from that. However the dataset is just a sample of the population of all the kinds of objects it represents, and some of the properties that the model would learn would in fact be statistical noise that is not representative or useful. If the model is trained on all the data, it will learn these properties that exist only in the sample, and will be a poor predictor when faced with other inputs — it is ‘overfit’. To detect this, the dataset is split into test and training sets, with the former being used to assess the model’s performance on data it has never seen before once the model is trained. Additionally, when the model is training and tuning its parameters, it further sub-divides the training set into ‘folds’ and reserves one fold for testing/optimising during training.

### **2.1.2 Supervised Learning Algorithms**

Supervised learning is the process by which a system learns to map input vectors representing the measurements of some object to discrete output categories, based on the values of those input vectors. However, there are multiple methods and algorithms for doing this. Each of them is suited to different kinds of problem, and these methods will be outlined here.

The most basic machine learning algorithm tries to find the parameters to some function of the input values, and then finds a threshold above which the positive class should be predicted. For example, some of the earliest methods to be developed were the Perceptron and the Adaptive Linear Unit. These create a linear function, with parameters that are learned during training. In this case ‘learning’

means that the output of this function is evaluated for every input in the training set, the sum of the squared errors is found (which acts as a score of how poorly the model has performed with its current parameters), and the derivative of this squared error function is used to change the values of the parameters using gradient descent. Gradient descent is a common optimisation algorithm that tries to find the minimum of a function, in this case the sum of squared errors as a function of input vector. In practice, a variant of gradient descent called stochastic gradient descent is generally preferred, where a random subset of the training data is used each time. This speeds up the rate of optimisation and allows the function to escape local minima.

Linear perceptrons are relatively primitive. They use a linear function and so can only separate data that is linearly separable in the first place, meaning that a line, or plane (or higher dimensional ‘hyperplane’) can divide the data in space. However they do illustrate many of the principles that are common to most or all supervised learning algorithms. Many of them have the overall architecture of a function with learned parameters and a threshold, for example. They also illustrate the difference between parameters and ‘hyperparameters’. A parameter is something the algorithm itself learns, and in this case the parameters are the coefficients of the linear equation, and the threshold value. So for  $n$ -dimensional input vectors, there will be  $n + 1$  parameters to learn. A hyperparameter on the other hand is a ‘setting’ that the user manually sets. For example, the number of passes through the data to make (epochs).

A more sophisticated variant of the perceptron is a supervised learning algorithm called logistic regression. In logistic regression, you still have a linear function whose coefficients the system tries to learn using (usually stochastic) gradient descent. However in this algorithm, before checking the output of the function against a threshold, the value is passed into a second ‘activation function’ which is not linear. In this case the activation function is a sigmoid-like function that maps all inputs to some value between -1 and 1. This logistic function takes the form:

$$\text{logistic}(x) = \frac{1}{1 + \exp(-x)}$$

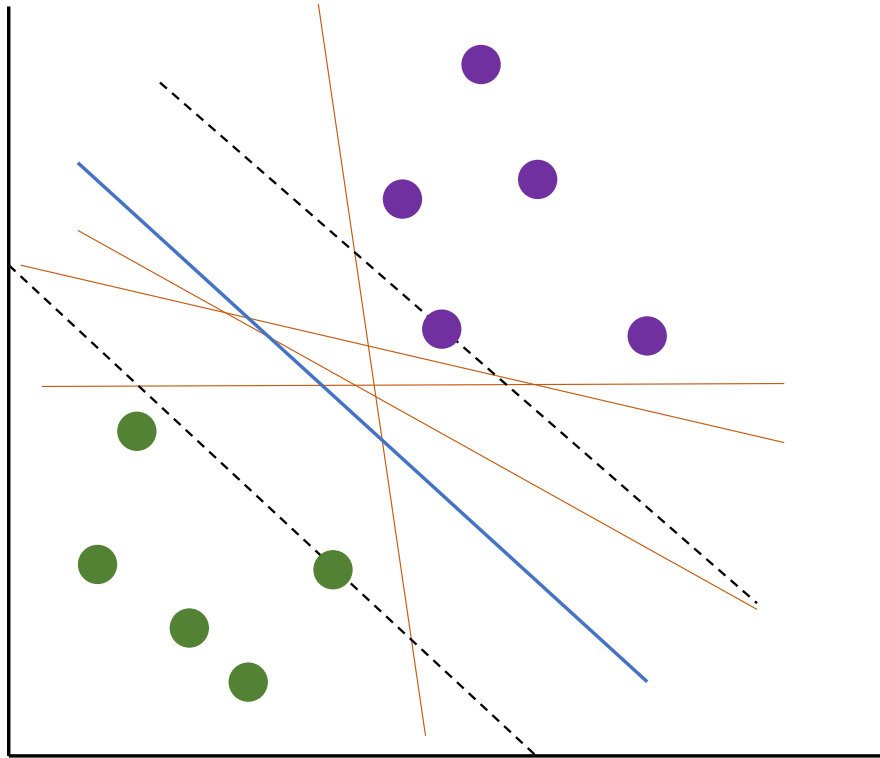


FIGURE 2.2: An example of Support Vector Machines applied to two dimensional inputs. The blue line is the decision boundary, equidistant between the support vectors (which form the dotted black lines). The orange lines are the other possible decision boundaries that a perceptron might arrive at, which would be much less generalisable.

Support Vector Machines (SVMs) are another variant of the basic perceptron model which is widely used in classification tasks. In this case, the learning algorithm is iterated until the distance between the closest input vectors to the decision boundary (the support vectors) is maximised — an extra consideration that increases the cleanness of separation between the categories (see Figure 2.2).

Not all classification algorithms take this form of fitting the parameters of a function and then applying a threshold cutoff.



Decision trees, for example, divide the input space into regions by a series of branching cutoffs. At the first ‘node’, a dimension and a value which maximally separates the categories is picked, bifurcating into two branches. For each of these two branches, another bifurcation takes place, and so on until the categories are separated. The number of nodes (the depth) is a hyperparameter that is chosen beforehand. This can be a very important consideration — too few nodes, and the algorithm will usually not be able to cleanly separate the data, with low recall and/or precision being achieved. However, if there are too many branches, the model will become overfit — it may perfectly separate the training data, but in doing so will learn statistical noise present only in that sample, reducing its ability to generalise.

The metric that decision tree algorithms use to identify how to bifurcate is entropy, a measure of uncertainty in this case. At each bifurcation point the algorithm must select a feature and a cutoff value for that feature, and it does so based on the combination that would best split the two output categories under consideration (i.e. the combination that would most reduce entropy).

An extension of the decision tree that can be very effective is the random forest classifier. This algorithm generates multiple decision trees (the precise number being a tunable hyperparameter), each of which is trained on a random subset of the available training data. When a new input is to be classified, each of the trees classifies it according to its own training, and a majority vote is taken to determine the forest’s verdict as a whole. This is an example of ‘bagging’ (bootstrap aggregating), where the noise and errors of individual models are canceled out in averaging.

Another supervised learning classification algorithm is K-Nearest Neighbours. This is a slightly unusual algorithm as it has no parameters that are tuned during training. Instead ‘training’ in this case simply means giving it the entire training dataset, which it stores, and this entire dataset is used when classifying new inputs. The  $k$  closest vectors in the training set to the input to be classified (the  $k$  nearest neighbours) and identified using some distance metric (typically standard Euclidian distance) and a majority vote is taken. The category most represented

in the nearest neighbours is the predicted category for the input. The main hyperparameter for this algorithm is the value of  $k$  - the number of neighbours to look for. An odd number is usually preferable, so that for binary classification problems there will always be a winner of the majority vote.

K-Nearest Neighbours illustrates a problem that is present in many machine learning algorithms, the so-called ‘curse of dimensionality’. That is, the more dimensions a dataset has, the harder it is to create effective, predictive models. This is not just because of the extra computational burden. Higher dimensional data is more ‘sparse’; the distances between objects gets larger. In the case of K-Nearest Neighbours, this can be a problem because the nearest neighbours are further away, and the distance of the nearest neighbours approaches the average distance between data points as dimensionality increases, reducing the information content in those neighbours.

The above supervised learning methods are established algorithms for classifying inputs into categories using single functions. They are widely used, and have been used with considerable success in bioinformatics problems, including the identification of zinc and other metal binding sites (see below). However, there are limits to the discriminatory power of a single function, linear or otherwise, and hence limits to the complexity of the underlying system they can model. Their ‘capacity’ — the space of possible functions they can assume to model reality — is limited.

Rather than utilising a single function, artificial neural networks (ANNs) by contrast use nested functions, each with its own set of parameters, to capture different layers of abstraction in the input data. There is an input layer, which maps the input vector to some intermediate vector which represents some aspect of the input data. One or more hidden layers then acts on these intermediate vectors, eventually passing the data to the output layer which produces a vector representing the model’s degree of certainty that the input belongs to each of the possible categories. Each layer acts as a kind of perceptron with a non-linear activation function — if they were all linear then the model as a whole would be linear. This architecture is a neural network, and this is the classical implementation, with

every layer fully connected to the layer before it. When there are multiple hidden layers, the model is a kind of ‘deep’ learning.

Training a neural network requires finding the optimal values for the weights and threshold for each layer. Using the usual technique of defining a cost function and the differentiating this over the various layers would be very computationally difficult to do with the necessary precision. Fortunately a shortcut exists called backpropagation, which allows the neural network to be trained with much fewer computational resources. This heuristic for efficiently calculating the gradients of the nested functions of a neural network is in large part what underpins the current proliferation of deep learning applications.

There are variants of the artificial neural network model outlined above which are better served for different kinds of tasks. Convolutional Neural Networks (CNNs) are a widely used variant that are optimised for processing images and other input spaces that can be thought of as regularly spaced grids in  $n$  dimensions. Rather than having every layer fully connected to the layer before it, there are a number of ‘convolutional’ layers in which the perceptrons only connect corresponding localised regions, followed by normal fully connected layers at the end. Rather than standard matrix multiplication, a mathematical operation called convolution is used to determine the forward flow of information through the network.

Another variant is the Recurrent Neural Network (RNN). In standard neural networks, the input data is unordered, and each input is treated in isolation. The network ‘forgets’ what it has just done and treats each input like it was the first. However often datasets have inherent order, and knowledge of what the last input was classified as (or what the last  $n$  classifications were) contains important, usable information for classifying the current input. Recurrent neural networks are usually used for sequential data therefore, and are therefore of use in dealing with biological sequences (see below). They operate by adding an extra term relating to previously classified inputs.

Deep Learning is a kind of representational learning — the model itself is learning the features in the input space that are important. This can be seen when processing images with CNNs — the model’s first layer may look for low-level properties of the image, such as regions of high contrast representing borders, the next layer

may look at simple combinations of these, and later layers will be sensitive to more high level combinations of visual features. This is inspired by (though does not work in precisely the same way as) the mammalian visual cortex.

### 2.1.3 Evaluation

A trained model will (in this case) map potential binding sites to one of two categories — zinc binding or not zinc binding. To determine how useful a model is, or if it still needs improving, there needs to be a measure of how well the model is performing. There are a number of metrics for determining classification model performance.

To evaluate the model, a number of test inputs are given to the model, and for this purpose, the model can produce one of two outcomes, true or false, so all of the test results fall into one of four categories: true positives (correctly identified as a binding site), false negatives (was a binding site, but not detected), false positives (not actually a binding site, but wrongly identified as one), and true negatives (not a binding site, not identified as one). The first and last categories are the ones we want to optimise for, the other two the errors.

The simplest means of grading a model is with accuracy: the total number of correct predictions (true positives and true negatives) as a proportion of all test inputs. This can be a misleading metric however, if the dataset is class-imbalanced — that is, if one class is much more prevalent than the others. Accuracy is misleading here, because a model which just always predicts the more-prevalent class can get a very high accuracy, despite being completely useless as an identifier of rare states.

$$\text{accuracy} = \frac{TP+TN}{TP+FP+FN+TN}$$

Similarly the error rate — the total number of incorrect predictions (false negatives and false positives) as a proportion of all assignments — suffers from the same problem, and should be treated with the same caution in a class-imbalanced problem like this.

Two more meaningful metrics are precision and recall. Precision (or Positive Predictive Value) is the number of true positives, as a proportion of all predicted positives; that is, how many positive predictions were correct? Recall (also referred to as sensitivity) is the inverse, the number of true positives, as a proportion of all actual positives; that is, how many actual positives were identified? A high precision means that false positives are being kept down, with the reported positive cases being mostly correct. A high recall means that false negatives are being kept down, with most of the positive cases being identified.

$$\text{precision} = \frac{TP}{TP+FP}$$

$$\text{recall} = \frac{TP}{TP+FN}$$

However these two metrics exist in a balance, with improvements in one causing a decline in the other. The more ‘eager’ the model is to assign a positive prediction to an input, the more of them will be identified and the higher recall will be, but a smaller proportion of the predicted positives will be correct, lowering precision. Likewise, the more resistant the model is to predicting a positive label, the higher precision will be, but the lower recall will be.

$$F1 = \frac{TP}{TP + \frac{1}{2}(FP+FN)}$$

Another important metric is specificity, often contrasted with recall/sensitivity. This is the proportion of inputs the model reports as negative which are actually negative, and is a measure of the model’s ability to correctly ignore negatives.

Usually it is convenient to have a single value representing the effectiveness of the model, which is done using the F1 Score. This is simply the ‘harmonic average’ of precision and recall — the ratio of their product and sum, multiplied by two.

Another single value measure of model effectiveness is the Matthew’s Correlation Coefficient, or MCC. This takes into account all four values in the confusion matrix, and has certain advantages over the F1 score in that it ‘*produces a high score only if the prediction obtained good results in all of the four confusion matrix categories*’ [58].

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP+FP) \cdot (TP+FN) \cdot (TN+FP) \cdot (TN+FN)}}$$

There are metrics which show not how effective the model is in its current state, but how its ability to distinguish one class from another varies over different thresholds. Most classification algorithms can assign a probability of being in one class or the other, and the actual prediction is made by setting some threshold — if the probability is above this threshold, the positive case is predicted. If not, the negative case is predicted. A ROC Curve will show the model's effectiveness changes as this threshold is changed, by plotting false positive rate against true positive rate for every threshold value (to some given resolution). Purely random classifiers will produce a diagonal line - as the threshold is increased, both true positive predictions and false positive predictions will increase by equal amounts because the model can't actually distinguish one class from another at all. The better the model is at making this distinction however, the more this line will curve into the top-left corner, because true positives will increase much faster than false positives in a 'good' model.

This graphical ROC curve can be represented in a single metric called the AUC (Area Under the Curve). As the name suggests, this is the area underneath the ROC curve, which will be 0.5 for a purely random model, and 1 for a perfect model (see Figure 2.3). The reason a perfect model has an AUC of 1, is because a perfect model will have some threshold which produces a true positive rate of 1 and a false positive rate of 0, meaning it will touch the top left corner of the plot. Again this is a measure of how easy it is for the model to separate the two classes.

Another means of evaluating a model is examine how its performance increases with increases in training set size — learning curves. In this analysis, the model is trained in random subsets of the training data of increasing size, and the training and test scores taken at each stage — the score used depends on what is being assessed. This is useful in assessing the rate of convergence (how the model's training performance is approaching the test performance) and in determining whether a model would benefit from more training data.

Another pair of properties of models that is important to monitor is their bias and variance. These are both measures of the inadequacy of the model, but represent different things. A model's bias is the error that results from it making incorrect assumptions about the thing being modelled — it might be too simplistic

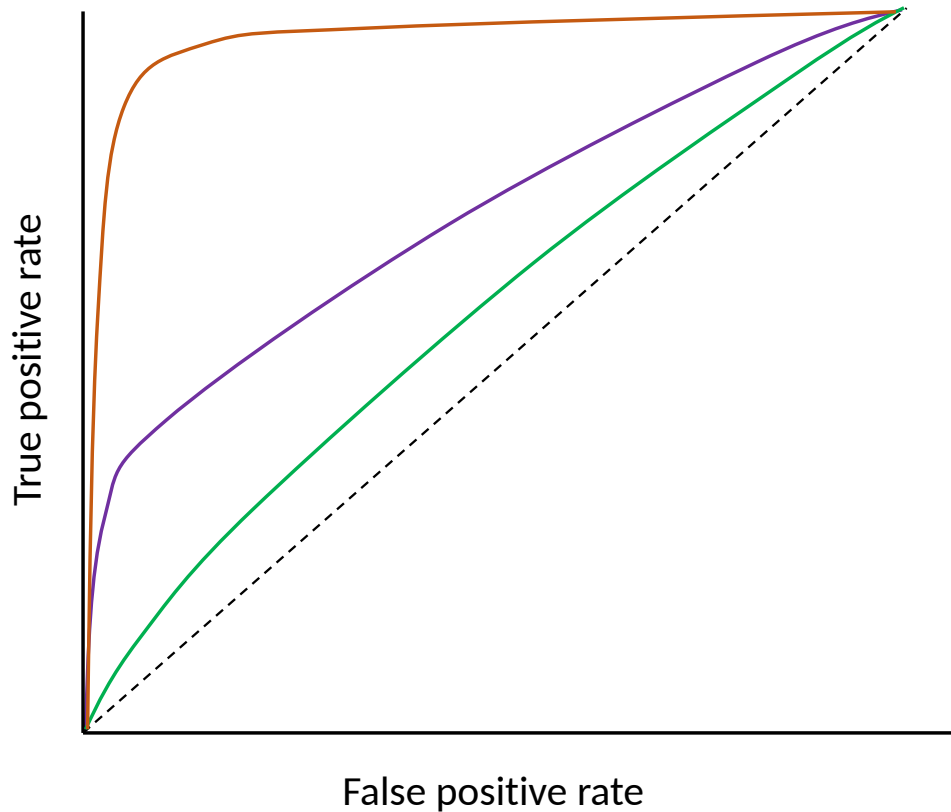


FIGURE 2.3: Area Under Curve — three different classifiers are shown here, with the green model being little better than random allocation, the orange model being near perfect, and the purple model being intermediate.

for example. Variance by contrast is error caused by being too sensitive to the specific data points used to train it.

This is related to the concepts of underfitting and overfitting respectively. A model which is underfit does not have enough parameters to properly represent the function it is modelling, and a model that is overfit is too attuned to the training data, which is after all only a sample. An overfit model will perform very well on the data it was trained with, but poorly on inputs it hasn't seen before — it has 'learned its training set.' A common way to detect this is to train the model

on a subset of the available data — typically around 80% — and then measure the model's performance on the remaining test set.

Any measures taken to reduce overfitting are called regularisation. These include general best practices, such as the previously mentioned practice of evaluating a model on a different dataset from the one used to train it, as well as algorithm specific measures, such as (in the case of Decision Trees/Random Forests) reducing the number of splits the tree is allowed to make.

Machine learning has been successfully applied to many areas of Bioinformatics in recent decades, with particular successes in the last decade. It has been used for the specific application of predicting zinc and other metal binding sites since there was enough data to construct the initial minimal datasets in the 2000s, and it is worth reviewing the progress that has been made here.

## **2.2 Machine Learning to Predict Zinc Binding**

There are still many proteins whose function is unknown. In many cases we only know their sequence, though some of these have had their structure solved experimentally. One means of determining what function a protein might perform is to look for certain properties associated with a given function in already characterised proteins, and then look for these properties in uncharacterised proteins. If, for example, it can be shown that such a protein had one or more zinc binding sites, that could offer insights into what function the protein performs. Indeed, that is ultimately the goal of this project.

Owing to this clear usefulness, there have been a number of attempts to develop such predictors in the past, dating back to the early 1990s. These methods and techniques form the essential context in which my own project should be understood, and also provide the benchmark of success against which this project's methods should be judged. Here their progress will be reviewed. They can essentially be divided into two broad categories - prediction from structure and prediction from sequence.



### 2.2.1 Prediction from Structure

Predicting zinc binding from structure takes the atom coordinates of some protein as its input, and produces as output residues in that structure which are believed to be able to bind zinc if zinc were present, generally along with some estimated probability that this should be the case.

This at first seems to be of only limited utility, as very often proteins which can bind zinc tightly will have zinc bound when the structure is solved, and so no prediction is required. Where such methods are of use however is when a protein capable of binding zinc is crystallised without zinc being present, presumably because the researchers didn't know it could bind zinc and so did not provide any in the medium. These apoproteins can be scanned by predictors like this to identify unfilled zinc binding sites.

The earliest attempt at a general purpose prediction algorithm was the Hydrophobic Contrast Function, developed in 1990 [10] to detect metal binding sites in general. This began from the observation that metal atoms tended to be surrounded by a shell of hydrophilic atoms, which in turn were surrounded by an outer shell of hydrophobic atoms. They developed a function which took as its input a coordinate in space, and returned a measure of this 'hydrophobic contrast' at that location. They showed that by scanning every point in the structure on a grid, evaluating the function at every point, and then ranking the points by their score (the function returned values between -1 for inverted contrast and 1 for desired contrast), the highest scoring points would be clustered at sites of metal binding.

This was the first demonstration that a property of metal binding sites could be used to create a predictive model. However it should be noted that it was only tested on structures with metals already present, not on apoproteins, and the initial observation itself was based on a very small number of metal proteins that were available at that time. It was later expanded upon by combining it with a template-based searching algorithm, whereby the known metal binding sites were represented as stripped-down 'templates' of just alpha and beta carbons, and used to search proteins using a simple pattern matching algorithm [11]. Once a set of three residues matching a known template was found, the hydrophobic contrast

function was applied to verify that the binding site was a region of high contrast, which it generally was. This was part of a pipeline for engineering binding sites into proteins, so it didn't require the matching residues to be of any particular type, but the general principle of using known site geometry and hydrophobic contrast properties to look in new structures was still in use.

The use of physical properties such as hydrophobicity has in some cases been taken to the logical extreme through the use of molecular dynamics, and the full-scale modelling of the interactions between protein residues and a hypothetical metal atom and the use of empirical force fields which can describe the attraction that a small molecule should experience towards a region of a protein. However, the standard force fields developed for molecular dynamics tend not to model metal-protein interactions well, so specific force fields for this branch of chemistry need to be developed. One such force field, the Fold-X empirical force field, was adapted for use on single atom ligands such as water and metal atoms [59]. Not only were they able to 'decorate' protein surfaces with identified metal binding sites, but the implied affinity of the binding sites reported by the force field correlated closely with experimentally reported affinities where available.

Generally however, many systems for predicting zinc binding in structures use geometric searches of some kind – particularly the early attempts. In 1997 a database of 'structural motifs' was built, as well as a simple algorithm for searching a structure for them called TESS (Template Search and Superposition Algorithm) which used hash tables for fast lookup [60]. While not specifically targeted at finding metal binding sites, the small structural motifs that it specialised in are particularly applicable to small ligand binding sites.

Simple geometric templates were eventually used to specifically identify metal binding sites again in the 2000s. Calcium binding sites have been successfully predicted by simply looking for clusters of oxygen atoms and identifying their centre [61] for example. In 2008, Goyal and Mande built up more complex templates using pairs of residues to identify previously unknown sites — from a now much larger starting dataset of structures. While this attempt was a general purpose metal binding site predictor, it was largely trained on zinc binding sites as they found that all metal binding sites had broadly similar geometric distributions

of alpha and beta carbons. This observation is important in distinguishing zinc binding sites from other metal binding sites.

One of the first template-based techniques which looked at zinc binding sites specifically was the TEMSP method, developed in 2011 [62]. Here again the problem is simplified by creating a library of ‘residue pairs’, where for each zinc binding site obtained from the PDB, they store each pairwise combination of liganding residues (actually just certain properties of them, such as alpha-alpha distance etc.) and then search the target protein for these. Once one is found, other template pairs are superimposed on the structure to find additional liganding residues. They used a stricter definition of ‘true positive’ than previous attempts, and still reported a sensitivity of 86.0% and a specificity of 95.9% in their test dataset, barely lower than the values obtained when testing on the training dataset used to refine their parameters. When applying their tool to a dataset of proteins of unknown function, they found some very promising results, finding 186 probable zinc binding sites. Unfortunately, the online version of this tool is no longer accessible at the time of writing.

The use of geometric techniques to identify both zinc and other metal binding sites has continued up until the present day, despite the increase in interest in more complicated machine and deep learning techniques over the 2010s. In 2013 a straightforward database survey of high quality zinc binding proteins (low resolution, high occupancy, low B-values) produced geometric templates that identified zinc binding sites at rates comparable with other methods — though like many of those other methods they restricted themselves to sites made of the four common residues cysteine, histidine, aspartate and glutamate, the CHED residues [63]. The ubiquity of these residues inspired the name of a similar tool around this called ‘CHED’, which again relied on geometric templates to predict zinc binding sites specifically — though with an extra filtering step for removing false positives [37].

As has been noted, while much attention has been paid to predicting whether a region of a protein binds zinc or some other metal, discriminating which metal binds to a region of a protein known to bind a metal is less well studied, and not straightforward owing to the similar preferences that transition metals tend to have in terms of the geometry, distance and chemical nature of their liganding

atoms. A geometric approach was used on this problem in 2015 with the algorithm mFASD, which examines the atoms in contact with the metal centre to create a ‘functional atom set’, creates a library of known functional atom sets for particular metals, and determines the set’s ‘distance’ (not Euclidian distance, but rather a measure of the difference between two data structures) from each characteristic set — the functional atom set distance (FASD) [64]. Using just this information they were able to get AUC scores ranging from 90% and 50% (i.e. random), depending on the metal pair in question, and demonstrated the feasibility of distinguishing the binding sites of different metals. The lowest scoring pair was manganese and iron, at 50.6%, which had previously been shown to have similar coordination preferences and tend to have weak binding affinities for their binding sites [65]

The most recent purely geometric approach to predicting metal binding is from 2019, which uses known metal geometry rules rather than a survey of known protein structures particularly [66]. Their method, called GaudiMM, is notable in that after making the side chains in the test set ‘flexible’, their success rate only slightly decreased — an important property of any system that proposes to find metal binding sites in apoprotein structures.

The research outlined above did not use machine learning techniques, but rather simple geometric templates and other rules based on observed properties of zinc and other metal binding sites in proteins. What distinguishes them from those which do use machine learning is not the use of geometry as such — geometric attributes are often part of the featureset fed to machine learning algorithms — but rather the use of a human to determine what features to pay attention to. Machine learning algorithms by contrast, while they vary widely in implementation details, are generally left to determine by themselves what features are important and how to use the data given to them to classify objects.

MetSite was one of the earliest uses of machine learning techniques to predict binding sites as we would use the term today, in 2004 [67]. They used artificial neural networks to detect metal binding sites in low-resolution structural models, such as those generated computationally from sequence data. They set themselves a target of having a false positive rate no higher than 5%, and using this threshold were able to get an accuracy of 94.2% and, more informatively, a recall of 60%,

across all sites. These figures fall to 84% and 39% for specific metals, from which they infer that many metals in the structures used are labelled incorrectly. They show that their model can predict metal binding in proteins of unknown function, and in models generated computationally, because the features they use do not use information from the precise positions of liganding atoms, but from outer residues and secondary structure.

An important development in 2008 showed that decision trees and support vector machines could be used to identify metal binding sites in apo-protein structures. This was an important proof-of-concept as it demonstrated that in the absence of the metal atom, the liganding residues are still sufficiently close to their liganding conformation to allow them to be identified from atom positions. A similar study in the same year showed that Bayesian classifiers (models which use training dataset to progressively apply Bayes' rule of probabilistic inference) alone could identify zinc binding sites specifically, using only concentric shells of atoms around the site as features [68]. This latter paper makes the point that Bayesian classifiers have a distinct advantage over more complex machine learning methods in that the model is less of a black box — you can follow the decision making of the classifier more easily.

Another use of Random Forest from that same year also produced reasonably encouraging results — an AUC of 0.8 [69]. Random Forests are particularly suited to measuring the importance of the features that are given to the model, and in this case they found that accessible surface area and the level of evolutionary conservation of the residues in question were particularly important properties to examine. This was also another example of a model whose efficacy remained largely the same when apo-structures were examined. This finding was repeated in a similar piece of research in 2012 [70].

There have been uses of deep learning in recent years however, with encouraging results. In 2019 a system in which an ensemble of different classifiers were used as the inputs to a neural network was developed quite successfully — demonstrating the power of using multiple classifiers trained in different ways [71].

There has also been some attention given to the problem of detecting half binding sites — regions of a protein chain's surface which would form part of a binding site

if another half binding site is present. In 2011 the mutation in the complement factor H protein which causes them aggregate was investigated by a number of experimental means, as well as with a bioinformatics approach [72]. Here the approach was largely one of molecular modelling, creating multiple dimers of the protein in various conformations and looking for arrangements in which there was no steric clash and in which an arrangement of residues is made which could bind zinc (they used METSITE for this). While they didn't find a definitive zinc binding site in this way, they did rule out a site which they had strongly suspected of being responsible beforehand. Note also that this is a search for a full zinc binding site between two specific proteins — it is not a search for a half binding site on one protein. That is, it relied on knowing what the other 'half' of the binding site would be, at least at the whole protein level.

There have also been some purely experimental investigations into this phenomenon, such as the demonstration that zinc mediates the oligomerisation of the Alzheimer's Amyloid Precursor Protein, shown through FRET analysis [73]. In general however, the examination of finding half binding sites in protein structures has been very understudied.

### **2.2.2 Prediction from Sequence**

A system for identifying zinc binding residues merely from the sequence of residues it contains is arguably much more useful, as protein sequences are much more readily available. It allows for whole genomes to be scanned for zinc binding proteins. As such, there has been much effort in the past to create techniques for predicting zinc binding from sequence.

The earliest research in this area was largely a 'manual' effort, given the small number of sequences available. The discovery of zinc finger proteins in 1985, for example, involved the observation of repeating C...C...H...H patterns in the proteins being studied, and the hypothesis that these were binding zinc [14]. Similarly, zinc binding in the E6 and E7 subunits of a Papillomavirus was predicted from the observation of cysteine repeats [74]. Slightly more complex patterns were later identified, such as the observation that enzymatic zinc binding sites frequently

have two residues close to each other in the sequence, and then a third much more distant — the ‘short and long spacers’ [12]. This kind of manual examination of individual protein sequences is obviously not scalable to the vast numbers of protein sequences now available, and can only identify relatively obvious binding sites in any case — but the same principles that were used in these early studies of characteristic repeats largely underpins the automated techniques now in use.

As the number of sequences grew, identifying zinc binding sites through the use of sequence alignment to known zinc binding sequences became possible. This was used to identify plausible sites in Protein Kinase C [75], in *Giardia lamblia* surface proteins [76], and in the Alzheimer’s Amyloid Precursor Protein (APP) [77]. While this sort of manual inspection for characteristic sequences would largely be superseded by more automated methods from the mid 1990s onwards, it is still occasionally a useful feature of research when a single protein is being researched in detail — being done recently to identify a novel zinc binding protein in yeast from an observed CXXCXXXXXCXXC pattern [78].

One of the ultimate advantages of prediction from sequence over prediction from structure is that it allows you to quickly and efficiently search very large datasets for evidence of zinc binding. Again, early efforts relied on sequence alignment — in 2004 every then-known metal binding sequence was encoded as a series of residue codes and gap sizes (referred to as ‘metal binding patterns’), and these were then BLAST searched against entire genomes of various species [79]. The same team went on to study zinc binding in the human genome specifically using a refined version of this technique and assert, based on this method, that the human genome encodes 2,800 +/- 400 zinc binding proteins — about 10% of the believed protein encoding genes at the time[1]. Shortly afterwards they expanded their analysis to representative genomes from eukaryotes, prokaryotes and archaea, and found that this proportion varies from 4% to 10% — with eukaryotes having more due to higher numbers of regulatory proteins [80].

Largely however, as with prediction from structure, the bulk of research efforts have focused on the use of machine learning methods. One of the first uses of traditional machine learning methods was in 1995, when neural networks were employed to predict zinc finger proteins from sequence data [81]. They hardcoded

the specific C...C...H...H spacer pattern they were looking for into the model, rather than having the model learn it from existing sequence data by encoding it as a binary feature, and also incorporated features for electric charge, hydrophobicity, and side chain types. Overall this was largely successful; they reported an accuracy of 97%, though for a very specific type of zinc binding sequence that was largely hardcoded in from the start. Another early use of neural networks to predict metal binding using neural networks was in 2005, when a one-hidden-layer network was used to predict a variety of metal binding sites in sequence [82]. Though they did not look at zinc specifically, it, like the previous example, demonstrates that it is possible to build effective models — with high sensitivity in this case — from comparatively small amounts of data (at least compared with the volume of sequences available now).

Both SVM and neural networks were used to predict the metal binding cysteine and histidine residues of sequences by the same group that developed the metal binding pattern concept — though it relies on the sites present in the searched sequence belonging to previously identified motifs, as it is these previously identified patterns that their models were trained with [83]. They would later extend this to use inter-cysteine distance information [80].

Another use of SVM models from the same year reported a sensitivity of 74.9% and a specificity of 98% for zinc, using a variety of features, and found that of all the features they looked at, hydrophobicity, solvent accessibility and polarity were the most important features used by the model in predicting zinc binding from sequence [84]. The combination of raw sequence data with homology modelling was validated using another SVM model in 2008, in a tool called ZincPred [85]. They reported high scores, but it should be noted that they had a relatively forgiving metric of success (at least one residue in a binding site correctly identified) and, like many sequence based models, it only predicts at the residue level — that is, the inputs are individual residues, and the model predicts whether that one residue is metal/zinc binding or not, rather than identifying discrete binding sites. SVM models were widely used throughout the 2010s, most recently in 2018 [86], with broadly similar results.



Random Forest has also been used successfully in the prediction of zinc binding sites from sequence, again relying heavily on Position-Specific Scoring Matrices [70], [87] — an advantage of which is the relatively straightforward means of determining which features are the most important in making correct predictions.

In keeping with the structural models, the use of ensemble methods — where multiple models are trained and then vote — has also been used, in the case of sequences for a model called ZincExplorer [88]. An interesting example of this was in 2017, when three previous zinc binding site predictors — ZincExplorer, ZincFinder, and ZincPred — were used to predict zinc binding from sequence, and then the output of these individual models were fed into a novel linear regression model, yielding very high results [89].

In the past couple of years the use of deep learning techniques has begun to have a more pronounced effect on the field. In 2019 an alternative to molecular docking — where the binding affinity of a ligand and protein is calculated by simulating the correct pose/position of the ligand on the protein, and then calculating the energy of binding from that position — was proposed called DeepAffinity [90]. This uses mostly recurrent neural networks to remove the need for these separate steps, and simply predicts affinity as a function of the structure alone.

Other uses of deep learning to predict metal binding from sequences include combining convolutional and recurrent neural networks to predict metal binding histidines and cysteines (which found the mutation rate feature to be most significant) [91], a novel means of representing protein sequence using continuous vectors, which they apply among other things to the task of metal binding site prediction [92], and a use of a technique called ‘extreme gradient boosting’, which predicts ligand binding sites generally at a similar success rate as previous techniques, but faster [93]. This is a modification of the gradient boosting algorithm, which has also been used to predict metal binding residues from sequence, albeit largely relying on structural attributes predicted from the sequence [94].

Machine learning has therefore been used successfully in the prediction of metal binding sites generally and zinc binding sites specifically. One recurring feature that should be pointed out in closing, particularly in the sequence-based predictive models, is the focus on zinc binding *residues* rather than zinc binding *sites*. In most

cases, the entity examined by the predictive model is the individual residue, often with a surrounding linear sequence ‘window’ of residues. The model then assigns a probability as to whether that residue is a zinc binding residue. As outlined above, this approach has had a measure of success, but it is a somewhat artificial concept. There is, after all, no such thing as a zinc-binding residue in isolation. The individual residues of a high-affinity zinc binding site of the kind considered here are only zinc-binding when the other residues are present, and conversely many non-zinc-binding residues could bind zinc if other residues were present in the correct locations. It is particular *combinations* of residues, not individual residues, which are zinc binding — an important fact not usually considered in research of this kind.

Another commonality is the treatment of zinc binding sites as a single category, and the presumption of properties that are common to them all regardless of the residues of which they are comprised. This may well be sufficient — particularly as there are essentially only four residues that make up the vast majority of zinc binding sites — but it is possible that properties used for prediction have much tighter distributions within particular sub-categorisations of zinc binding sites.

These are both crucial issues, which this PhD addresses.

## 2.3 Sequence Techniques

As part of the dataset generation process, sequence clustering is used. This takes a set of sequences (polypeptide sequences in this case) and creates a non-redundant set from them where very similar sequences are grouped together, with a similarity cutoff used to determine which sequences should be considered similar enough to group. This has the advantage of ensuring that sequences which are overrepresented are not given undue weighting in analysis simply because they are present in more copies. The program CD-HIT is used to perform this analysis [95].

BLAST (Basic Local Alignment Search Tool) searching is also used in this project, which is an algorithm for searching through a database of protein (in this case) sequences for those which align to a query sequence. It uses heuristics to perform

this search very quickly. It is ‘local’ alignment in the sense that the query is not just aligned to the whole target sequence — if it has a better alignment with a substring of the sequence, that will also be identified, and also uses substrings of the query sequence [96].

## 2.4 Web Technologies

A major part of this project’s utility to the wider scientific community is the presentation of the data and tools created for it to the world. All of them are available via the web, and various web technologies are used to accomplish this.

### 2.4.1 Django

Django is an open source Python web framework, used to create database-backed web applications [97]. All database population scripts are written in Python, and use the Django Object-Relational Mapping utilities to populate the database.

### 2.4.2 GraphQL

GraphQL is an API technology and query language which allows users to construct requests for precisely the data they need [98].

Traditionally, web APIs have used the REST architecture, whereby every meaningful object is given a URL address. The problem with REST that GraphQL seeks to address is that REST APIs often involve overfetching (you get the full representation of an object whether you want it or not) and underfetching (you have to make multiple requests to get the data you need). With GraphQL APIs, you define your objects using a graph of relationships, and then request exactly the data you need.

GraphQL APIs are used here to provide computational access to both the dataset itself, and to the predictive models.

The Python/Django library graphene is used here to implement the GraphQL schemas and API services [99].

### 2.4.3 React

While the web services that provide computational access are useful for programmatic access, for browsing the data initially a website frontend is much more useful. In order to avoid duplication of logic, a Javascript framework which connects to the existing API is used here: React.

React uses individual components to build up a single page web application, and the React router package is used to create individually addressed pages.

Other JavaScript technologies used on the frontend are the 3D macromolecular visualisation package NGL (used for visualising the zinc binding sites) [100] and the charting library Highcharts (used to give an overview of the data).

## 2.5 Conclusion

Predicting zinc binding sites is a binary classification task — all potential sites in a protein, whether that is defined as a location in space or a combination of residues, must be assigned to the category of ‘true zinc binding site’ or ‘non zinc binding site’, using information about what distinguishes these categories from a dataset of confirmed zinc binding sites.

This is a task for which there are a variety of well established machine learning techniques applicable, the basic theory, advantages, and scope for customisation of which have been outlined here. There is a history of such techniques being applied to the problem, the results of which have which have also been reviewed here — though few if any are open source and/or still publicly available over the web as they were at the (often quite recent) time of their initial publication.

These are the computational techniques which will be used to build the predictive models. To make these models available to researchers, a variety of modern web

technologies are used. Before any predictive models can be utilised however, the dataset which will ultimately form the training data must be built.

## Chapter 3

# atomium — The Python Structure Parser

*Note that some of the material in this chapter has been published in the journal *Bioinformatics* under the title ‘atomium—a Python structure parser’ (2020) pp. 2750–2754. [101]*

Much of this project relies very heavily on reading in structure data from files deposited in the Protein Data Bank.

Traditionally these files have been stored and distributed in the form of PDB (.pdb or .ent) files. These are text files that consist of a list of records, with each record being limited to 80 characters. Information is stored in these records at fixed offsets from the start of the line. This file format has a number of limitations, most notably the fact that they cannot store more than 100,000 atoms because only five characters are allocated to atom IDs, and so they cannot go past 99,999.

Consequently, the Protein Data Bank also provides files in the newer mmCIF (.cif) file format [102], and indeed, structures with more than 100,000 atoms are *only* available in this file format.

There are a number of parsers available that can handle these file formats, and which were considered for use in this project, both in Python and other languages. However, for reasons that will be outlined shortly, it was decided to create a novel Python parser. This parser is called atomium.

## 3.1 Rationale

The tool used to read structure files is of such critical importance to this project, that it was important to ensure that the correct tool with the correct *capabilities* was used. Specifically, the tool would need the following properties:

- The ability to read .cif files. Some structures are *only* available in this form as they contain too many atoms to fit in a single .pdb file, so .pdb parsing alone would lead to an incomplete (though still very large) coverage of the available structures.
- The ability to generate biological assemblies from the raw, asymmetric unit coordinates of the file. As will be outlined below, the structure actually described by these files is often not entirely representative of biological reality, and certain transformations must be performed before analysis can be done.
- The ability to extract certain items of descriptive data from the files. In addition to the actual coordinates of the structures' atoms, the files also contain 'meta' information about the file, details of the experiment that produced them and the species the molecules came from, quality information such as resolution, and sequence information not obtainable from the model itself (which often contain missing residues). This information was needed to annotate a local database.

Probably the major Python PDB parser at time of writing is BioPython. This is a multi-purpose bioinformatics library, with many modules for dealing with general biological data formats. One of these modules allows for parsing of PDB structure files, and is frequently used in Python analyses of protein structures. However, BioPython's PDB reader does not support the processing of Biological Assemblies — it can only give the user the asymmetric unit as a model. As this project relies on having that functionality as a core part of the parsing process, this makes using BioPython difficult. It does meet the other requirements in that it can parse .cif files and read the metadata, but biological assembly generation is too crucial for the library to be considered.

It was therefore decided to create a novel Python library for handling biological structures, which would focus solely on structural models (BioPython has many other disparate functions). An additional advantage to doing this was that it gave a much greater insight into the structure of these files, and of structural biology concepts generally.

## 3.2 Data Structures

The object returned by the various parsing functions is a `File` object. This essentially represents the actual file object itself rather than its structural contents, and has the ‘metadata’, descriptive attributes referred to above. Some key attributes are:

- `code` The PDB identifier of the structure.
- `title` An overall description of the structure.
- `technique` The experimental procedure used to generate the data, such as X-Ray Diffraction, NMR, etc.
- `source_organism` The organism that the molecules were originally derived from. Note that in cases where the file lists multiple species, only the first is used.
- `expression_system` The organism that the molecules were actually expressed in. Again just the first is assigned to this property if there are multiple.
- `resolution` A measure of structure quality.

There are many others, and the full documentation (<https://atomium.bio>) lists them all.

The two most crucial attributes of the root `File` object are `model` and `models`. A `File` object has one or more `Model` objects, representing the actual structures themselves, and these are accessible via `models`. Because most files just contain a single model, the `model` attribute points to the first `Model` object as a convenience.



The most fundamental structural object is the **Atom**. All structures are ultimately amalgamations of atom objects arranged in space, and all higher structures are just collections of atoms. An **Atom** object has attributes for its Cartesian coordinates, its element, its unique numeric ID, its name, and various other pieces of information extractable from PDB structures. Even before they are grouped into structures, **Atom** objects can do various feats based on these simple properties. The mass of an atom can be determined from its element, its distance to or angle with some other atom can be determined using their locations, they can be translated and transformed through space, etc. All of these are performable using methods of the **Atom** object.

Ultimately though, structures are not modeled as mere clouds of atoms, they are grouped into various molecules and other higher order structures. In atomium, the two objects which are direct containers of atoms are **Residue** and **Ligand**; the former is used when the structure of which they are part is part of a chain, and the latter when it is not. They both have properties for their ID and name, and residues additionally have methods pointing to the next and previous residue in the chain they are part of.

They also both have an attribute which points to their atoms, but these are not stored using a Python built-in object type such as list or set. Instead they are stored in an atomium-specific object called a **StructureSet**, which is optimised for storing atomium structure objects. In many respects this works in a very similar way to an actual set, but internally the objects are stored as a Python dictionary, where the IDs are keys, and the values are lists of objects with that ID. This makes lookup by ID much faster than it would otherwise be which is absolutely crucial in large structures.

Residues themselves are stored in **Chain** objects. This implements the Python iterable tools, so they have a length, allow access to residues by index, and can be iterated through in **for** loops. They store their residues in a **StructureSet**, but they are returned as an ordered tuple.

Both **Chain** and **Ligand** inherit from the class **Molecule**, which in atomium means they are ‘top-level’ structures as far as a model is concerned. That is, a model is a collection of chains and ligands. In fact a **Model** object is initialised by passing in

chains and ligands, with the ligands then being divided into water and non-water ligands, as this is often an important distinction.

The relationship of the classes is outlined in Figure 3.1.

Models, chains, residues and ligands all inherit from the class `AtomStructure`. This is an abstract base class that any atom-containing class can inherit from, and provides many useful functionalities using those atoms. It allows a structure's mass or charge to be determined by summing the masses or charges of the atoms, allows for translation or rotation of whole structures, the generation of atomic formulae using Python `Counter` objects, and RMSD (Root Mean Square Deviation — a measure of how different two structures are in the layout and location of their atoms) comparisons with other structures. The full functionality is described in the atomium documentation.

The structure classes also make use of an atomium metaclass called `StructureClass`. Essentially this confers on any object created from these classes, a set of lookup methods, allowing them to search the structures in their `StructureSets` by any property they might have, or any derivative of those properties. For example, chains can return any residues whose name matches a particular regex pattern, and models can return any atoms with a mass greater than some threshold. Earlier versions of atomium had these properties manually coded in, but the current metaclass-based implementation ensures that all properties, all derivatives of properties, and any future properties can be instantly and rapidly searched (see Table [table:filtering](#)).

These search functionalities all come in both single result (where a single structure or `None` is returned) and multiple result (where a set of results is returned, even if empty) versions.

Finally, all structures have an awareness of their context. That is to say, all atoms know which residue or ligand they are in, which model they are in, and so on, as do higher structures. This allows for neighbour searching. For example, atoms and higher structures can both look for atoms within a certain cutoff and which match certain criteria — useful in, for example, a project that relies on identifying binding sites.

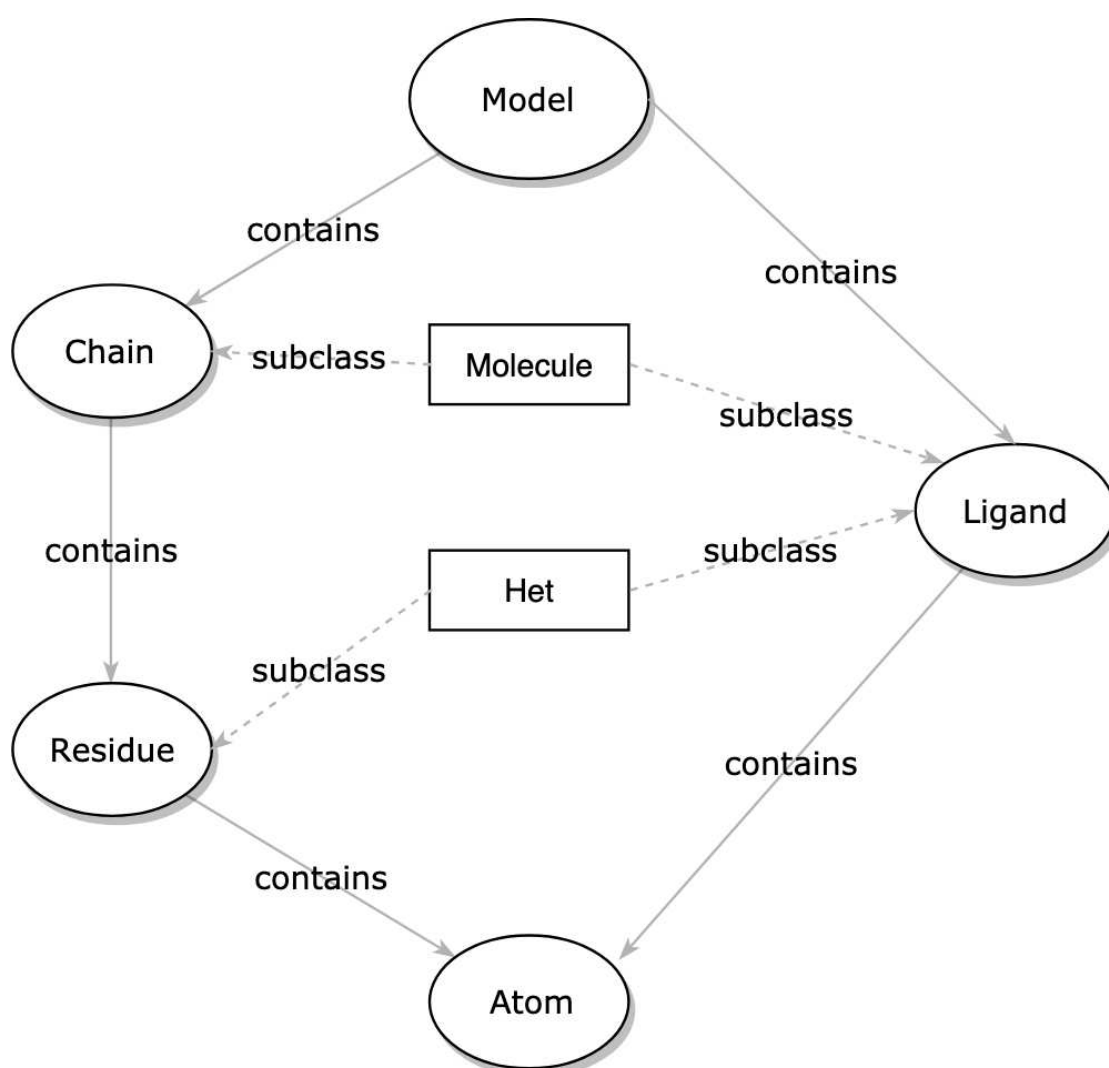


FIGURE 3.1: The relationship of structure classes in atomium, representing the hierarchy of types. While the structures can be created from scratch, this hierarchy has been designed to reflect the hierarchy of object types in PDB and mmCIF files.

Command	Result
<code>model.atoms()</code>	All atoms
<code>model.atoms(element='N')</code>	All nitrogen atoms
<code>model.atoms(mass_gt=14)</code>	Atoms with mass greater than 14
<code>model.atoms(name_regex='CA CB')</code>	CA and CB atoms
<code>model.atoms(het_name_regex='CYS HIS')</code>	Atoms in cysteine and histidine residues
<code>model.atoms(chain_length_lt=100)</code>	Atoms in chains shorter than 100 residues

TABLE 3.1: Examples of the filtering syntax that all atomium structures have by virtue of implementing the `StructureClass` metaclass.

### 3.2.1 Distance Optimisation

While atomium is a general purpose PDB parser, some of its features are of particular significance to this project. An example of this is that of proximity detection, which allows atoms to identify nearby atoms within a cutoff. Higher structures can find nearby higher structures, but ultimately all of this is still relying on atoms searching for nearby atoms. As alluded to earlier, this is of particular importance in this project as a zinc atom's liganding atoms are determined by distance (and element). Any optimisation that can be done to this process is therefore important.

This process particularly needed optimisation as it is such a significant bottleneck in the generation of the dataset. To identify nearby atoms, an atom gets the full set of atoms in the containing model and iterates through them all, calculating the distance between that atom and itself. Those that fall below the given cutoff are returned. The algorithm which calculates distance is not particularly complex (it is just the Pythagorean equation extended to three dimensions) but when it is performed many times the burden can become considerable as the number of atom distances to calculate increases with the square of the number of atoms in the structure for every proximity search. In large models with many thousands of atoms, finding an atom's nearby atoms means getting its distance to thousands of other atoms. For each metal atom in a structure, the metal's nearby atoms must be found, and for each binding site, each atom in each liganding residue must also find its nearby atoms to determine stabilising contacts in the secondary shell. Large structures also tend to have many binding sites, meaning that some PDBs would take more than a day to fully process.

To resolve this problem, atomium has been given the ability to speed up this process at the expense of greater memory consumption. An atomium model has an `optimise_distances` method which, when called, organises the atoms into a grid. This is implemented using the Python `defaultdict` data structure, which is a mapping that creates a key if a new key is used, rather than throwing an error. These are nested three-deep so that each of the three dimensions is represented, and all the atoms in a single 10Å cube are stored in one container. Once this grid is created, whenever an atom needs to search its neighbours, it uses its own location to work out which entries in the grid to look up using its own coordinates

as keys, and then only searches these. The number of adjacent grid compartments to search is determined by the cutoff being used in the proximity search.

This optimisation can reduce the time taken to process a large PDB by three orders of magnitude.

### 3.3 Parsing

atomium can read .cif, .mmtf (a compact, stripped-down, binary representation of mmCIF files optimised for transmission over the web), or .pdb files. In each case the overall process is the same.

1. Get the file contents as a string.
2. Determine which filetype it is by looking at the file extension or, if not possible, by looking at file contents.
3. Convert the filestring to a Python dictionary whose structure is specific to that file type.
4. Convert that dictionary to a standard atomium data dictionary, whose structure is the same regardless of the file type origin.
5. Convert that data dictionary to an atomium `File` object with one or more `Files` within it.

This process for parsing has a number of advantages over just trying to go from filestring to processed Python object in one step. By making the parse process of the three filetypes converge at one data structure — the atomium data dictionary — it prevents duplication of effort involved in going from ‘data’ to ‘Python structure’. It also means that every file can have a consistent, dictionary representations, which means that they can all be represented as JSON if desired. It is also easier for testing, as each stage in this (relatively complex) parsing process can more easily be tested in isolation.

### 3.3.1 File Contents and File Type Detection

Getting the file contents as a string is a straightforward process: it is simply opened and read using the built in Python functions for that purpose. The process is abstracted into a single function which will first try to read the file as a unicode string (appropriate for .cif and .pdb) and then as binary data (appropriate for .mmtf).

atomium also enables ‘fetching’ a file from a remote server. Generally this is via HTTP, from the RCSB web servers [57] with a four letter code, though the user can select any URL. The Python library requests [103] is used for this. Alternatively the user can fetch a file over SSH using a different atomium function.

Once the file contents are extracted as a string (or bytestring), atomium then determines which of the three file types it has been given, so it knows which functions to use to process it further. This is done using a straightforward algorithm: if the file extension is given, this will be used to determine filetype. If not, or if the filetype is not one of the three it recognises (such as .ent for example), atomium will assume it is .mmtf if the data is binary, .cif if it is a unicode string that contains the string `_atom_sites`, and .pdb otherwise.

The parsing time for the three file formats in atomium are of a similar order of magnitude, with .cif taking the longest (see Figure 3.2). In all five cases, the relationship between the number of atoms and the time taken to parse is linear and, for all comparisons, care was taken to ensure the same kinds of parsing were being done—no biological assembly generation, proper relationship parsing and assigning for the sub-structures, etc.

### 3.3.2 File Dictionaries

The next step is to convert this string into a Python dictionary that reflects the internal structure of that file type.

.cif files are essentially a list of connected tables: tables in the sense that each block has a list of headers, and then one or more lists of values that belong to

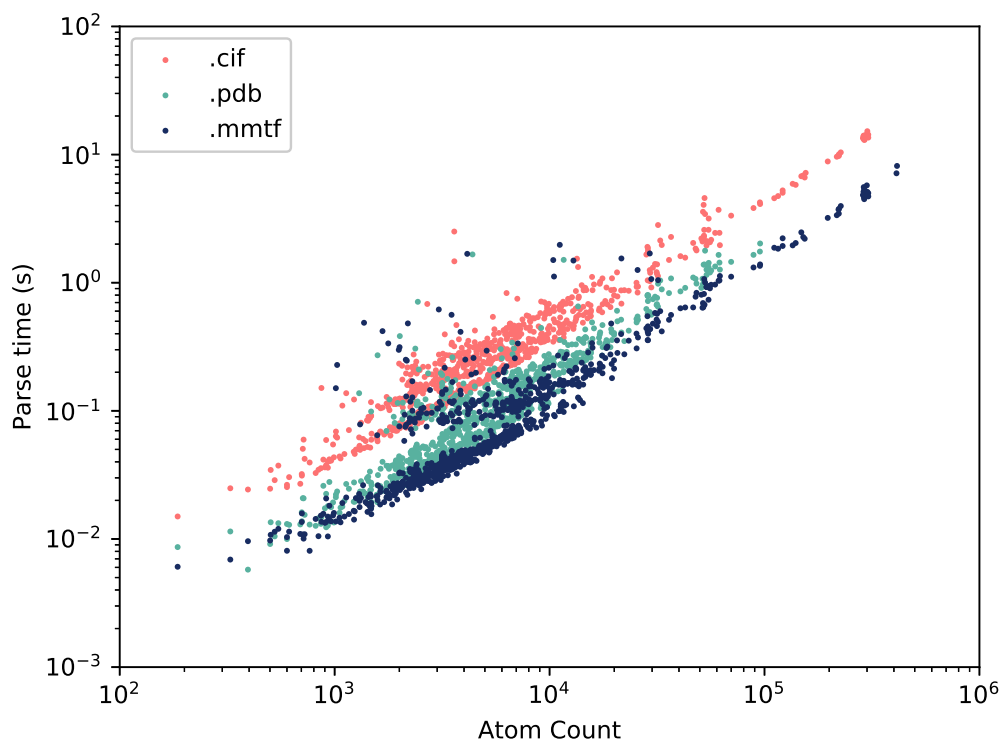


FIGURE 3.2: Time taken to parse the same 1000 randomly chosen single-model structures in the three file formats using atomium. Time as a function of atom count is linear and it can be seen that mmCIF structures take the longest, followed by PDB structures and MMTF structures.

those headers, and connected in the sense that a row may have an ID as its first value which is referred to in another table. atomium represents these tables as lists of dictionaries, where the headers are keys, and the row cells the values. Each of these table lists is a value in the top-level dictionary. The main bottleneck in this process is splitting values on a single line — they are white-space separated, albeit with whitespace within quotes ignored, and with both quote types permissible and escapable within the other. Initially the Python library `shlex` was used to do this splitting, but its speed was just too slow so a custom function was written which was faster, though still the slowest part of the process.

.mmtf files are binary encoded Python dictionaries, so once the Python `msgpack` library has decoded it, it is essentially already in dictionary form. The only extra steps that need to be taken are to decode the .mmtf formatted binary fields within this dictionary, using the algorithms specified in the .mmtf documentation and implemented in atomium.

.pdb files have much less internal structure than the other two, and are essentially just lists of records. These records can be grouped by record name however, and this is what atomium does. The file dictionary has record names as keys, and lists of records that belong to that record type as values. The only exceptions are **REMARK** records, which are further grouped by remark number, and all model related records (**ATOM**, **TER**, **HETATM** etc.), which are grouped together because their order relative to each other is information that would be lost if they were split up into record types (**TER** records separating chains being the main reason).

This resultant file dictionary, whose internal structure is specific to file type, is essentially just a Python representation of the file contents — no ‘parsing’ is done as such, it just gets the information in a form that later processes can understand.

### 3.3.3 Data Dictionary

This file dictionary is then turned into an atomium ‘data dictionary’. This is a dictionary with a consistent internal structure regardless of what the original file type was, and essentially consists of a number of sub-dictionaries.

For example there is the description sub-dictionary, which contains meta information about the file such as title and deposition date. There is an experiment sub-dictionary, with information about how the model was created. The quality sub-dictionary contains resolution and other related information. The geometry sub-dictionary contains transformation matrices, such as those needed to create biological assemblies.

The most important one is ‘models’, which is actually a list of sub-dictionaries. Each one contains information about the chains, ligands and water molecules in that model, containing all the information needed to create an atomium **Model**.

Each of the three file dictionaries have their own functions for extracting information from them and building a data dictionary. In the case of .cif and .mmtf file dictionaries, this is relatively straightforward, as much of the data has already been extracted in the previous step, and this is largely just a question of transferring data from one dictionary to another, and nesting atoms within the right structures.



For .pdb files however, the previous step just grouped records, and the data is still locked away within these strings at pre-defined offsets, so for this filetype it is this stage which is the bottleneck, as it takes longer to extract the necessary information.

The parsing process is summarised in Figure 3.3.

### 3.3.4 Model Generation

The structure of the data dictionary is optimised to make model creation as rapid as possible. The polymers, ligands and water molecules are already segregated as they will be in the model, and all names and IDs are already parsed. Relevant `Model`, `Chain`, `Ligand` etc. molecules are created from this information.

The surrounding `File` object is made from the rest of the data dictionary with the meta information, with the values of the sub-dictionaries becoming properties of that object. The `Model` objects exist as a list within that `File`, with a pointer to the first `Model` if the user only needs the first.

Biological assemblies can be created using a `generate_assembly` function. In earlier versions of atomium, each individual molecule was transformed separately, but in very large structures this can be time consuming. Instead, the copies are created, and then the atoms of all the structures are transformed in one step.

## 3.4 Saving

One of the more unusual features of atomium is that it allows changes to models to be made and then for those changes to be saved in *any* of the three file types that atomium supports.

Unlike the process of parsing, in which the structures pass through a number of different representations reflecting the different layers of abstraction of that process, saving is much more straightforward. A function takes in an atomium `AtomStructure` (it does not need to be a model — chains etc. can be saved individually) and turns it into the relevant filestring.

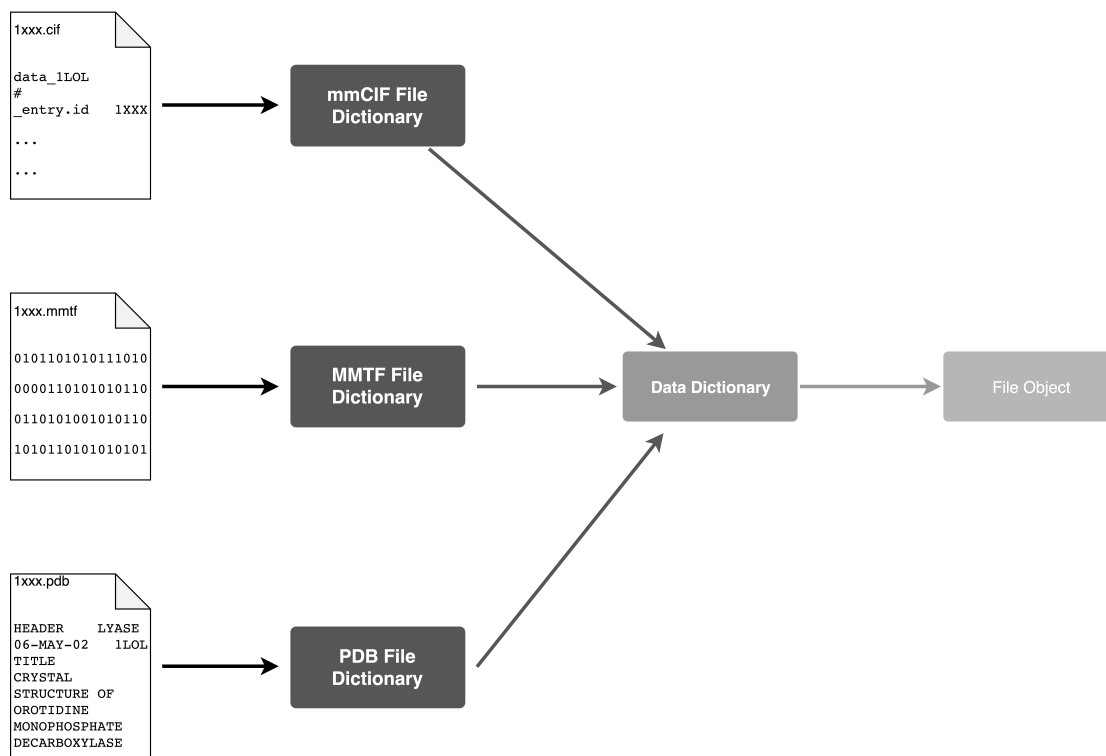


FIGURE 3.3: An overview of how parsing of the three file types is done. Because the three file types converge on a single data dictionary, all three file types can be represented as JSON.

Only structural information is saved. The various file annotations that are parsed from the original files are presumed to be properties solely of those original files, and not of any modified versions of them that might exist. So, atom records are saved, along with anything else needed to construct details of the actual structure model, but not titles, deposition dates, resolution etc.

## 3.5 pdb2json

As already noted, the process of parsing a structure file involves turning the raw filestring (or bytestring) into two successive Python dictionaries, before then being turned into a Python object. Initially this choice of Python dictionary as internal representation was a decision made to make development easier.

However, Python dictionaries have a structure very similar to JSON, a data format that is frequently used for sending data over the web in a very human readable way, as it is essentially just nested key-value pairs. Therefore, atomium is also

potentially a tool for turning any PDB structure into JSON, if the data dictionary were simply converted to JSON using Python's built-in `json` library.

To this end, `pdb2json` was created. This is a simple, lightweight Django web app which uses `atomium` to take any PDB code and return the resultant structure as JSON. This is done using a URL - `/2S0D/` will return the JSON for the PDB `2S0D`, for example.

It is currently available at <https://pdb2json.samireland.com/>.

By default, `pdb2json` tells `atomium` to use the `.cif` representation for parsing, but this can be altered using, for example, `/2S0D.pdb/`. The structure of the JSON returned will be the same, as `atomium` creates the same data dictionary regardless of file type, but some values might be different. For example, many PDBs have titles etc. in ALL CAPS whereas `.cif` files use title case, and atom IDs may be numbered slightly different. Generally the actual semantic content should be the same across all file types though.

If the user so wishes, they can get the initial Python dictionary as JSON (the one that is specific to that file type) using `/2S0D/?file` notation. This is generally of limited interest in the case of `.pdb` and `.mmtf`, except as a means of checking the original file contents pre-processing, but in the case of `.cif` it can be very useful. This is because *every* attribute of the structure will be accessible in this dictionary, so if the subset of attributes `atomium` pulls out of files to annotate its final representation isn't enough, they can be obtained from this representation. For example, `atomium` File objects have `rvalue` and `rwork` attributes, but there are many supporting metrics for these calculations in the original file.

Obviously, PDB structures can be quite large, with some containing hundreds of thousands of atoms. The user may not wish to download the JSON for an entire structure when they only need a single metric or set of metrics. Therefore, `pdb2json` allows the user to traverse the keys of the JSON structure using the supplied URL. For example, while `/2S0D/` will return the JSON for the entire structure, `/2S0D/quality/` will return only the `quality` sub-dictionary that was part of the original JSON object. This traversal can be as deep as the user wishes:

`/2SOD/models/0/non-polymer/0.153/atoms/4382` will return information about a single zinc atom, for example.

Note however that while this will reduce the volume of data that will be sent over the network, on the server the structure is still being fully processed. So this feature can save on bandwidth, but not really on request speed.

`pdb2json` then, is a lightweight web server for providing users with JSON representations of any PDB structure, acting as a thin wrapper around `atomium`. It was not necessary to the overall project or even to `atomium` itself, but it was felt the ratio of work needed to benefit provided to the wider community was so low that it would be a useful exercise to create. It means that researchers no longer require a PDB parser installed to get basic structural information — just an HTTP client, such as a browser. It is language agnostic; most programming languages will have JSON libraries as part of their standard library, whereas PDB processing libraries like `atomium` or `BioPython` are much more specialised pieces of software. It also means that all structures now have a much more human readable format that is easily viewable.

## Chapter 4

# ZincBind — The Database of Zinc Binding Sites

*Note that some of the material in this chapter has been published in the journal Database under the title ‘ZincBind—the database of zinc binding sites’ (2019) [104], though the values and analysis have been updated for the updated dataset.*

This PhD focuses on developing novel means of predicting Zinc Binding Sites using the known properties of previously identified Zinc Binding Sites. As such, the initial step was to create a dataset of these already-known sites.

This is an undertaking that has been performed previously on a smaller scale, several times (see Chapter 1). One of the primary reasons that the effort has been duplicated so many times is because in none of the previous dataset generations did the authors make their data publicly available in an easy-to-use resource. Therefore from the very beginning of this project, the intention was always to not *just* create this dataset of Zinc Binding Sites, but also to make this database publicly available via a web resource, that would be continually updated with new sites as they become available.

This chapter will describe the creation of that dataset, and the web application that offers users access to the data: ZincBind.

## 4.1 Data Generation

ZincBind uses as its primary data source the Protein Databank [17]. This contains almost two hundred thousand protein structures, a subset of which contain zinc atoms and which can be inspected to see if a zinc binding site can be identified.

The RCSB web services [57] allow a user to query the entire databank by, among other things, the chemical formulae of its small molecules. The PDB IDs of all zinc atom containing structures can therefore be obtained by issuing a request to these web services for a `ChemCompFormulaQuery` with the formula `Zn`.

If the dataset is being created from scratch, each of these PDB IDs is iterated through in turn to look for zinc binding sites. If the dataset is merely being updated with new structures, only those IDs that do not already exist in the database are used.

### 4.1.1 Structure Inspection

The algorithm iterates through each PDB code and, for each, determines what zinc binding sites are present, if any, and saves the relevant objects to the database once its analysis is complete.

Each PDB ID is analysed within a single database transaction, meaning that the SQL statements are built up by Django throughout the analysis and then executed all at once at the end of each PDB ID analysis. This has two benefits. Firstly, it makes the whole process much faster — there are database records for each atom in the binding site, so opening up database connections, creating a record, and closing the connection multiple times per structure is much more time consuming than just doing that once at the end. Secondly, and more importantly, if the program is interrupted in the middle of building a database, it will essentially rollback to the last fully processed PDB, rather than saving records for half-processed PDBs which could cause errors when the program is restarted.

The structures are requested from the RCSB web servers in the mmCIF file format. This, and all subsequent structure parsing and analysis, are abstracted into the atomium Python library (see Chapter 3).

The first processing step with the obtained and parsed structure, is to generate a biological assembly. The raw coordinates of the PDB structure contain the ‘asymmetric unit’, which is often a subset or superset of the ‘true’ biological assembly. Each PDB structure contains a list of possible biological assemblies that can be created from the chains in the asymmetric unit to make a more biologically realistic structure (see Chapter 1), and each is associated with a list of transformation matrices and calculated metrics. The  $\Delta G$  of the assemblies are used to rank them, and the lowest  $\Delta G$  assembly that still contains zinc is selected as the ‘real’ biological assembly.

This step is critical. While asymmetric units are perfectly suitable if you are merely concerned with intra-chain features, they are often unsuitable for examining the interfaces between chains. Since many zinc binding sites are between chains, relying on asymmetric units would produce data that is not particularly meaningful. An example of this would be insulin, which is a hexamer of six chains and two zinc binding sites, with each chain providing one histidine residue to one of the two binding sites. For many insulin structures, the asymmetric unit contains just one chain in this hexamer and suggests that the ‘binding site’ is a single histidine residue. Only by using the lowest energy biological assembly is the full hexamer constructed, and the true binding sites made available for analysis.

However it is important to keep in mind that, necessary though biological assembly generation is, it does introduce some problems. Many assembly instructions are simply to use a subset of the chains in the asymmetric unit, which is fine, but some require copies of chains to be created. For example, a structure that starts out with an A chain and a B chain may, in its biological assembly, have two A chains and two B chains. As there is no established protocol for generating new IDs for these chains (and since creating one would create inconsistencies with other software that uses the data in the ZincBind database, such as the NGL protein viewer), the solution used here is simply to have multiple chains with the same

ID. This can cause a number of problems which will be outlined, along with their solution, as they arise in this section.

Once the correct assembly has been generated, the algorithm then checks that the resultant structure is usable, and is not just alpha carbons as some structures are. If it is such a ‘skeleton structure’, the zinc atom(s) in that structure are saved to the database with an annotation explaining that they do not have a binding site because the PDB was unusable. Note that this is the case whenever a zinc atom is rejected — it is still saved to the database with an explanatory annotation, because by doing this *every* zinc atom in the Protein Data Bank can be accounted for in ZincBind. If this were not done, and a user tried to find a particular zinc atom from the data bank in ZincBind which had not been assigned a binding site, that user would have no way of knowing if the zinc atom was absent because ZincBind had not examined that PDB, or if it was absent because a binding site could not be assigned to it.

Any zinc atoms that are in the asymmetric unit but not the biological assembly (common in cases where the asymmetric unit is just the biological assembly repeated multiple times) are also saved to the database without a binding site, with an explanatory explanation as to why it has none, for the same reason.

The next step is to actually identify all of the binding sites. This is done as follows:

1. Identify all metals in the structure. This is done using element names in atomium, but it is here that the first biological assembly related problems must be dealt with. When these assemblies are being generated, sometimes atoms end up superimposed on top of each other — so called ‘special position’ atoms. The correct interpretation of this is that there is just one atom here, not multiple atoms occupying the same position in space, so duplicates have to be removed. This is done with atomium by taking all atoms of a given element, comparing it with every other atom of that element, and if the distance between them is less than 1 Å, one is removed.
2. Determine the atoms which ligand each metal in the structure. All atoms within 3 Å of the metal which are not carbon or hydrogen are determined, and again duplicated atoms must be removed from these using the same



algorithm as above (insulin structures for example, often have a chloride ligand which would appear multiple times in the same location otherwise). Once this ‘cloud’ of suitable liganding atoms is obtained, they are ordered by distance to the metal, and for each one a check is made that the angle formed between it, the metal, and any closer atom is not less than  $45^\circ$ . If it is, the atom is removed from the set of liganding atoms as it is assumed a coordinate bond would be infeasible if there is another coordinate bond so close obstructing it.

3. Remove metals that probably aren’t physiologically relevant. Sometimes a metal atom in a model is just there as a consequence of the crystallisation environment, and while it may interact with one or two residues, those residues might not be a ‘real’ zinc binding site in the sense of being an evolutionarily selected combination of residues there specifically to bind zinc. There is no way to identify with certainty which category a given zinc atom falls into from atom coordinates alone, but ZincBind implements a filter whereby if a metal atom has fewer than three protein liganding atoms, it is discarded as probably irrelevant. Again, those atoms which are excluded at this stage are given an annotation explaining the reason, if they are zinc atoms.
4. Merge the metals into single binding sites where necessary. So far, all metals have been considered in isolation, though in reality some of them may form co-functional units with each other. To determine if this is the case, the residues/ligands that the liganding atoms are part of are examined and, if two metals share a residue/ligand, they are assigned to the same ‘cluster’.
5. Remove non-zinc binding sites. ZincBind is a database of *zinc* binding sites. So far all metals have been considered because some non-zinc atoms may be part of a multi-metal site that contains zinc, so had to be included at every step. However now that the fully processed sites have been generated, those that do not contain zinc can be removed.
6. Identify the secondary residues of each of the binding sites. This is the set of all residues which are within  $3 \text{ \AA}$  of a primary liganding residue. These residues are saved, along with the details of which atoms mediate these interactions. The importance of second shell residues in stabilising the

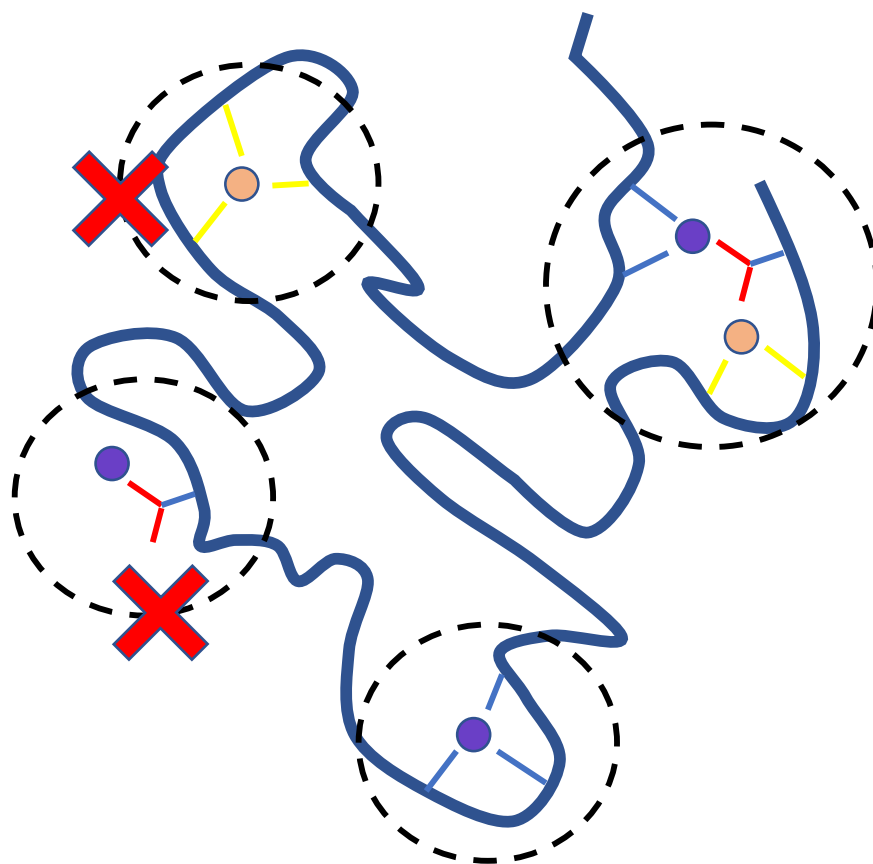


FIGURE 4.1: An illustration of how the ZincBind build algorithm identifies binding sites. Those with no zinc (top left) or too few liganding atoms (bottom left) are rejected.

primary liganding shell has been well documented, so identifying how many of these stabilising contacts exist is important information.

See Figure 4.1 for an illustration of this process.

Now that the zinc binding sites have been identified, they can be saved to the database. Before this is done however, the chain objects are saved to the database, as these records will be needed when the sites are saved. Any chain in the structure with a residue that is part of a binding site is saved. The sequence used is the full canonical sequence listed in the header of the structure file, not the actual sequence

of residues in the model, which may contain missing residues (particularly at the ends of the sequence). These two sequences are aligned using the Smith-Waterman local alignment algorithm [105] so that the liganding residues in the model can be identified in the full sequence, and capitalised.

*Then* the sites themselves are saved to the database, along with records for any metals, residues, and residue atoms. So-called ‘Chain-Site Interactions’ are also saved, which are essentially chains with only a single site annotated within them. While the plain chain objects saved above have residues involved in any binding site highlighted, here only the residues involved in that particular site are. This is important as chains may be involved in multiple sites, but predictive models need to know which residues contribute to *single* sites.

### 4.1.2 Equivalent Sites and Families

The database as it stands at this point in the algorithm will be a list of zinc binding sites, where each site is considered distinct and separate from all others. This is somewhat misleading, as many proteins appear in the Protein Data Bank several times, and so the binding site(s) in those structures will appear in ZincBind several times. The binding site in (for example) Carbonic Anhydrase will appear in ZincBind many hundreds of times, yet each of these records refer to the same biological entity.

To account for this, ZincBind clusters binding sites into equivalent sites called ‘groups’. The first step is to cluster the zinc binding chains (some of which will be associated with multiple sites, some of which will make up only part of individual sites) that were saved as part of the earlier generation process into clusters on 90% sequence identity, using the program CD-HIT [95]. It is assumed that chains in a CD-HIT sequence cluster are functionally equivalent.

The binding sites themselves are then clustered using these chain clusters. Two zinc binding sites are assigned to the same cluster if (1) they are associated with the same chain cluster(s), (2) they have the same residue names in the same order, and (3) their surrounding residue names are the same. These latter two steps are

important to ensure that chains with two or more zinc binding sites along them do not have those binding sites incorrectly assigned to the same cluster.

A group object is stored in the database for each of these clusters, and the corresponding zinc binding sites are linked with them. Thus ZincBind stores individual zinc binding sites as they occur in the Protein Data Bank, and unique zinc binding sites in the form of groups.

Binding sites and groups form two components of the three-part hierarchy of data in ZincBind, the third being the family. Each binding site is given an alphanumeric code based on its residue content and count — so, for example, binding sites with three histidine residues are ‘H3’, those with two histidines and two cysteine residues are C2H2, and so on. These make up the ‘family’ to which the binding site belongs. Binding sites are clustered into groups, which are in turn clustered into families based on residue content.

## 4.2 The ZincBind Web Resource

As already noted, the creation of this dataset was intended to serve both as the primary dataset of this project, and as a publicly available resource. This resource is called ZincBind, and is available over the web.

The database is accessible via a GraphQL API, which allows any specific object, or set of related objects, to be requested in a single HTTP request (see Figure 4.2). This is available at [api.zincbind.net](http://api.zincbind.net) and is the primary means of obtaining data from ZincBind in an automated fashion. As outlined in Chapter 2, this is created using the Python web framework Django (particularly its Object Relational Management system) and the GraphQL library graphene.

More accessibly for casual browsing, there is also a website available at [zincbind.net](http://zincbind.net). This is a lightweight web frontend that interacts with the database via the GraphQL API and presents it to the user graphically. It is a responsive, mobile-first layout created with the front-end framework ‘react’, that adapts to different screen sizes.

```

{
  pdb(id: "3W7Y") {
    title
    rvalue
    depositionDate
    zincsites {
      count
      edges {
        node {
          id
          family
        }
      }
    }
  }
}

```

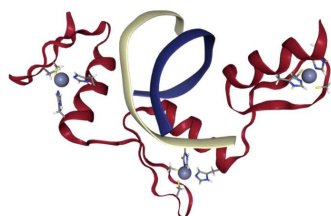
⇒

```

{
  "data": {
    "pdb": {
      "title": "0.92A structure of 22n human insulin at 100K",
      "rvalue": 0.16105,
      "depositionDate": "2013-03-11",
      "zincsites": {
        "count": 1,
        "edges": [
          {
            "node": {
              "id": "3W7Y-1",
              "family": "H3"
            }
          }
        ]
      }
    }
  }
}

```

FIGURE 4.2: An example of a ZincBind GraphQL API query, in this case for a PDB object. There requesting all the zinc binding sites within that PDB — there is one in this case. In this case the fields `id` and `family` are requested for each, though they also have other fields.



### Classified and Clustered

ZincBind identifies all zinc atoms in the PDB, determines their bind site where appropriate, and clusters them on sequence identity. Each site is then organised into groups and families.

FIGURE 4.3: The home page of ZincBind, showing the overview statistics in the top-left, and the quick search text box in the top right. The menu bar at the top collapses on small screen widths, making them accessible via a drop-down menu on mobile.

The home page (see Figure 4.3) contains a short, clear statement of what ZincBind is (a “database of zinc binding sites, automatically generated from the Protein Data Bank”) as well as a brief overview of its key metrics: number of unique binding sites, total number of binding sites, and the number of PDB files the sites are derived from.

The screenshot shows the ZincBind website's advanced search interface. At the top, there is a dark purple navigation bar with the ZincBind logo on the left and links for Search, Predict, Data, About, and Help on the right. Below the navigation bar, the page is divided into three main sections. The first section is titled 'Advanced Search'. The second section is titled 'PDB Search' and features a dropdown menu with 'Title contains' selected and a text input field containing the example text 'e.g. antibody, carbonic anhydrase'. Below this input field are two buttons: 'New Term' and 'Search'. The third section is titled 'Site Search' and features a dropdown menu with 'Structure Title contains' selected and a text input field containing the same example text 'e.g. antibody, carbonic anhydrase'. Below this input field are also two buttons: 'New Term' and 'Search'.

FIGURE 4.4: The advanced search interface, allowing searching of PDB files, or individual zinc binding sites.

Searching of the database can be done in one of four ways. Firstly, from the home page, the user can enter a search term that will be used to search multiple PDB fields at once: title, PDB code, classification, etc. This is for users who just want to find entries that have any relation to a given term at all, and provides an easy, low-friction way for a user to submit a search immediately.

Secondly, the user can navigate to the Advanced Search page (see Figure 4.4), where they can specify a more precise search by one or more PDB fields. This also allows searching by numeric cutoffs, such as for PDBs better than a certain resolution or deposited after a certain date — essentially any of the PDB metrics that are stored in the database.

Thirdly, via this same page they can search for specific binding sites rather than for containing PDB structures. The user can pick the same metrics as for PDBs (sites in PDBs better than a certain resolution, with a certain classification, etc.) as well as site-specific metrics, such as sites of a particular family, or containing particular residues.

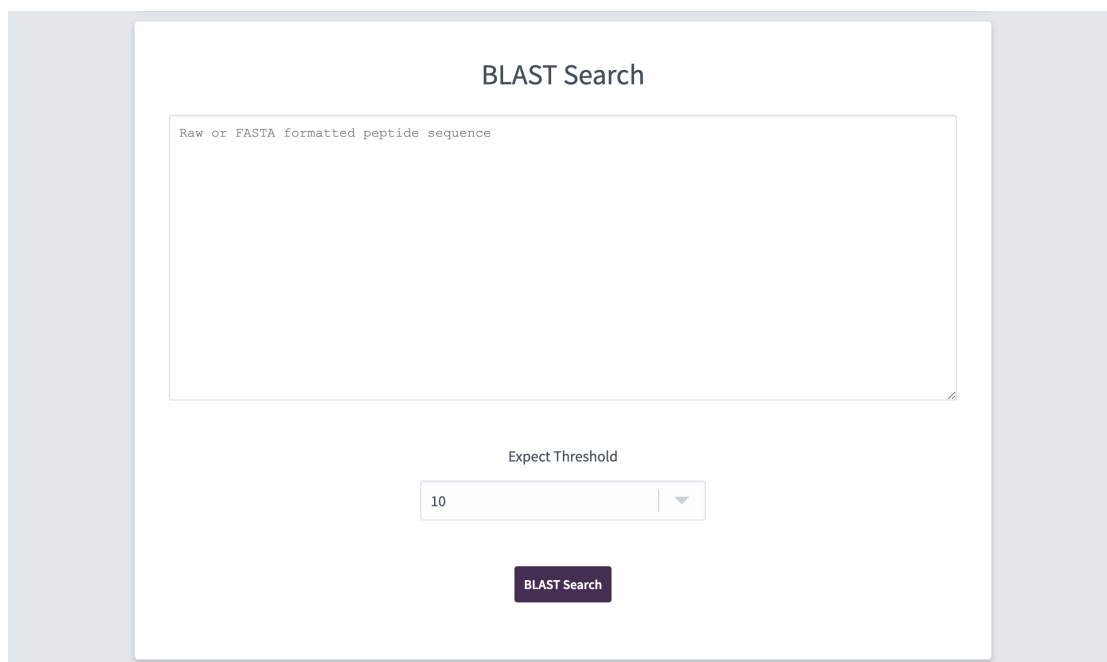


FIGURE 4.5: The BLAST search interface.

Finally, the user can provide a sequence and perform a BLAST search against the database (see Figure 4.5). This uses the NCBI `blastp` binary on the backend, whose JSON output is processed by python functions and exposed via the GraphQL API. This produces chain sequence results (see Figure 4.6), rather than PDB/sites as the other methods do.

The search results for the first two modes of searching are returned as a list of matching PDBs, with binding sites nested within them. Both the PDB and the sites themselves are clickable and will take the user to the page for that entity. Search results can be sorted by a variety of metrics, and are paginated to 25 results per page (see Figure 4.7).

PDB pages represent PDB records (see Figure 4.8). They provide basic information for that PDB (resolution, R-value, classification, source and expression organisms, experimental technique, deposition date, and keywords — the latter of which acts as a link to search by any of those keywords), as well as overviews of the zinc content of that PDB. The zincs associated with a binding site are listed as clickable links, while those without a binding site are listed along with their omission reason. Finally, there is a 3D manipulatable view of the PDB with all of its zinc binding sites. This uses the NGL Javascript library, and beside

The screenshot shows the ZincBind interface with a dark purple header containing the site name and navigation links (Search, Predict, Data, About, Help). The main content area is titled "BLAST Results" and features a search bar with the number "1" inside. Below the search bar, three search results are displayed in a table-like format. Each result includes the PDB ID, chain, E-value, score, and a brief description of the structure. The first result is 2QQW, Chain A, with an E-value of 7.33196 and a score of 61, describing the crystal structure of a cell-wall invertase from *Arabidopsis thaliana*. The second result is 4AV7, Chain D, with an E-value of 7.3589 and a score of 62, describing the structure determination of a double mutant from *Pseudomonas sp. dsm 6611*. The third result is 4AV7, Chain A, with the same E-value and score, also describing the structure determination of a double mutant from *Pseudomonas sp. dsm 6611*. Each result also includes a sequence alignment and a list of zinc binding sites (e.g., 2QQW-1 (D1H1), 4AV7-2 (D2E1H4)).

FIGURE 4.6: An example of BLAST search results.

The screenshot shows the ZincBind interface with a dark purple header containing the site name and navigation links (Search, Predict, Data, About, Help). The main content area is titled "Search Results" and features a search bar with a dropdown menu set to "Newest to Oldest" and two pagination buttons labeled "1" and "2". Below the search bar, two search results are displayed in a table-like format. The first result is 6VYO, dated 27 February, 2020, from *Severe acute respiratory syndrome coronavirus 2*, describing the crystal structure of the RNA binding domain of the nucleocapsid phosphoprotein. It includes the text "VIRAL PROTEIN | X-RAY DIFFRACTION 1.7 Å" and lists four zinc binding sites: 6VYO-1 (D1H2), 6VYO-2 (D1H2), 6VYO-3 (D1H2), and 6VYO-4 (D1H2). The second result is 6LZG, dated 19 February, 2020, from *Homo sapiens*, describing the structure of the novel coronavirus spike receptor-binding domain complexed with its receptor ACE2.

FIGURE 4.7: An example of paginated (over two pages in this case) search result showing matching PDBs, and the zinc binding sites they contain.



**ZincBind** Search Predict Data About Help

**Crystal Structure of SARS-CoV-2 Nsp16/10 Heterodimer in Complex with (m7GpppA2m)pUpUpApApA (Cap-1), S-Adenosyl-L-homocysteine (SAH) and Manganese (Mn).**

**Code:** [7L6R](#)

**Resolution:** 1.98

**Classification:** VIRAL PROTEIN/RNA

**Source Organism:** *Severe acute respiratory syndrome coronavirus 2*

**Experimental Technique:** X-RAY DIFFRACTION

**Deposited:** 2020-12-23

**R-value:** 0.1504

**Biological Assembly:** 1

**Expression Organism:** *Escherichia coli BL21(DE3)*

**Keywords:** Structural Genomics, Center for Structural Genomics of Infectious Diseases, CSGID, nsp16, nsp10, complex, VIRAL PROTEIN, SAH, Cap-1, VIRAL PROTEIN-RNA complex

**Zinc-Bearing Chains: 1**

**Chain B**  
1 binding site  
snagnatevpanstvlscfavdaakaykdylasggppitncvkmicthtgtgqaitvtpeanmdqesfggascClyCchidEppkgfCdikgyvqipttcandpvgtfltkntvctvcgmwkgygsc

FIGURE 4.8: An example of a PDB page in ZincBind, showing basic information for that PDB. The three-dimensional structure view is further down the page.

it is a set of basic options for manipulating the view, such as rotation options, background colour, and binding site highlighting — this is in addition to NGL’s built-in touch/drag/zoom manipulation. Generally the PDB page limits itself to information relevant to the structure’s role as a zinc binding protein, but a link to the equivalent RCSB page is also provided for more information.

The binding site pages (see Figure 4.9) contain more detail for that specific binding site, such as its mode of coordination, and links to equivalent sites in the same group. There is also a 3D manipulatable model of the binding site on this page — this is broadly the same as that on the PDB page, except zoomed in to the specific site and with selectable individual residues and metals. The page also contains a summary of information for the containing PDB, and a link to that PDB’s page.

ZincBind also has a page that gives an overview of the dataset as a whole (see Figure 4.10). This contains charts for residue distribution, experimental methods, residue codes, resolution, species frequency, and PDB classification. These are intended to give a sense of the key properties of the database ‘at a glance’. There are also links to a page containing all data (essentially a modified version of the search results page that shows all PDBs, paginated), a link to an overview of the

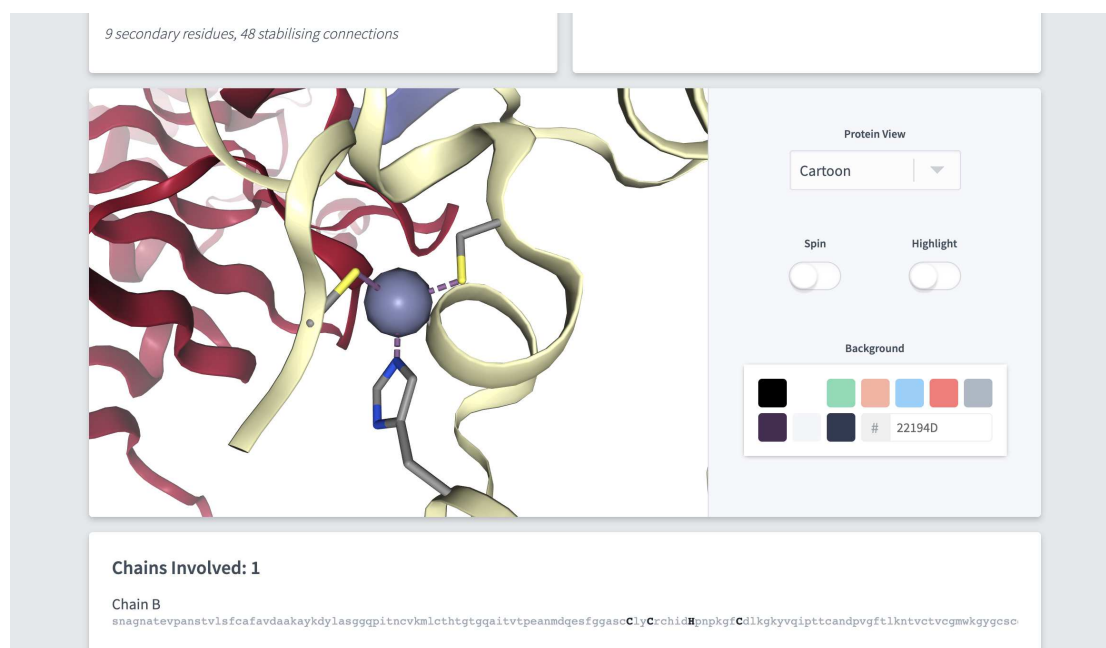


FIGURE 4.9: An example of a binding site page in ZincBind, with the manipulatable 3D NGL viewer, zoomed in to this binding site.

GraphQL API, and a link to a page showing how the sites are clustered into families and groups. Here each group lists the keywords and classifications that the sites in that group have in common to try to provide an automated way of annotating what the binding site the group represents actually does (see Figure 4.11).

There is a page dedicated to the prediction of binding sites that uses the separate prediction API, but this will be covered in greater detail in the next chapter.

Finally, ZincBind has a help page, structured as a series of questions that a user may have.

Ultimately, the website is intended to serve as a way of broadening access to the database, which is continually updated. The API provides access to the data for researchers who need bulk access to the data, whereas the web frontend allows an individual to explore the data more casually to get a sense of what research might be carried out with it.

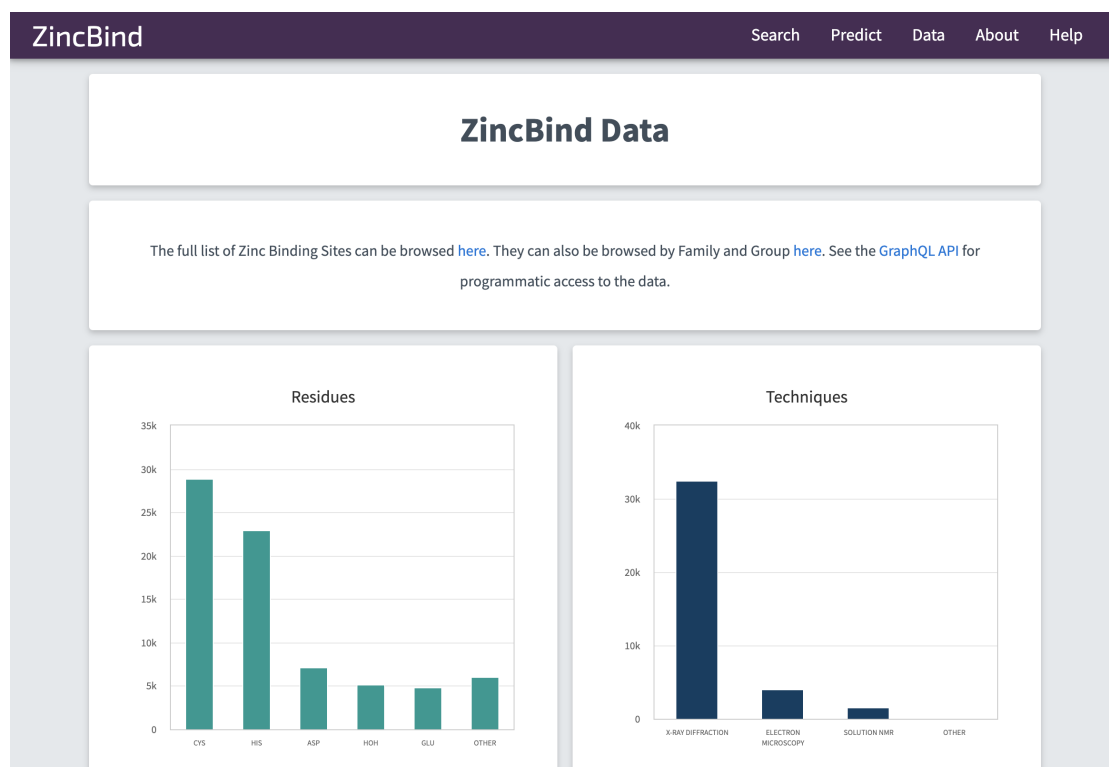


FIGURE 4.10: Overview statistics for the dataset presented on the ZincBind data page.

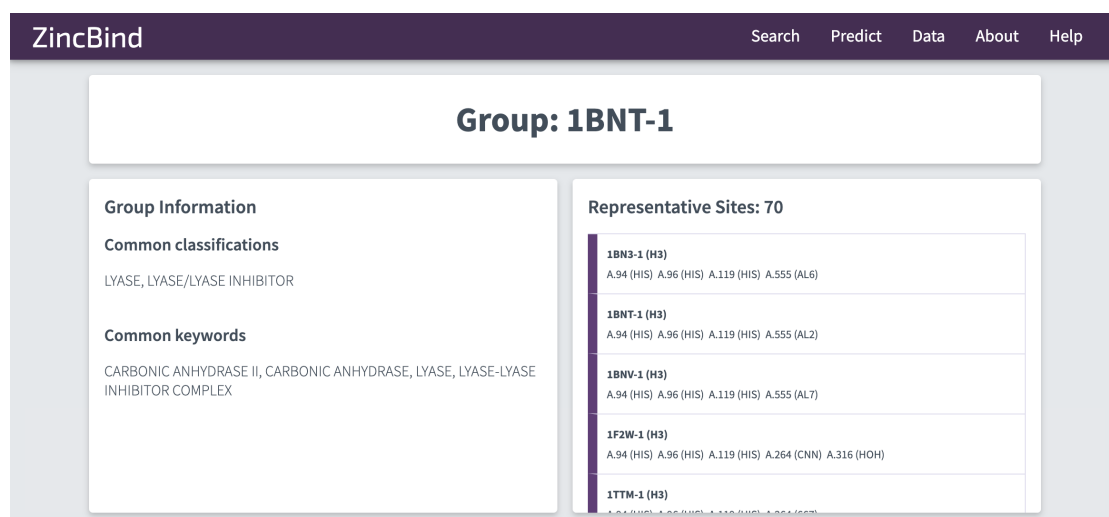


FIGURE 4.11: A ZincBind group page, summarising the clustered zinc binding sites and providing links to individual site pages.

## 4.3 Data Analysis

The finished dataset offers a useful insight into the general properties of zinc binding sites, or at least that subset of zinc binding sites for which structural information is available. Some of these properties are of use in building predictive models of zinc binding sites, whereas others are of more general interest. An overview of the main findings that can be gleaned from the dataset will be presented here. All values are correct at the time of writing (February 2021), though as this is a continually and automatically updated dataset, the precise values will vary as time goes on. It is not anticipated that these changes will alter the overall conclusions, as the dataset is very large already.

### 4.3.1 The Prevalence of Zinc

Of the 174,014 PDB structures in the Protein Data Bank at the time of the most recent database update, 17,556 contained at least one zinc atom and are accounted for in ZincBind. This proportion, 10.1%, is very close to the 10% of human proteins usually said to contain zinc, though as the Protein Data Bank does not purport to be a representative sampling of any genome, human or otherwise, there was no particular reason for this to be the case.

### 4.3.2 Qualifying Atoms

In total, when run in February 2021, ZincBind identified 65,595 zinc atoms and 1,138 other metal atoms associated with zinc atoms when searching the raw coordinates of the PDB asymmetric units. Other metals are only stored in ZincBind if they are part of a multi-metal binding site with at least one zinc atom, so the vast majority of metals in the database are zinc. Of the non-zinc metals, the most common of these ‘co-active’ metals are magnesium, potassium, iron and copper (see Table 4.1).

Of the 65,595 zinc atoms in the database, 24,246 (37.0%) are not associated with any binding site, and are in the database solely to acknowledge their existence.

TABLE 4.1: Prevalence of various non-zinc metals in ZincBind.

Metal	Count	Relative Percentage
Potassium	319	28.0%
Iron	188	16.5%
Copper	15	13.6%
Magnesium	132	11.6%
Sodium	100	8.8%
Calcium	90	7.9%
Manganese	75	6.6%
Cadmium	38	3.3%
Cobalt	31	2.7%
Nickel	6	0.5%
Vanadium	3	0.3%
Barium	1	0.1%

The most common reason for not assigning a zinc atom to a binding site is that the atom is duplicated multiple times in the asymmetric unit, but appears only once in the biological assembly used for processing; 13,124 metals were excluded for this reason. For example, PDB entry 1A4L contains four chains, each with one zinc atom, but the biological assembly only uses one. The other three are stored in ZincBind with an omission reason, but have no binding site assigned to them. Figure 4.12 shows the breakdown of zinc atoms by omission reason.

Using the full biological assembly leads to zinc ions being both excluded, and included as being associated with binding sites. As stated above, it is common for the asymmetric unit of a PDB file to contain multiple copies of the biomolecule of interest as a result of crystallization, but once the ‘correct’ assembly is chosen, these duplicated zinc atoms will be absent from the final structure. Conversely, using the full biological assembly rather than just the raw asymmetric unit coordinates of the PDB is crucial when the zinc atom is present at an interface between chains. In the asymmetric unit, there may be a single residue coordinating with the ion; if symmetry were not considered, such a model would be discarded by the ZincBind algorithm as a salt.

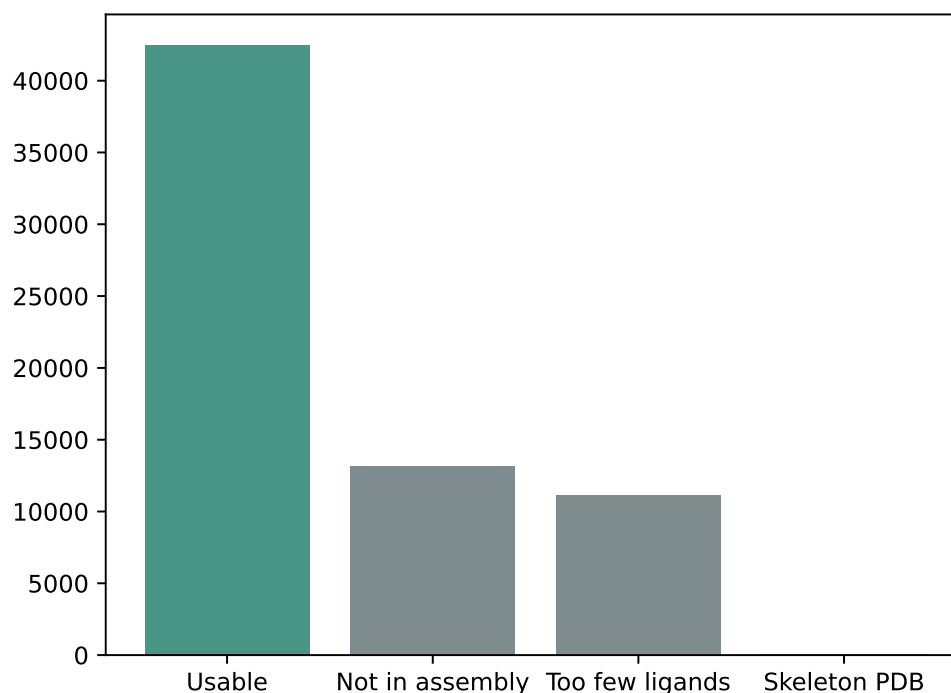


FIGURE 4.12: Illustrating the relative frequency of omission reasons for zinc atoms that were ultimately not assigned to a binding site. There were 8 which were omitted because the PDB structure did not contain side chains — insignificant compared with the other two reasons.

### 4.3.3 Zinc Binding Sites with High Representation

The 42,487 metal atoms, for which binding site information is stored, belong to a total of 38,035 zinc binding sites in ZincBind. After clustering the proteins at 90% sequence identity and then clustering the binding sites as described above, there are 17,471 unique zinc binding sites in the database. The binding site with the most copies is from carbonic anhydrase (355 copies), a catalytic serum protein and, as already noted, the first known zinc binding protein. This is followed by JMJD2D (247 copies), a lysine-specific demethylase. 11,785 (67.5%) zinc binding sites are currently unique in that they have only one structure, and are therefore only represented once in ZincBind. This distribution is shown in Figure 4.13

For each cluster of equivalent zinc binding sites, the best-resolution structure is chosen to provide a single site which is flagged as the ‘representative’ for that cluster.

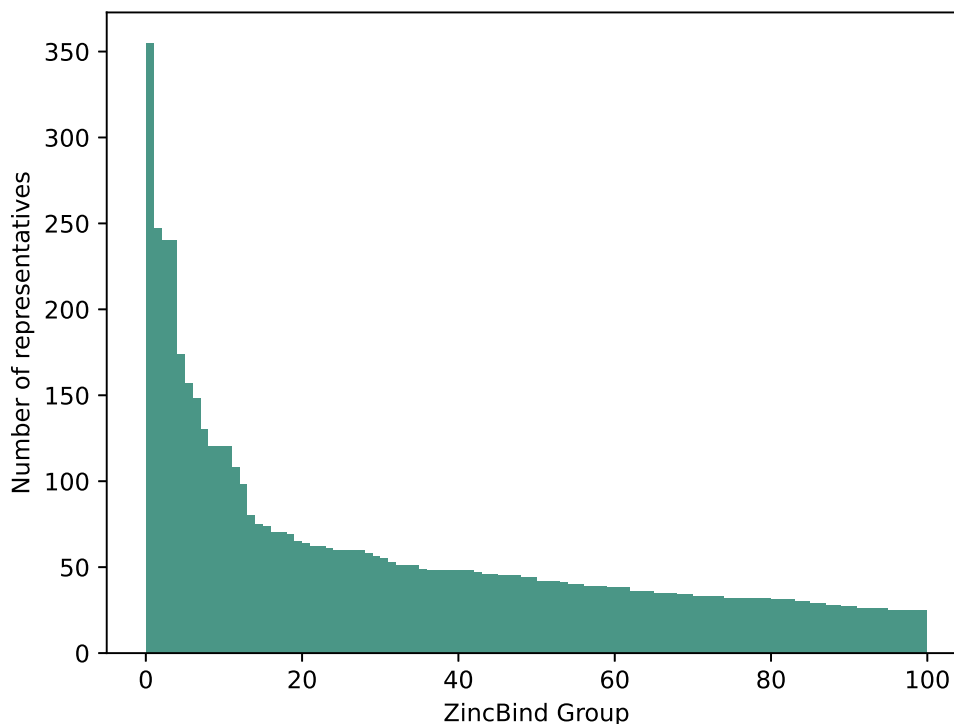


FIGURE 4.13: The distribution of binding sites among groups for the hundred most populated groups.

#### 4.3.4 Liganding Residues

The binding residues that dominate zinc binding sites are well established: cysteine, histidine, and the acidic residues aspartate and glutamate. This is supported by the data in ZincBind. There are a total of 161,324 liganding residues in ZincBind, of which those four ‘CHED’ residue types, together with water, comprise 93.0% of all zinc-liganding residues. Note, however, that this considers every binding site in the database. When non-redundant sites are studied (i.e. only one site per cluster thereby removing the bias towards more intensely studied proteins), these five comprise a slightly smaller percentage of zinc-liganding residues (92.0%). This suggests that the binding sites most frequently appearing in the PDB show slightly less variation than a more representative sampling.

The secondary residues, those which contact the primary liganding residues, are also stored in ZincBind and here there is more variation. The most common secondary residue is Glycine (9.1%), followed by leucine (7.4%) and alanine (7.2%).

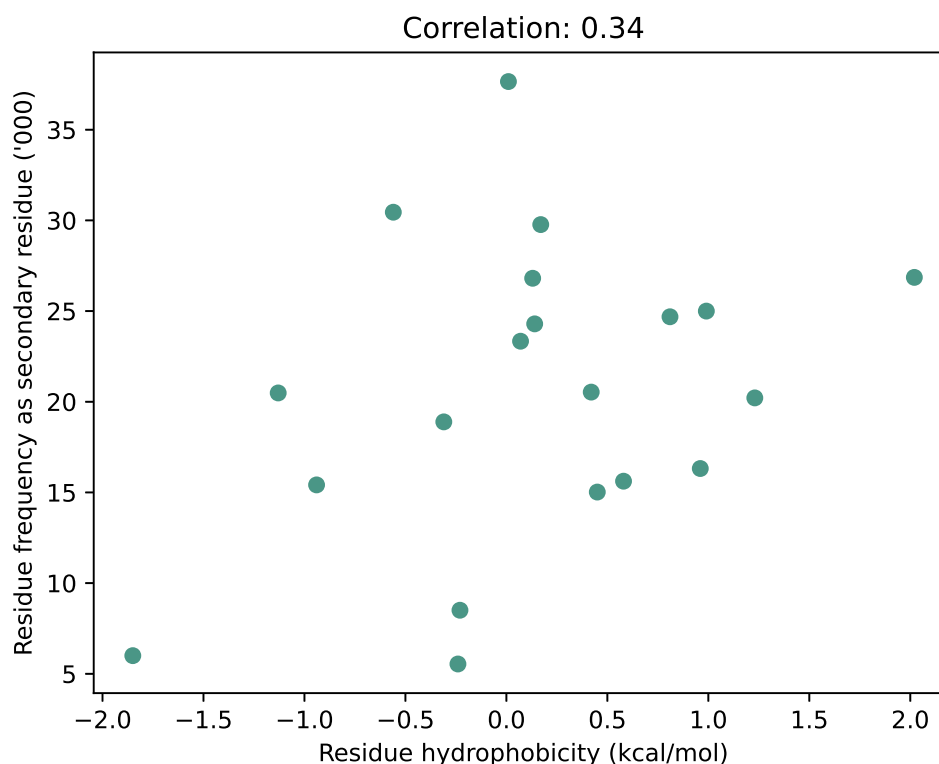


FIGURE 4.14: There is a weak correlation between residue hydrophobicity and their frequency as a secondary residue in zinc binding sites.

Broadly speaking, hydrophobic residues tend to be slightly more predominant than hydrophilic residues, which can be seen when the twenty amino acids' hydrophobicities are plotted against their frequency, which shows a correlation coefficient of 0.34 — a weak but significant link (see Figure 4.14).

If the primary binding residues' single letter codes are combined to give an overall 'signature' of the binding site, C4 (four cysteine residues) is the most prevalent, with 3,069 of the 17,471 unique binding sites having this arrangement of residues (17.6%) - followed closely by C3H1, with 2149 (12.3%). That the most common signature makes up just 17.6% of the total is indicative of the relatively high diversity in such signatures. Between them, the top ten residue signatures account for just 64.4% of the total.

Of the 24,200 redundant binding sites that contain just one zinc atom, and which come from structures with resolutions better than 3Å, the most common mode of coordination is via four liganding atoms: 15,077 examples (62.3%). 5-coordination



and 3-coordination have similar prevalences: 4504 (18.6%) and 1840 (7.6%) respectively. It is likely that 3-coordination is being over-represented here as, in practice, many sites which have three liganding atoms in the PDB structure are likely missing a fourth ligand in the form of water, meaning that tetrahedral geometry is probably even more widespread than suggested here.

This same subset of binding sites can be used to investigate liganding atom distances. Nitrogen and sulphur atoms both have characteristic distances with tight distributions:  $2.12 \pm 0.16\text{\AA}$  and  $2.34 \pm 0.12\text{\AA}$  respectively (error ranges are standard deviations). Oxygen however has a much wider distribution, as it can be provided by either of the carboxylate oxygens of the acidic side chains, or from water, with the angle of these two atoms varying continuously. Its average distance to zinc is  $2.29 \pm 0.22\text{\AA}$ . Overall the average metal-ligand distance is  $2.25 \pm 0.24\text{\AA}$  (see Figure 4.15)

### 4.3.5 Co-active Binding Sites

In some cases multiple metals act in concert to form a single functional unit. Such sites are generally referred to as co-active binding sites in the literature (or ‘co-catalytic’ where the site is known to have a catalytic function). Here, such sites are defined as those instances where a single residue is liganded to more than one metal, according to the criteria defined above. The metals are organised into a single site as described in the methods.

In ZincBind, 3976 from the total of 38,035 zinc binding sites (12.7%) contain multiple metals: 3550 contain two metals, 394 contain three, 22 contain four, two contain five, and eight zinc binding sites contain six metals (though two of these are from a synthetic construct). In some cases these are binding sites with multiple zinc atoms, but in others the zinc is acting in concert with other metals. Potassium is the most common, represented 319 times, followed by iron (188) and copper (155).

These multi-metal sites account for all of the non-zinc metals in the ZincBind database. While the term ‘co-catalytic’ is sometimes used interchangeably with ‘co-active’, only 83.6% are derived from an enzymatic protein (defined here as a

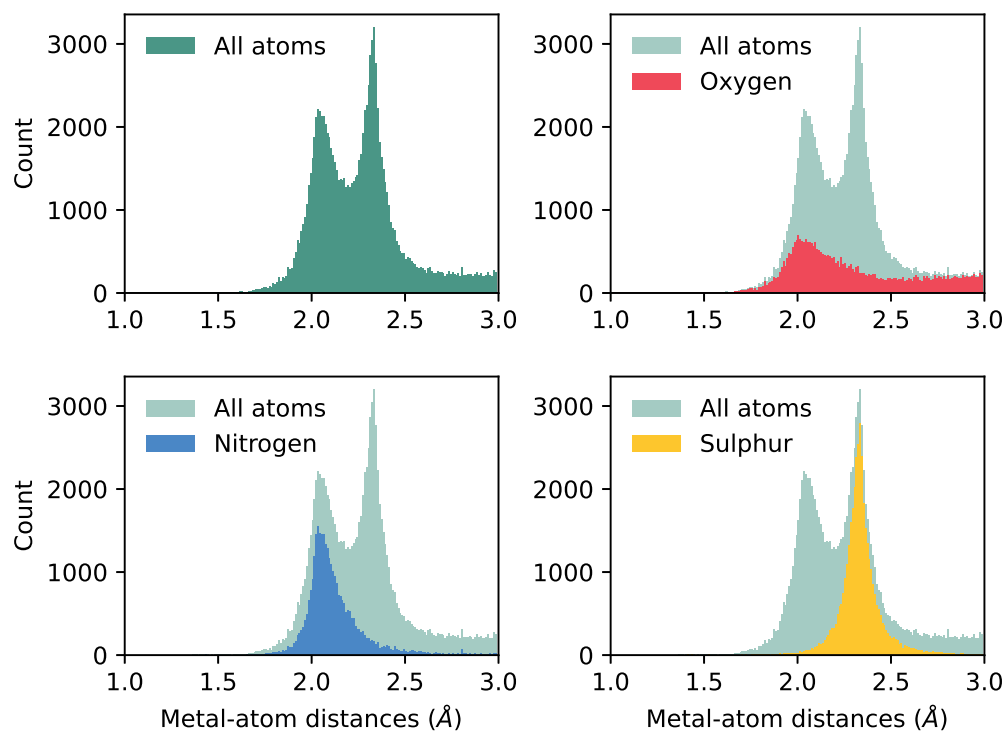


FIGURE 4.15: The distribution of liganding atom distances in all single-zinc sites, with a resolution better than  $3\text{\AA}$ . The overall distribution can be seen as bimodal, with the first peak being the preferred distance of nitrogen and oxygen, and the second peak being the preferred distance of sulphur, which has a greater Van der Waals radius. While nitrogen and sulphur both have tight distributions, oxygen does not, and has a much more prominent tail in its distribution.

protein name ending in *-ase*). However this compares with just 61.1% of all zinc-only binding sites that are enzymatic. A Fisher exact test showed the difference to be significant ( $p < 0.00001$ ).

## 4.4 Conclusion

ZincBind is a database of zinc binding sites — as such it is both a valuable dataset for this PhD, in the sense that it will serve as training data for predictive models of zinc binding (see Chapter 5), but it is also a useful resource in its own right. It is publicly accessible, with a modern GraphQL API and responsive website providing access to the data. The data itself is continuously and automatically updated, and

has characteristics broadly in line with previously observed properties of zinc and other metal binding sites in previous, smaller datasets.

The remainder of this thesis will focus on the dataset's role as a training set for predictive models.

# Chapter 5

## Predicting Zinc Binding Sites

*Note that some of the material in this chapter has been published in the journal Molecules under the title ‘Zincbindpredict—Prediction of Zinc Binding Sites in Proteins’ (2021) pp.966 [106].*

The aim of this part of the PhD is to create a system which conceptually is quite simple — it would take as its input a protein sequence or protein structure, and output a list of zero or more residue combinations which are predicted to form a zinc binding site, with an accompanying probability.

This system is a machine learning system; it uses binary classifiers trained on the large dataset of zinc binding sites described in Chapter 4 which ‘learn’ what zinc binding sites look like from that data.

This chapter will describe the approach taken to solving this problem, the models created, the architecture of the system from the user’s point of view, and how the models are accessed and used.

### 5.1 Approach

As explored in Chapter 2, there have been multiple studies in the past which have used machine learning (or simpler methods) to create zinc-binding predictors (or general-purpose metal-binding predictors). Many of the general principles of these

studies are adopted here, however there are two crucial methodological changes that have been made.

One recurring feature of previous work, particularly in the sequence-based predictive models, is the focus on zinc binding *residues* rather than zinc binding *sites*. In most cases, the entity examined by the predictive model is the individual residue, often with a surrounding linear sequence ‘window’ of residues. The model then assigns a probability as to whether that residue is a zinc binding residue. As outlined in Chapter 2, this approach has had a measure of success, but it is a somewhat artificial concept. There is, after all, no such thing as a zinc-binding residue in isolation. The individual residues of a high-affinity zinc binding site of the kind considered here are only zinc-binding when the other residues are present, and conversely many non-zinc-binding residues could bind zinc if other residues were present in the correct locations. It is particular *combinations* of residues, not individual residues, which are zinc binding — an important fact not usually considered in research of this kind. As this chapter will explain, models have been created here which inspect residue combinations, not single residues.

Another commonality is the treatment of zinc binding sites as a single category, and the presumption of properties that are common to them all regardless of the residues of which they are comprised. This may well be sufficient, particularly as there are essentially only four residues that make up the vast majority of zinc binding sites, but it is possible that properties used for prediction have much tighter distributions within particular sub-categories of zinc binding sites. As such, rather than creating a single model that predicts zinc binding in structures and another that predicts them in sequence, one model has been created per *family* of zinc binding site, of the kind described in Chapter 4<sup>1</sup>. That is, there is a model for predicting H3 binding sites (those made of three histidine residues) in structure, one for predicting H3 binding sites in sequence, one for predicting C2H2 binding sites in structure, one for predicting C2H2 sites in sequence, and so on. As well as the increased specificity this affords, another advantage of this approach is a purely practical one. All the possible H3 binding sites in a protein are found by taking all combinations of three histidine residues, and while the combinatorics of

---

<sup>1</sup>‘Family’ is used in the sense of the family of liganding residues and is not related to homologous families.

this can lead to large numbers of potential sites to check, it is vastly smaller than the combinations of *all* residues that would need to be checked if the model were supposed to look for generic zinc binding sites — an infeasible task for all but the smallest of proteins as the number of residue combinations quickly becomes impractically enormous.

The overall pipeline of the resulting system for predicting zinc binding then, is that when a protein structure or sequence is given, for each family for which there is a model, all residue combinations within that protein which match the family are identified (all unique combinations of three histidine residues for H3, for example) and passed to the relevant model, which assigns a probability that the residues comprise a zinc binding site, as well as making a binary yes-or-no prediction. This is repeated for every combination, and for every family, to build up a list of predicted binding sites.

## 5.2 Data Preparation

The first decision to make was which families to use. There are 701 families in total in ZincBind, ranging from C4 with its 3069 unique representatives, to obscure families like C1D1E1H2 (one cysteine, one glutamate, one aspartate and two histidine residues) which has only one representative. In fact there are 244 families with one representative, and many with only a small number of representatives. To train a model that can recognise some combination of residues as being a zinc binding site of some family rather than some random combination, the model needs to be trained on many examples of such sites, so there is clearly a minimum number of representatives in the database that a family must have for it to make sense to train a model for it. It is not immediately obvious what this minimum number should be. Initially it was decided to use the top ten families: C4, C3H1, C2H2, D1H2, E1H2, H3, C3, E1H1, C2H1 and D1H1. This was originally a somewhat arbitrary cutoff and the intention was to adjust it based on results, though as this chapter will show, ten families is probably the correct number in terms of resultant dataset size.

For a binary classifier of the kind being created here, the training set is a collection of positive samples (combinations of residues of that family which represent zinc binding sites) and negative samples (combinations of residues of that family which are not zinc binding sites). An assumption being made here is that if a combination of residues does not have a zinc atom bound to it in a structure, it is not a zinc binding site. This may not be true occasionally, as the structure simply may not have been crystallised in the presence of zinc. For this reason, some of the negative samples in the training sets may actually be positive. It is assumed however, that these cases are sufficiently infrequent as to have a negligible effect.

The residue combinations are represented in the training data as a vector of measurements, with the final measurement being the indicator of whether it is positive (1) or negative (0). Each row in the training data represents a residue combination, each column a feature. There are twenty training datasets altogether; for each of the ten families there is a training set for sequence data and a training set for structural data.

### 5.2.1 Sequence Training Data

To generate the sequence training datasets, for each family the ZincBind API was queried for all binding sites of that family, and those split over multiple chains were removed to leave single sequence binding sites — the multi-chain zinc binding sites do not have a single sequence containing all the necessary residues in the combination. The resulting sequences were turned into feature vectors which contained the number of residues between each pair of binding residues, the average hydrophobicity (Wimley and White's scale [107]) of residues either side of the binding residues, using windows of size 1, 3 and 7, and the average number of charged residues either side of the binding residues, using the same window sizes. These features are summarised in Table 5.2. This created a dataset of positive samples.

For the negative samples, for each family a sequence was chosen at random from the set of all unique sequences in UniProtKB and a combination of residues within that sequence matching the family but not a known binding site, was selected — this

was done repeatedly until a list of negative samples was built up equal in size to the positive dataset. The exclusion of sequences known to bind zinc, even if they have many residues combinations within them matching the family which do not bind zinc, ensured that no positive samples tainted the negative dataset (unidentified zinc binding sites notwithstanding). The two datasets were combined into a single dataset for each family. The resultant dataset is summarised in Table 5.1.

## 5.2.2 Structure Training Data

To generate the structure training datasets, for each zinc-binding family, all relevant zinc binding sites belonging to a PDB structure with resolution better than 2 Å were downloaded. As many of the measurements made are geometric, precise atom distances are important, and structures with too poor a resolution would create unreliable data for this purpose. For each PDB entry, the structure was downloaded and parsed using the Python library *atomium* (see Chapter 4), assembled into the correct biological assembly, and then each binding site was turned into a feature vector using the following measurements: mean inter  $C\alpha$  distance of the liganding residues, standard deviation of the  $C\alpha$  distances, minimum  $C\alpha$  distance, maximum  $C\alpha$  distance — and the corresponding measurements for the  $C\beta$  atoms, for a total of 8 geometric features. The distances used are all the pairwise combinations of the atoms involved, so H3 sites will have three inter  $C\alpha$  distances, C4 sites will have six, and so on. The final feature is the ‘hydrophobicity contrast function’, calculated at the centre of the  $C\beta$  atoms with a radius of 7 Å. This algorithm is a measure of how much outer atoms in a sphere are more hydrophobic than inner atoms, with higher values previously shown to be associated with centres of metal binding [10, 11], as described in Chapter 1.

As with the sequence data, this was repeated on combinations of residues with no known zinc binding ability until there was an equal number of negative samples. For these random combinations, PDB structures with no zinc atoms were selected at random (random sampling with replacement), and any random combination of residues from within that structure which matched the binding site family in question was selected — after applying a distance cutoff heuristic of 30 Ångströms between the alpha carbons to ensure that the distances between residues in the



TABLE 5.1: The sizes of the twenty training set sizes used to train the models. In each case the size is less than the theoretical maximum of double the total number of sites per family, because the sequence sites have those with multiple chains filtered out, and the structure sites have those from poor resolution structures filtered out — in the latter case this effect is considerable in reducing the total from its theoretical maximum.

Family	Structure Dataset Size	Sequence Dataset Size
C4	2825	15332
C3H1	3232	9158
H3	3078	4524
E1H2	1287	2574
C2H2	702	3715
D1H2	982	2406
C3	407	2591
C2H1	506	1926
D1H1	522	804
E1H1	416	812

negative dataset are of the same order of magnitude as those in the positive dataset. This step is necessary to ensure that the model did not simply learn that residues far apart spatially are not zinc binding and residues close together are zinc binding (as the majority of the negative dataset would be filled with very distant residues), but rather forces it to learn to identify when the residues are spatially close but not zinc binding. The resultant dataset is summarised in Table 5.1.

### 5.3 Model Training

While the quality and size of the training set is a large determinate of eventual model quality, the choice of machine learning algorithm is also crucial. Chapter 2 explored the history of metal binding site predictive models and the various algorithms used — as expanded upon there, there has been a shift towards the use of artificial neural networks over the past five years. The primary reason behind this shift has been the vast increase in the size of available datasets, which are generally required for something which uses as many layers of abstraction as neural networks. As the datasets here are necessarily quite small compared with

TABLE 5.2: Details of how features are calculated for residue combinations in structure and sequence models. Hydrophobicity of sequence residues is defined using Wimley and White’s scale [107], charge is the count of charged residues (aspartate, glutamate, arginine, histidine and lysine). Structural features were chosen based on previously identified markers of zinc binding in structures (see Chapter 1). Sequence features were chosen based on features known to be effective at predicting zinc binding in sequences (see Chapter 2).

Model type	Feature
<b>Sequence</b>	
	Inter-residue distance (one per gap)
	Average hydrophobicity around residues (window 1)
	Average hydrophobicity around residues (window 3)
	Average hydrophobicity around residues (window 5)
	Average number of charges around residues (window 1)
	Average number of charges around residues (window 3)
	Average number of charges around residues (window 5)
<b>Structure</b>	
	Mean Inter-C $\alpha$ distance
	Maximum Inter-C $\alpha$ distance
	Minimum Inter-C $\alpha$ distance
	Inter-C $\alpha$ distance standard deviation
	Mean Inter-C $\beta$ distance
	Maximum Inter-C $\beta$ distance
	Minimum Inter-C $\beta$ distance
	Inter-C $\beta$ distance standard deviation
	Hydrophobic contrast (radius 4 Å)

those datasets (a deliberate tradeoff made in the hopes of creating highly specific models), it was decided that such algorithms were not appropriate.

K-Nearest Neighbor, Support Vector Machines, and Random Forest algorithms were therefore considered. Originally, the intention was to train a model with each of these algorithms and have the final model be a consensus of these three whereby they vote on each incoming residue combination. However early testing showed that Random Forest so consistently and significantly outperformed both the other two models in isolation, and the resultant consensus model, that it was ultimately decided to solely use Random Forest for training.

The actual training of the models was done using the Python library scikit-learn [108]. For each training dataset, the data was randomly split into training and test

data using an 80:20 split — the former used to train, the latter used to evaluate. As explained in Chapter 2 this is vital to ensure the resultant model is not overfit.

The hyper-parameters for each model were selected separately using 5-fold cross validation of the training set. The hyper-parameters explored were the impurity measure (gini vs. entropy — the algorithm used to split individual trees at each node), the maximum depth that the component trees could have (4, 6, 8 or no maximum), the number of trees in the forest (10, 100 or 1000), and the means of determining the best number of features at each split (either the square root of the number of features, or the  $\log_2$  of the number of features). Once optimal hyper-parameters were identified (determined by which combination produced the best MCC score in the cross-validation), the models were trained with those hyper-parameters using the entire training dataset.

Once the model was trained on the ideal hyperparameters using all the training data, the model was evaluated on the test data, and the true positive, false positive, true negative and false negative counts were calculated. From these the recall, precision, F1 score, and Matthews' Correlation Coefficient were calculated. All of these were saved to the model by making them properties of the actual Python object, and then this Python object was serialised to disk using the Python 'joblib' library for later use.

## 5.4 Model Performance

For the structural models, the lowest MCC score was 0.88 (for the E1H1 model). This, and the D1H1 model (MCC=0.91), relies on the geometry between just two residues, which makes creating a distinct separation between the two classes somewhat more difficult — though their performance is still very close behind the three-residue and four-residue family models. The structure models had an average MCC of 0.960 (see Table 5.3).

The sequence models also had high scores, though were more variable. The four-residue sites in particular had highly conserved patterns of residue spacing and flanking hydrophobicity. The average MCC score for the sequence models was

TABLE 5.3: Results for structure models, sorted by Matthews Correlation Coefficient (MCC). The two-residue families' performance was lower than the others as there is essentially just the measurements between two centres to perform the classification, but still scored relatively highly. Four-residue sites in particular were found to have very predictable properties.

Family	Dataset Size	Recall	Precision	F1	MCC
C2H2	702	1.00	1.00	1.00	1.00
C4	2825	1.00	1.00	1.00	1.00
C3H1	3232	1.00	0.99	1.00	0.99
E1H2	1287	1.00	0.99	1.00	0.99
C2H1	506	1.00	0.98	0.99	0.98
H3	3078	1.00	0.98	0.99	0.98
D1H2	982	1.00	0.98	0.99	0.98
C3	407	1.00	0.98	0.99	0.98
D1H1	522	1.00	0.91	0.95	0.91
E1H1	416	0.93	0.95	0.94	0.88

0.87, with the lowest MCC being 0.61 for the E1H1 model and 0.74 for the D1H1 model — again the two two-residue models were some way behind the MCC of 0.84 for the C3 model (see Table 5.4).

The high performance of the models appears to justify the methodological changes made compared with previous metal binding classifiers. The performance scores here compare favourably with recent comparable predictive models based on structure and sequence covered in Chapter 2 — most notably the ‘SVM and Sample-weighted Probabilistic Neural Network’ (MCC= 0.80) [71], the ‘meta-zinc predictor’ (MCC= 0.79) [89] and ZincExplorer (MCC= 0.78) [88].

While the training is affected by dataset size, this does not appear to be a significant limiting factor for most of the models here. Figure 5.1 shows the model performance (as MCC) for the sequence and structure models. The performance of the sequence models falls off as the dataset falls below a threshold number of data points in the low thousands — as does the performance of the structural models. The lowest three performing structural models were also the lowest three in dataset size (C3, E1H1, D1H1), but two of these have only two residues so, as discussed above, the performance might not be expected to be very good.

TABLE 5.4: Results for sequence models, sorted by Matthews Correlation Coefficient (MCC).

Family	Dataset Size	Recall	Precision	F1	MCC
C4	15332	1.00	0.98	0.99	0.98
H3	4524	0.98	0.99	0.98	0.97
C2H2	3715	0.97	0.99	0.98	0.95
C3H1	9158	0.98	0.96	0.97	0.94
E1H2	2574	0.95	0.97	0.96	0.92
D1H2	2406	0.94	0.95	0.94	0.90
C2H1	1926	0.93	0.95	0.94	0.88
C3	2591	0.95	0.89	0.92	0.84
D1H1	804	0.80	0.93	0.86	0.74
E1H1	812	0.81	0.83	0.82	0.61

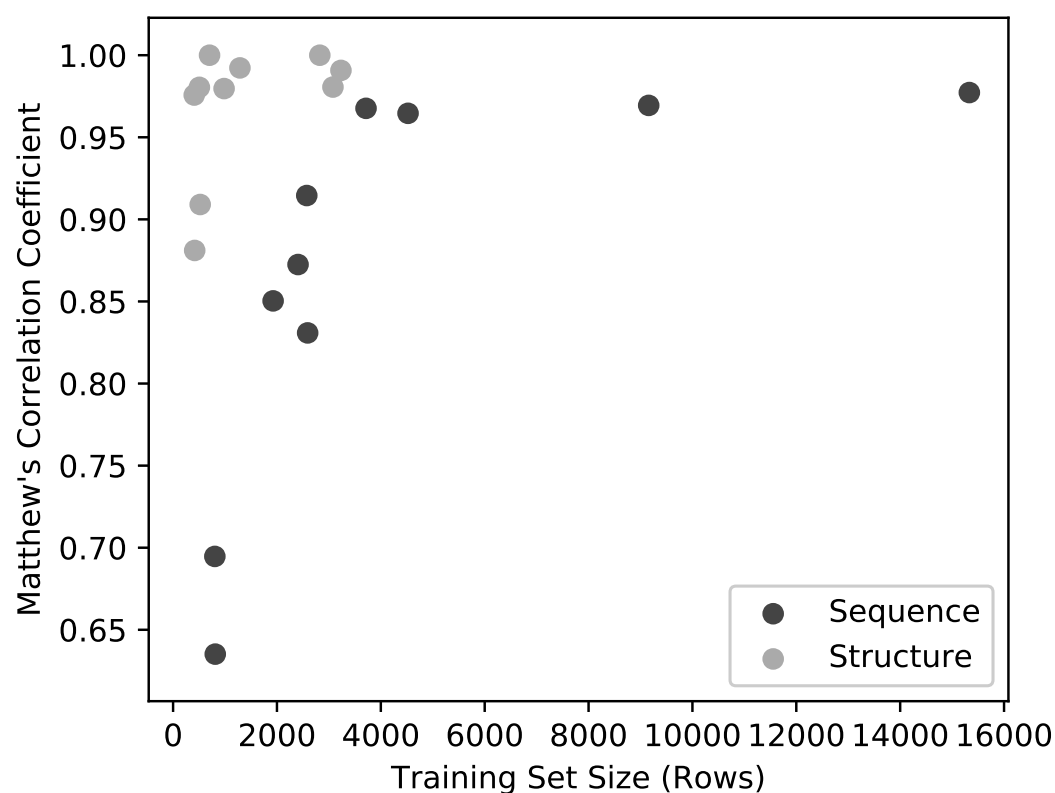


FIGURE 5.1: Model Performance (MCC) as a function of training set size. Below 4,000 rows, performance declines sharply, though above this threshold there ceases to be a strong correlation between the two.

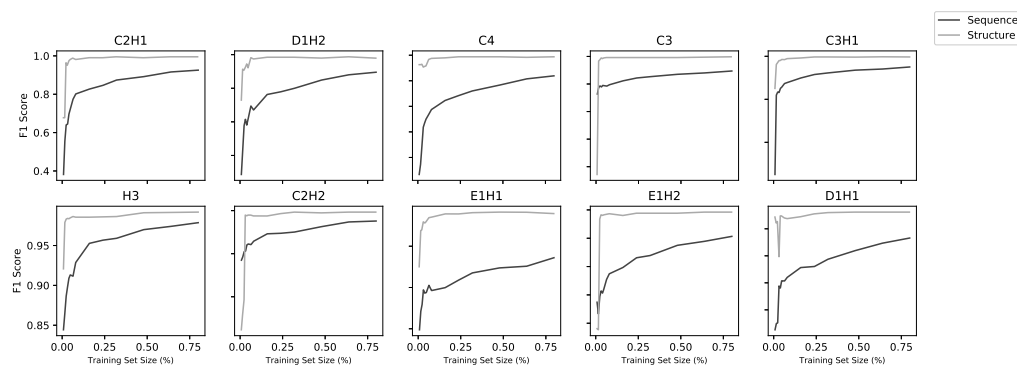


FIGURE 5.2: Learning curves for all 20 models. Each model was trained on increasing subsets of the overall training set using five-fold cross-validation. Sequence models improved with increasing dataset size, suggesting smaller dataset sizes would not be feasible, whereas structure models did not.

Learning curves (Figure 5.2) using fractions of the datasets show a correlation with dataset size for the sequence models, but above around 1000 sequences, the structure models do not improve with larger datasets.

The level of abstraction used to describe both sequences and structures made it unlikely that any homology between data in the training and testing sets would artificially improve the performance. The features are largely calculated from residues around the binding residues, rather than the sequence in which they occur. However given the presence of similar sequences in the dataset, it was thought prudent to confirm that these were not artificially increasing the models' performance.

To this end, the aim was to explore what would happen if only unique sequences were used, whereby the available sequences are clustered on some similarity threshold, and one representative from each used. Different sequence identity thresholds were used for clustering with CD-HIT and a training set was created from this new set of sequences for each threshold, and used to create a model. When clustering at 40% sequence identity (the lowest used), there was slightly lower performance, but clustering at this level did result in smaller datasets. As indicated previously, this is a major determinant of these sequence models' performance, so it was necessary to determine if this lowered performance was because the dataset size was smaller (which would mean the original models trained on all the data were not compromised) or if a model trained on unique sequences performed less well, which would call into question the high scores of the models as being due to such sequences.

TABLE 5.5: Model Performance on datasets clustered at different similarity thresholds.

Threshold	Family	Dataset Size	Recall	Precision	F1	MCC
<b>40%</b>						
	C2H1	286	0.793	0.821	0.807	0.621
	C2H2	394	0.952	0.952	0.952	0.898
	C3	368	0.947	0.9	0.923	0.839
	C3H1	870	0.976	0.901	0.937	0.877
	C4	1220	0.976	0.923	0.949	0.895
	D1H1	300	0.72	0.6	0.655	0.372
	D1H2	422	0.83	0.867	0.848	0.669
	E1H1	348	0.583	0.618	0.6	0.201
	E1H2	362	0.771	0.692	0.73	0.456
	H3	252	0.864	0.792	0.826	0.686
<b>50%</b>						
	C2H1	326	0.971	0.919	0.944	0.88
	C2H2	480	0.958	0.885	0.92	0.836
	C3	406	0.929	0.975	0.951	0.904
	C3H1	1050	0.957	0.925	0.941	0.865
	C4	1502	0.968	0.932	0.949	0.894
	D1H1	304	0.63	0.63	0.63	0.336
	D1H2	460	0.884	0.776	0.826	0.659
	E1H1	350	0.659	0.879	0.753	0.489
	E1H2	400	0.8	0.837	0.818	0.597
	H3	288	0.697	0.852	0.767	0.533
<b>60%</b>						
	C2H1	356	1.0	0.778	0.875	0.753
	C2H2	546	0.92	0.885	0.902	0.818
	C3	450	0.911	0.788	0.845	0.675
	C3H1	1154	0.981	0.898	0.938	0.882
	C4	1676	0.971	0.949	0.96	0.917
	D1H1	308	0.667	0.69	0.678	0.386
	D1H2	498	0.804	0.841	0.822	0.677
	E1H1	350	0.605	0.657	0.63	0.229
	E1H2	428	0.805	0.825	0.815	0.65
	H3	330	0.812	0.897	0.852	0.729
<b>70%</b>						
	C2H1	380	0.947	0.818	0.878	0.746
	C2H2	612	1.0	0.933	0.966	0.937
	C3	488	0.981	0.912	0.945	0.879
	C3H1	1236	0.976	0.931	0.953	0.904
	C4	1810	0.947	0.962	0.954	0.906
	D1H1	310	0.607	0.607	0.607	0.284
	D1H2	516	0.732	0.872	0.796	0.608
	E1H1	350	0.759	0.579	0.657	0.364
	E1H2	450	0.744	0.842	0.79	0.624
	H3	350	0.818	0.871	0.844	0.714
<b>80%</b>						
	C2H1	408	0.906	0.744	0.817	0.69
	C2H2	722	0.932	0.919	0.925	0.848
	C3	524	0.921	0.778	0.843	0.749
	C3H1	1372	0.992	0.921	0.955	0.915
	C4	1934	0.953	0.953	0.953	0.907
	D1H1	312	0.733	0.733	0.733	0.491
	D1H2	528	0.771	0.755	0.763	0.563
	E1H1	352	0.538	0.636	0.583	0.163
	E1H2	468	0.776	0.844	0.809	0.62
	H3	366	0.756	0.912	0.827	0.663
<b>90%</b>						
	C2H1	428	0.956	0.896	0.925	0.838
	C2H2	828	0.934	0.977	0.955	0.904
	C3	542	0.962	0.81	0.879	0.757
	C3H1	1500	0.987	0.949	0.968	0.934
	C4	2082	0.972	0.963	0.968	0.933
	D1H1	316	0.75	0.7	0.724	0.497
	D1H2	542	0.873	0.8	0.835	0.654
	E1H1	364	0.5	0.548	0.523	0.142
	E1H2	476	0.872	0.872	0.872	0.75
	H3	390	0.917	0.846	0.88	0.772
<b>100%</b>						
	C2H1	622	0.968	0.909	0.937	0.874
	C2H2	1188	0.952	0.945	0.949	0.89
	C3	700	0.941	0.81	0.871	0.739
	C3H1	2390	0.983	0.959	0.971	0.942
	C4	3224	0.985	0.955	0.97	0.938
	D1H1	382	0.707	0.806	0.753	0.513
	D1H2	778	0.9	0.9	0.9	0.819
	E1H1	424	0.543	0.758	0.633	0.346
	E1H2	800	0.845	0.909	0.876	0.785
	H3	1006	0.928	0.972	0.949	0.892

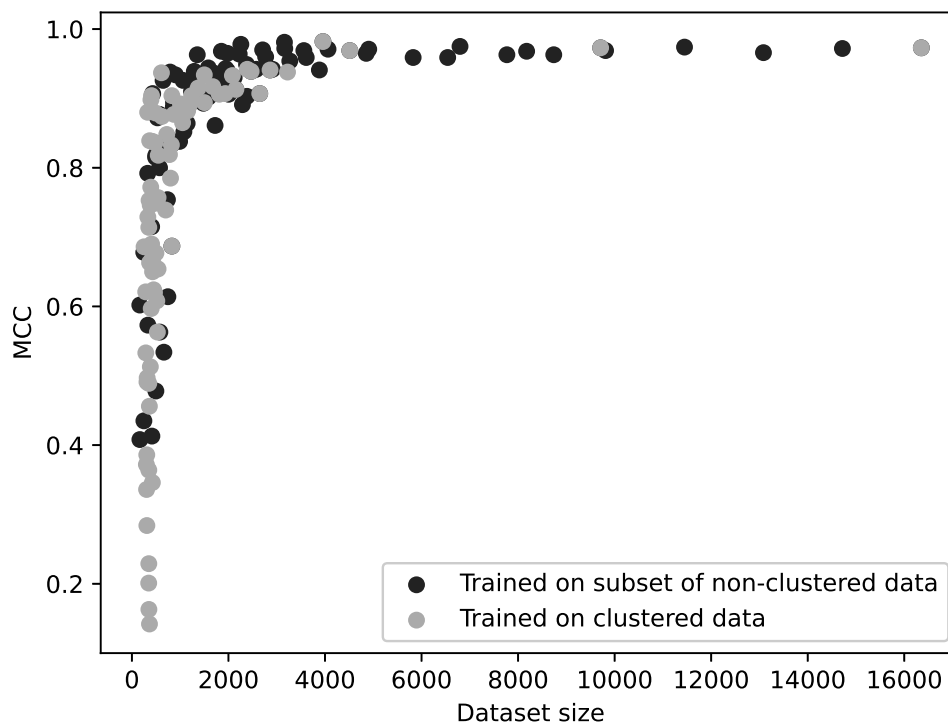


FIGURE 5.3: MCC as a function of dataset size for 160 different models. For each of the ten zinc-binding families, a classifier was trained on 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90% and 100% of the original, unclustered data, and also a classifier trained on data with sequences clustered by 40%, 50%, 60%, 70%, 80%, 90% and no clustering. Each of these models is shown here, with their performance (MCC) and size of the dataset used to train them. The two modes of dataset reduction are shown by different shades and it can be seen that the curves have an identical distribution. A model's performance is a function of its dataset size, regardless of whether any removal of similar sequences is performed.

In order to identify whether this lowered performance was because the models performed worse without the possibility of homologous sequences between the training and test sets, or whether it was a result of the smaller training set, for each zinc-binding family a classifier was trained on 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90% and 100% of the original, unclustered data, and also a classifier trained on data with sequences clustered by 40%, 50%, 60%, 70%, 80%, 90% and with no clustering (see Table 5.5). The performance of the models was then plotted against the resulting dataset sizes as shown in Figure 5.3. This demonstrates that it is dataset size that determines model performance, regardless of the similarity of the sequences in the training and testing datasets. For a given dataset, you



TABLE 5.6: Predictive ability of using BLAST alone to predict zinc binding in protein sequences using homology alone.

Family	Dataset Size	Recall	Precision	F1	MCC
C2H2	3960	0.99	0.95	0.97	0.94
C3H1	9710	0.29	0.87	0.44	0.33
C2H1	2154	0.24	0.88	0.37	0.3
D1H1	818	0.05	0.8	0.09	0.11
C3	2868	0.13	0.61	0.21	0.07
E1H1	828	0.06	0.62	0.11	0.06
D1H2	2470	0.03	0.53	0.06	0.01
H3	5058	0.01	0.19	0.02	-0.1
E1H2	2648	0.02	0.33	0.04	-0.06

could predict how well the model trained on it would perform based on dataset size alone — the similarity of the sequences within that dataset made no difference when dataset size was held constant.

As an additional means of showing how little effect sequence similarity has on ability to predict zinc binding, the sequence models were compared with using BLAST for predicting zinc-binding sites. For each zinc-binding family, a BLAST database was created using 80% of the available zinc-binding sequences, and BLAST’s ability to identify zinc binding sites from the remaining 20% was compared against an equivalently sized negative set. Results are shown in Table 5.6. With the exception of C2H2, using BLAST to find zinc binding based on homology performs much worse than the models presented here. Even in the case of C2H2, which seems to have much more similar sequences in its dataset, the machine learning model still narrowly outperforms BLAST.

However the models presented here are not intended to be general purpose zinc binding predictors that detect common properties of all zinc binding sites — they are family-specific predictors based on the principle that common, specific types of zinc binding site have more identifiable, consistent properties than do zinc binding sites in general. As a result, they will not readily detect binding sites of uncommon zinc-binding families. This abstract predictiveness has been deliberately discarded to create highly effective models for specific, common families of zinc binding sites. It is also noteworthy that the binding site itself is a useful unit of prediction using

TABLE 5.7: Percentage of genome predicted to be zinc binding by ZincBind-Predict for an assortment of bacterial genomes. Genomes were acquired from ensembl [109] in the form of translated polypeptide sequences, with a sequence labelled as zinc binding if any of the ten models finds at least one zinc binding site for that sequence/family combination.

Species	Percentage of Genome Predicted Zinc Binding
<i>Campylobacter jejuni</i>	6.4%
<i>Clostridioides difficile</i>	5.8%
<i>Enterococcus faecalis</i>	7.5%
<i>Listeria monocytogenes</i>	7.9%
<i>Mycobacterium tuberculosis</i>	11.3%
<i>Salmonella enterica</i>	11.1%
<i>Shigella flexneri</i>	10.1%
<i>Streptococcus pneumoniae</i>	7.6%

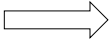
this methodology — even for sequences — rather than individual binding sites. The models are therefore identifying something biologically real (a zinc binding site) rather than something which does not actually exist in isolation (a single zinc binding residue), but which is a useful heuristic in some circumstances.

A demonstration of this can be seen by applying the sequence models to bacterial genomes to measure the proportion of typical genomes that the models predict to be zinc binding, as shown for a range of bacterial genomes in Table 5.7. For most genomes, fewer than 10% of proteins are flagged as zinc binding, with the average for the genomes examined being 8.46%. Given that the zinc-binding families for which predictors have been generated represent 67.0% of binding sites in ZincBindDB, this would imply a ‘true’ predicted proportion of 12.6% which is a little higher than the widely cited figure of 10%.

## 5.5 Access to Models

The models are available through a web app called ZincBindPredict. This is a GraphQL API, like the ZincBindDB database API, which allows users to submit jobs and get the results. A GraphQL request can be sent with either a protein sequence or protein structure, and a job ID will be returned (see Figure 5.4).

```
mutation {
  searchSequence(sequence: "INVGHSFHVNFEDNDNRSVL...") {
    jobId
  }
}
```



```
{
  "data": {
    "searchSequence": {
      "jobId": "1601810985030"
    }
  }
}
```

FIGURE 5.4: A mutation from the ZincBindPredict API. Mutations begin with the mutation identifier to indicate that the top level `Mutation` object is what the `searchSequence` mutation belongs to — queries can begin with `query` too but this is optional. Here the sequence is being given as an argument, and the ID of the job is returned.

This can then be polled for results as the protein or sequence is searched using each model in turn, with the identified binding sites returned as a list with the associated probability (see Figure 5.5). Internally, when a protein is submitted a ‘job’ folder is created, using the current UNIX time in milliseconds as the ID. This job ID is returned to the user, which they can use to query the status of the job, as a script which runs each of the model in turn runs as a background process and saves its results to the job’s folder on the server, for inspection by the API.

The ZincBind web interface that was explored in-depth in Chapter 4 also has a page that allows users to submit jobs using a more human-friendly interface, which itself consumes the ZincBindPredict API (see Figure 5.6). The user is prompted to provide the representation of a protein — either as a structure file, a sequence file, or a FASTA sequence pasted in. They also have the option of only searching for specific families. Positive results are listed on a results page once the job is complete, and the number of rejected residue combinations is listed.

## 5.6 Conclusion

The models created here are highly effective, albeit at a very particular task. They are not predictors of zinc binding in general, and they will not generally detect zinc binding sites of obscure families and residue combinations. They were never intended to. A deliberate trade-off between model effectiveness and model generalisability has been made in order to demonstrate the principal that properties of binding sites within families *are* more tightly distributed and hence create more

```

{
  sequenceJob(id: "1601810985030") {
    rejectedSites {
      residues probability family
    }
  }
}

```

→

```

{
  "data": {
    "sequenceJob": {
      "rejectedSites": [
        {
          "residues":
"maedldelldeveskfctpdllrrgmveqpkcgggthssdrnqakaketlrstetfkeddlslneile
epnlkdkpsklskssgntsvrasieglgkscspvylgssipcgigtiswradhlrciacdfllvvsydd
mdkscdylfrrnmpfhklkalkkkktrayaCqCswrtieevtdlqtdhqlrwcgkh",
          "probability": 0.945,
          "family": "C3"
        },
        {
          "residues":
"maedldelldeveskfctpdllrrgmveqpkcgggthssdrnqakaketlrstetfkeddlslneile
epnlkdkpsklskssgntsvrasieglgkscspvylgssipcgigtiswradhlrciacdfllvvsydd
mdkscdylfrrnmpfhklkalkkkktrayaCqCswrtieevtdlqtdhqlrwcgkh",
          "probability": 0.912,
          "family": "C3H1"
        },
        {
          "residues":
"maedldelldeveskfctpdllrrgmveqpkcgggthssdrnqakaketlrstetfkeddlslneile
epnlkdkpsklskssgntsvrasieglgkscspvylgssipcgigtiswradhlrciacdfllvvsydd
mdkscdylfrrnmpfhklkalkkkktrayaCqCswrtieevtdlqtdhqlrwcgkh",
          "probability": 0.9,
          "family": "C2H1"
        }
      ],
      ...
    }
  }
}

```

FIGURE 5.5: A query for the results of a ZincBindPredict job. The ID of the job is supplied, and this particular query requests the status of the job, as well the predicted sites. The rejected sites can also be requested, but since they are quite numerous, typically the flexibility of GraphQL in allowing you to choose to omit them is useful in conserving network resources.

The screenshot shows the ZincBind web interface. On the left, the 'Predict Zinc Binding' page has two sections: 'Sequence Prediction' with a text input for a sequence and a 'Predict' button, and 'Structure Prediction' with an 'Upload Structure File (.pdb, .cif or .mmtf)' button and a 'Predict' button. On the right, the 'Sequence Job' page shows a 'Status: complete' message, a 'Sequence' block with a long protein sequence, and 'Rejected sites: 2388'. Below that, it shows 'Predicted sites: 1' with a 'Display residue number' toggle. A note states: 'Note: no predictive model is perfect - predictions here are not guarantees of zinc binding. Some may be false positives, and some false negatives may be missing.' The protein sequence shown is: EINH2 2-1 ETUNGIDTDLTELAFENKWSQAZHKYNA YRKAASVIAKYPKIKSGAEAKKLPQVGTK IAEKIDFLATOKLRLEKTRDDTSSSIN FLTRVSGIOPSAARKFV DEGKTLKDLKANI EKLIKHWORISLKYVGFKEKRIIPRENLDM QDILVLEKVVQSEYIATVCGSFRGAESS GQDYLITHPSTFESTKQPKLLKQVYDQL QK4 HFETDLSKSGETKPHVQQLPSSKDEK EYFPHRIEIRLIPDQYVGVLYFTGSDIF NQMPRAALEKGFTEINEYTRPLVGTVAW EPLPVDSKQDFDYIQKVR EPKRSR

FIGURE 5.6: The prediction page on the ZincBind web interface, and an example of a predicted zinc binding site in a protein sequence.

accurate models. These models also detect something biologically real — full binding sites — rather than individual zinc binding residues which, as discussed, are an artificial concept.

However, this downside is not necessarily permanent. The models are limited to these ten models because these are the families for which there is sufficient (only just, in the case of some of the sequence models) data to train a classifier. However, the data in ZincBind is constantly growing thanks to the automated database update scripts in place. Over time, the eleventh-placed model will acquire enough data to warrant a model, and then the twelfth, and so on. The dataset-size threshold for training is not relative to the total size of the database, it is an

absolute amount, and so over time more and more families will be able to have models trained for them, and the models will become more comprehensive of zinc binding generally. The benefits of this trade-off are permanent — the deficiencies are temporary.

# Chapter 6

## Conclusion

This PhD has been an investigation into the properties of high affinity zinc binding sites in proteins, and the extent to which these properties can be used to create predictive models of zinc binding from protein structures or protein sequences.

Investigations of this kind into the nature of zinc binding are of importance at a basic research level, and in a direct medical sense. Understanding the geometric and biochemical properties of zinc binding sites offers insights into how they function — whether that function is catalytic or structure stabilising. Correspondingly, being able to determine whether a protein binds zinc can offer insights into what the function of that protein is, particularly if the location can be determined too. This is particularly effective in models which can predict from sequence, because entire genomes can be screened in minutes or hours to identify what zinc binding proteins might be present. Medically, the association of pathological zinc binding, particularly in the eye, is a demonstration of how vital a detailed understanding of the mechanisms of zinc binding is to understanding these diseases, and in how predicting the locations of binding sites can be used to aid pharmaceutical interventions.

This project, while an interesting intellectual exercise in and of itself, has been carried out with these crucial benefits to our understanding of the role of zinc in protein functions in health and disease at all times.

Understanding of the properties of zinc binding has been expanded by creating a more comprehensive database of all known zinc binding sites than previous research has created. The key properties and characteristics gleaned from Chapter 4 largely expanded in more detail on some properties that were already commented upon by previous works, albeit at a larger scale.

Probably a much more fundamental contribution to the future of this field, is the existence of ZincBindDB: not as a closed dataset that exists on a PhD Researcher's laptop and which can possibly, after an exchange of emails, be made available to other individuals — but rather a web accessible, continually updated resource that has both a web API and web interface. The vast majority of previous research of this kind has resulted in datasets of the former kind, which are not easily shared among researchers and which are, in any case, static once created. The data generated here is very easily accessed by any researcher in the world without the need for my involvement at all, and will continue to be updated with the very latest zinc binding structures indefinitely.

Similarly, those previous works which did result in a web interface have almost all ceased to exist. Even relatively recent papers from the mid-2010s contain URLs which, when accessed in 2021, point to nothing. Nobody can guarantee that they will maintain a service forever — particularly in the academic sciences where such services are funded by single standalone grants rather than revenue generated by those services. In this case however, all components of ZincBind are open source, cloneable, and reproducible. Even if neither I nor my lab continued to maintain ZincBind (it is currently very much the intention to do so), another researcher could recreate both the services and the data they contained very easily via the GitHub repositories and Docker images [110, 111], rather than having to recreate from scratch using the Methods section of a paper.

Over the long term, the particular insights into the properties of zinc binding that this PhD has uncovered will be expanded upon and may in some instances be superseded, but the long term presence and availability of this dataset via ZincBind may yet be the most important lasting contribution to the field of zinc binding.

The machine learning component of the PhD has, like the database creation component, built upon previous, similar studies. Here however the difference in methodology has been more stark. From the outset, I deliberately departed from two ubiquitous assumptions that these previous works made: that zinc binding was best detected at the individual residue level, and that the properties of zinc binding sites did not vary much between sub-categories of binding site. By making this tradeoff of only searching for a subset of zinc binding sites, you can achieve superior results in the prediction of zinc binding, and that this tradeoff will become less severe as time passes and the size of the underlying database grows.

Finally, this PhD has resulted in the creation of the Python library *atomium* for processing PDB structures. This library is already seeing use in the wider field and will continue to be updated with features.

## 6.1 Next Steps

It is worth examining, in closing, what the logical next steps would be were this project to continue. One very useful extension has already been remarked upon in chapter 5 and indeed will likely happen with very little input from the author or anyone else — the expansion of the predictive models beyond the existing ten. As chapter 5 explained in detail, there is currently enough data to justify these ten, but insufficient data for any more. As *ZincBindDB* is automatically updated every week from the Protein Data Bank, over time other families will cross the threshold of data size for these to have predictive models too, and the coverage of the models will gradually increase. This will simply require retraining the models, and adding families to the `.dat` file used by the build scripts. This will improve the usefulness of the tools over time.

Another possible expansion of the work done here that would have a rather high resultant benefit to effort required ratio, would be to expand beyond just zinc, to other metals. Because the primary objective of the original database creation was to identify properties of zinc binding sites, no previously identified properties were used in creating the database — the build scripts simply looks for PDB ligands with the name `ZN` and identifies binding sites for them based on general principles



of atomic interactions (sensible atom distance thresholds, minimum angles for atom clashing, etc.). Because nothing about this is specific to zinc, everything done here could just as easily be done for, as an example, copper, or iron. The only restriction used in looking for liganding atoms was the rejection of carbon as a possible liganding atom, but this will be true of all metals. Once this database of all transition metal binding sites were created, likewise creating predictive models for them would also require little to no changes to the already developed codebases. This was not done as part of this PhD to keep it tightly focused on the original research question and not allow for ‘mission creep’ — but it would greatly increase the usefulness of the already developed code with very little extra work.

An additional possible future avenue for research would be to try and create predictive models of a particular kind of zinc binding site - partial binding sites. These are those binding sites formed of multiple protein chains, in which often the zinc contributes to the oligomerisation of multiple subunits. Predicting these would be particularly useful in the aforementioned pathological aggregation of proteins in the presence of zinc, which is caused by cryptic half-binding sites in the proteins in question. Creating models that detect these is a much more difficult proposition, because it is very difficult to create a correctly labelled dataset. Positive samples are easily enough identified from ZincBindDB by filtering binding sites to include those from multiple chains — although they are less numerous than the single chain sites used here. Identifying negative samples is much more difficult though. The very nature of these sites means that unless a protein is crystallised in the presence of zinc and in the presence of a protein chain that could make up the other half of a binding site, it will go undetected. There is no easy way to reliably confirm that a pair of residues in a structure could not form a half binding site, simply because it does not in a particular structure, which means the methodology used here would not be suitable. That is not to say there is not path towards doing this, but unlike the previous two proposed future extensions, this could not be done by simply making superficial changes to existing code.

Other suggested features have been made by various interested parties over the course of the PhD — automatic classification of binding sites as structural or catalytic (sites are not labelled as such in PDB files, but they seem to have very

different properties so a dataset could be constructed with some effort), and prediction of binding site affinity (again this might be limited by the amount of available experimental data) to name two that would be of particular use.

Ultimately the best way to decide how future effort would best be spent is to be guided by the actual needs of people using the existing tools. All of the software created as part of this project is open source, and available on GitHub with its associated issue trackers and feature request managers. These offer a much more concrete guide to what would be most useful to the community, and is broadly the approach that will be taken in future. Indeed this decision to make every part of the project open to the community will hopefully be what makes ZincBind long-lasting in the way that its predecessors never were.

Embracing Open Science is the best way to sustain non-profit scientific software — it is this principle which has guided this PhD from the beginning, and will continue to guide ZincBind into the future.

# Appendix A

## Publications and Talks

### A.1 Published Papers

The following papers have been published from this PhD:

#### A.1.1 ZincBind—the database of zinc binding sites

*5 February 2019*

Sam M Ireland, Andrew C R Martin

GitHub: [github.com/samirelanduk/zincbinddb](https://github.com/samirelanduk/zincbinddb)

Docker: [hub.docker.com/r/samirelanduk/zincbinddb\\_django](https://hub.docker.com/r/samirelanduk/zincbinddb_django)

Abstract: Zinc is one of the most important biologically active metals. Ten per cent of the human genome is thought to encode a zinc binding protein and its uses encompass catalysis, structural stability, gene expression and immunity. At present, there is no specific resource devoted to identifying and presenting all currently known zinc binding sites. Here we present ZincBind, a database of zinc binding sites and its web front-end. Using the structural data in the Protein Data Bank, ZincBind identifies every instance of zinc binding to a protein, identifies its binding site and clusters sites based on 90% sequence identity. There are currently 24 992 binding sites, clustered into 7489 unique sites. The data are

available over the web where they can be browsed and downloaded, and via a REST API. ZincBind is regularly updated and will continue to be updated with new data and features.

### A.1.2 atomium—a Python structure parser

*11 February 2020*

Sam M Ireland, Andrew C R Martin

GitHub: [github.com/samirelanduk/atomium](https://github.com/samirelanduk/atomium)

Documentation: [atomium.bio](https://atomium.bio)

Abstract: Structural biology relies on specific file formats to convey information about macromolecular structures. Traditionally this has been the PDB format, but increasingly newer formats, such as PDBML, mmCIF and MMTF are being used. Here we present atomium, a modern, lightweight, Python library for parsing, manipulating and saving PDB, mmCIF and MMTF file formats. In addition, we provide a web service, pdb2json, which uses atomium to give a consistent JSON representation to the entire Protein Data Bank. atomium is implemented in Python and its performance is equivalent to the existing library BioPython. However, it has significant advantages in features and API design. atomium is available from [atomium.bioinf.org.uk](https://atomium.bioinf.org.uk) and `pdb2json` can be accessed at [pdb2json.bioinf.org.uk](https://pdb2json.bioinf.org.uk).

### A.1.3 Zincbindpredict—Prediction of Zinc Binding Sites in Proteins

*12 February 2021*

Sam M Ireland, Andrew C R Martin

GitHub: [github.com/samirelanduk/zincbindpredict](https://github.com/samirelanduk/zincbindpredict)

Docker: [hub.docker.com/r/samirelanduk/zincbindpredict\\_django](https://hub.docker.com/r/samirelanduk/zincbindpredict_django)

Abstract: Zinc binding proteins make up a significant proportion of the proteomes of most organisms and, within those proteins, zinc performs rôles in catalysis and structure stabilisation. Identifying the ability to bind zinc in a novel protein can offer insights into its functions and the mechanism by which it carries out those functions. Computational means of doing so are faster than spectroscopic means, allowing for searching at much greater speeds and scales, and thereby guiding complimentary experimental approaches. Typically, computational models of zinc binding predict zinc binding for individual residues rather than as a single binding site, and typically do not distinguish between different classes of binding site—missing crucial properties indicative of zinc binding. Previously, we created ZincBindDB, a continuously updated database of known zinc binding sites, categorised by family (the set of liganding residues). Here, we use this dataset to create ZincBindPredict, a set of machine learning methods to predict the most common zinc binding site families for both structure and sequence. The models all achieve an  $MCC \geq 0.88$ , recall  $\geq 0.93$  and precision  $\geq 0.91$  for the structural models (mean  $MCC = 0.97$ ), while the sequence models have  $MCC \geq 0.64$ , recall  $\geq 0.80$  and precision  $\geq 0.83$  (mean  $MCC = 0.87$ ), with the models for binding sites containing four liganding residues performing much better than this. The predictors outperform competing zinc binding site predictors and are available online via a web interface and a GraphQL API.

## A.2 Talks and Presentations

This PhD has been presented on the following occasions:

- 12 June 2018 — ISMB Graduate Symposium.
- 22 February 2019 — ISMB Friday Wrap.
- 24 January 2020 — GRC Metals in Biology Conference, Ventura, California.
- 6 March 2020 — ISMB Friday Wrap.
- 11 October 2020 — Exscientia (remote presentation).
- 15 October 2020 — Vernalis (remote presentation).

- 15 January 2021 — ISMB Friday Wrap (remote presentation).

# Appendix B

## atomium Documentation

atomium is a Python library for opening and saving .pdb, .cif and .mmtf files, and presenting and manipulating the information contained within. The rationale behind its development and details of its core concepts were outlined in Chapter 3 — this Appendix gives examples of its use in practice. The full documentation is at [atomium.bio](http://atomium.bio).

### B.1 Loading Data

While you can use atomium to create models from scratch to build an entirely *de novo* structure, in practice you would generally use it to load molecular data from an existing file...

```
>>> import atomium
>>> pdb1 = atomium.open('../1LOL.pdb')
>>> mmtf1 = atomium.open('/structures/glucose.mmtf')
>>> cif1 = atomium.open('/structures/1XDA.cif')
>>> pdb3 = atomium.open('./5CPA.pdb.gz')
>>> pdb2 = atomium.fetch('5XME.pdb')
>>> cif2 = atomium.fetch('5XME')
```

In that latter case, you don't need the file to be saved locally — it will just go and grab the PDB with that code from the RCSB.

atomium will use the file extension you provide to decide how to parse it. If there isn't one, or it doesn't recognise the extension, it will peek at the file contents and try and guess whether it should be interpreted as .pdb, .cif or .mmtf.

## B.2 Using Data

Once you've got your File object, what can you do with it?

### B.2.1 Annotation

There is meta information contained within the File object:

```
>>> pdb1.title
'CRYSTAL STRUCTURE OF OROTIDINE MONOPHOSPHATE DECARBOXYLASE COMPLEX WITH XMP'
>>> pdb1.deposition_date
datetime.date(2002, 5, 6)
>>> pdb1.keywords
['TIM BARREL', 'LYASE']
>>> pdb1.classification
'LYASE'
>>> pdb1.source_organism
'METHANOTHERMOBACTER THERMAUTOTROPHICUS STR. DELTA H'
>>> pdb1.resolution
1.9
>>> pdb1.rvalue
0.193
>>> pdb1.rfree
0.229
```

### B.2.2 Models and Assembly

All .pdb files contain one or more models — little universes containing a molecular scene.



```
>>> pdb1.model
<Model (2 chains, 4 ligands)>
>>> pdb1.models
(<Model (2 chains, 4 ligands)>,)

```

Most just contain one — it's generally those that come from NMR experiments which contain multiple models. You can easily iterate through these to get their individual metrics:

```
>>> for model in pdb2.models:
    print(model.center_of_mass)

```

This model contains the 'asymmetric unit' — this is one or more protein (usually) chains arranged in space, which may not be how the molecule arranges itself in real life. It might just be how they arranged themselves in the experiment. To create the 'real thing' from the asymmetric unit, you use *biological assemblies*.

Most .pdb files contain one or more biological assemblies — instructions for how to create a more realistic structure from the chains present, which in atomium are accessed using `File.assemblies`.

In practice, what you need to know is that you can create a new model (not the one already there containing the asymmetric unit) as follows...

```
>>> pdb3 = atomium.fetch('1XDA')
>>> pdb3.model
<Model (8 chains, 16 ligands)>
>>> pdb3.generate_assembly(1)
<Model (2 chains, 4 ligands)>
>>> pdb3.generate_assembly(10)
<Model (6 chains, 12 ligands)>
>>> [pdb.generate_assembly(n + 1) for n in range(len(pdb.assemblies))]
[<Model (2 chains, 4 ligands)>, <Model (2 chains, 4 ligands)>, <Model (2 cha
ins, 4 ligands)>, <Model (2 chains, 4 ligands)>, <Model (12 chains, 24 ligan
ds)>, <Model (12 chains, 24 ligands)>, <Model (6 chains, 12 ligands)>, <Mode
l (6 chains, 12 ligands)>, <Model (6 chains, 12 ligands)>, <Model (6 chains,
12 ligands)>, <Model (4 chains, 8 ligands)>, <Model (4 chains, 8 ligands)>]

```

Here you load a .pdb with multiple possible assemblies, have a quick look at the asymmetric unit with 1,842 atoms, and then generate first , and then all, of its possible biological assemblies by passing in their IDs.

### B.2.3 Model Contents

The basic structures within a model are chains, residues, ligands, and atoms.

```
>>> pdb1.model.chains()
{<Chain A (204 residues)>, <Chain B (214 residues)>}
>>> pdb1.model.chain('B')
<Chain B (214 residues)>
>>> pdb1.model.residues(name='TYR')
{<Residue TYR (A.37)>, <Residue TYR (B.1037)>, <Residue TYR (A.45)>, <Residue TYR (A.154)>, <Residue TYR (B.1206)>, <Residue TYR (B.1154)>, <Residue TYR (B.1045)>, <Residue TYR (A.206)>}
>>> pdb1.model.residues(name__regex='TYR|PRO')
{<Residue PRO (A.101)>, <Residue PRO (A.46)>, <Residue PRO (A.161)>, <Residue TYR (A.45)>, <Residue PRO (B.1046)>, <Residue TYR (A.154)>, <Residue TYR (B.1206)>, <Residue TYR (B.1045)>, <Residue PRO (B.1189)>, <Residue TYR (A.37)>, <Residue PRO (B.1129)>, <Residue PRO (B.1077)>, <Residue PRO (A.211)>, <Residue PRO (B.1180)>, <Residue PRO (B.1157)>, <Residue PRO (B.1211)>, <Residue PRO (B.1228)>, <Residue PRO (B.1101)>, <Residue TYR (B.1154)>, <Residue PRO (A.157)>, <Residue PRO (A.77)>, <Residue PRO (A.180)>, <Residue TYR (B.1037)>, <Residue PRO (A.129)>, <Residue PRO (B.1161)>, <Residue TYR (A.206)>}
>>> pdb1.model.chain('B').residue('B.1206')
<Residue TYR (B.1206)>
>>> pdb1.model.chain('B').residue('B.1206').helix
True
>>> pdb1.model.ligands()
{<Ligand BU2 (A.5001)>, <Ligand XMP (A.2001)>, <Ligand BU2 (B.5002)>, <Ligand XMP (B.2002)>}
>>> pdb1.model.ligand(name='BU2').atoms()
{<Atom 3196 (O3)>, <Atom 3192 (C1)>, <Atom 3193 (O1)>, <Atom 3197 (C4)>, <Atom 3194 (C2)>, <Atom 3195 (C3)>}
>>> pdb1.model.ligand(name='BU2').atoms(mass__gt=12)
{<Atom 3196 (O3)>, <Atom 3192 (C1)>, <Atom 3193 (O1)>, <Atom 3197 (C4)>, <Atom 3194 (C2)>, <Atom 3195 (C3)>}
>>> pdb1.model.ligand(name='BU2').atoms(mass__gt=14)
```

```
{<Atom 3196 (O3)>, <Atom 3193 (O1)>}
```

The examples above demonstrate atomium's selection language. In the case of the molecules — `Model`, `Chain`, `Residue` and `Ligand` — you can pass in an `id` or `name`, or search by regex pattern with `id__regex` or `name__regex`.

These structures have an even more powerful syntax too — you can pass in *any* property such as `charge=1`, any comparator of a property such as `mass__lt=100`, or any regex of a property such as `name__regex='[C]'`.

For pairwise comparisons, structures also have the `.AtomStructure.pairwise_atoms` generator which will yield all unique atom pairs in the structure. These can obviously get very big indeed — a 5000 atom PDB file would have about 12 million unique pairs.

Structures can be moved around and otherwise compared with each other...

```
pdb1.model.ligand(id='B:2002').mass
351.1022
>>> pdb1.model.ligand(id='B:2002').formula
Counter({'C': 10, 'O': 9, 'N': 4, 'P': 1})
>>> pdb1.model.ligand(id='B:2002').nearby_atoms(2.8)
{<Atom 3416 (O)>, <Atom 3375 (O)>, <Atom 1635 (OD1)>}
>>> pdb1.model.ligand(id='B:2002').nearby_atoms(2.8, name='OD1')
{<Atom 1635 (OD1)>}
>>> pdb1.model.ligand(id='B:2002').nearby_residues(2.8)
{<Residue ASP (B.1020)>}
>>> pdb1.model.ligand(id='B:2002').nearby_structures(2.8, waters=True)
{<Residue ASP (B.1020)>, <Water HOH (B.3155)>, <Water HOH (B.3059)>}
>>> import math
>>> pdb1.model.ligand(id='B:2002').rotate(math.pi / 2, 'x')
>>> pdb1.model.ligand(id='B:2002').translate(10, 10, 15)
>>> pdb1.model.ligand(id='B:2002').center_of_mass
(-9.886734282781484, -42.558415679537184, 77.33400578435568)
>>> pdb1.model.ligand(id='B:2002').radius_of_gyration
3.6633506511540825
>>> pdb1.model.ligand(id='B:2002').rmsd_with(pdb1.model.ligand(id='A.2001'))
0.133255572356
```

Here we look at one of the ligands, identify its mass and molecular formula, look at what atoms are within 2.8 Å of it, and what residues are within that same distance, rotate it and translate it through space, see where its new center of mass is, and then finally get its RMSD with the other similar ligand in the model.

Any operation which involves identifying nearby structures or atoms can be sped up — dramatically in the case of very large structures — by calling `.Model.optimise_distances` on the `Model` first. This prevents atomium from having to compare every atom with every other atom every time a proximity check is made.

The `Atom` objects themselves have their own useful properties.

```
>>> pdb1.model.atom(97)
<Atom 97 (CA)>
>>> pdb1.model.atom(97).mass
12.0107
>>> pdb1.model.atom(97).anisotropy
[0, 0, 0, 0, 0, 0]
>>> pdb1.model.atom(97).bvalue
24.87
>>> pdb1.model.atom(97).location
(-12.739, 31.201, 43.016)
>>> pdb1.model.atom(97).distance_to(pdb1.model.atom(1))
26.18289982030257
>>> pdb1.model.atom(97).nearby_atoms(2)
{<Atom 96 (N)>, <Atom 98 (C)>, <Atom 100 (CB)>}
>>> pdb1.model.atom(97).is_metal
False
>>> pdb1.model.atom(97).structure
<Residue ASN (A.23)>
>>> pdb1.model.atom(97).chain
<Chain A (204 residues)>
```

Chains are a little different from other structures in that they are iterable, indexable, and return their residues as a tuple, not a set...

```
>>> pdb1.model.atom(97).chain
```

```

<Chain A (204 residues)>
>>> pdb1.model.chain('A')
<Chain A (204 residues)>
>>> len(pdb1.model.chain('A'))
204
>>> pdb1.model.chain('A')[10]
<Residue LEU (A.21)>
>>> pdb1.model.chain('A').residues()[5]
(<Residue VAL (A.11)>, <Residue MET (A.12)>, <Residue ASN (A.13)>, <Residue
ARG (A.14)>, <Residue LEU (A.15)>)
>>> pdb1.model.chain('A').sequence
'LSRRVDVMDVMNRLILAMDLMNRDDALRVVTGEVREYIDTVKIGYPLVLSEGMDIIAEFRKRFGCRIIADFKVAD
IPETNEKICRATFKAGADAIIVHGFPGADSVRACLNVAEEMGREVFLITEMSHPGAEMFIQGADEIARMGVDLGV
KNYVGPSTRPERLSRLREIIGQDSFLISPGVGAQGGDPGETLRFADAIIVGRSIYLDNPAAAAAGIIESIKDLII'

```

The sequence is the ‘real’ sequence that exists in nature. Some of them will be missing from the model for practical reasons.

Residues can generate name information based on their three letter code, and are aware of their immediate neighbors.

```

>>> pdb1.model.residue('A.100')
<Residue PHE (A.100)>
>>> pdb1.model.residue('A.100').name
'PHE'
>>> pdb1.model.residue('A.100').code
'F'
>>> pdb1.model.residue('A.100').full_name
'phenylalanine'
>>> pdb1.model.residue('A.100').next
<Residue PRO (A.101)>
>>> pdb1.model.residue('A.100').previous
<Residue GLY (A.99)>

```

## B.3 Saving Data

A model can be saved to file using:

```
>>> model.save('new.cif')
>>> model.save('new.pdb')
```

Any structure can be saved in this way, so you can save chains or molecules to their own separate files if you so wish.

```
>>> model.chain('A').save('chainA.pdb')
>>> model.chain('B').save('chainB.cif')
>>> model.ligand(name='XMP').save('ligand.mmtf')
```

Note that if the model you are saving is one from a biological assembly, it will likely have many duplicated IDs, so saving to file may create unexpected results.

# Bibliography

- [1] C. Andreini, L. Banci, I. Bertini, and A. Rosato, “Counting the zinc-proteins encoded in the human genome,” *Journal of Proteome Research*, vol. 5, no. 1, pp. 196–201, 2006.
- [2] B. L. Vallee and D. S. Auld, “Zinc coordination, function, and structure of zinc enzymes and other proteins,” *Biochemistry*, vol. 29, no. 24, pp. 5647–5659, 1990.
- [3] M. J. Lachenmann, J. E. Ladbury, J. Dong, K. Huang, P. Carey, and M. A. Weiss, “Why zinc fingers prefer zinc: ligand-field symmetry and the hidden thermodynamics of metal ion selectivity,” *Biochemistry*, vol. 43, no. 44, pp. 13910–13925, 2004.
- [4] R. Williams, “The biochemistry of zinc,” *Polyhedron*, vol. 6, no. 1, pp. 61–69, 1987.
- [5] J. A. Tainer, V. A. Roberts, and E. D. Getzoff, “Metal-binding sites in proteins,” *Current Opinion in Biotechnology*, vol. 2, no. 4, pp. 582–591, 1991.
- [6] B. L. Vallee and D. S. Auld, “Functional zinc-binding motifs in enzymes and DNA-binding proteins,” *Faraday Discussions*, vol. 93, pp. 47–65, 1992.
- [7] J. E. Coleman, “Zinc proteins: enzymes, storage proteins, transcription factors, and replication proteins,” *Annual Review of Biochemistry*, vol. 61, no. 1, pp. 897–946, 1992.
- [8] P. Chakrabarti, “Geometry of interaction of metal ions with sulfur-containing ligands in protein structures,” *Biochemistry*, vol. 28, no. 14, pp. 6081–6085, 1989.

- [9] P. Chakrabarti, "Geometry of interaction of metal ions with histidine residues in protein structures," *Protein Engineering, Design and Selection*, vol. 4, no. 1, pp. 57–63, 1990.
- [10] M. M. Yamashita, L. Wesson, G. Eisenman, and D. Eisenberg, "Where metal ions bind in proteins.," *Proceedings of the National Academy of Sciences*, vol. 87, no. 15, pp. 5648–5652, 1990.
- [11] D. S. Gregory, A. C. Martin, J. C. Cheetham, and A. R. Rees, "The prediction and characterization of metal binding sites in proteins," *Protein Engineering, Design and Selection*, vol. 6, no. 1, pp. 29–35, 1993.
- [12] B. L. Vallee and D. S. Auld, "Short and long spacer sequences and other structural features of zinc binding sites in zinc enzymes," *FEBS Letters*, vol. 257, no. 1, pp. 138–140, 1989.
- [13] K. Patel, A. Kumar, and S. Durani, "Analysis of the structural consensus of the zinc coordination centers of metalloprotein structures," *Biochimica et Biophysica Acta (BBA)-Proteins and Proteomics*, vol. 1774, no. 10, pp. 1247–1253, 2007.
- [14] J. Miller, A. McLachlan, and A. Klug, "Repetitive zinc-binding domains in the protein transcription factor IIIA from *Xenopus oocytes*," *The EMBO journal*, vol. 4, no. 6, pp. 1609–1614, 1985.
- [15] F. Payre and A. Vincent, "Finger proteins and dna-specific recognition: Distinct patterns of conserved amino acids suggest different evolutionary modes," *FEBS Letters*, vol. 234, no. 2, pp. 245–250, 1988.
- [16] J. P. Mackay and M. Crossley, "Zinc fingers are sticking together," *Trends in Biochemical sciences*, vol. 23, no. 1, pp. 1–4, 1998.
- [17] H. M. Berman, "The protein data bank," *Nucleic Acids Research*, vol. 28, pp. 235–242, Jan. 2000.
- [18] I. L. Alberts, K. Nadassy, and S. J. Wodak, "Analysis of zinc binding sites in protein crystal structures," *Protein Science*, vol. 7, no. 8, pp. 1700–1716, 1998.



- [19] R. R. Roe and Y.-P. Pang, "Zinc's exclusive tetrahedral coordination governed by its electronic structure," *Molecular Modeling Annual*, vol. 5, no. 7-8, pp. 134–140, 1999.
- [20] J. H. Laity, B. M. Lee, and P. E. Wright, "Zinc finger proteins: new insights into structural and functional diversity," *Current Opinion in Structural Biology*, vol. 11, no. 1, pp. 39–46, 2001.
- [21] N. V. Grishin, "Treble clef finger-A functionally diverse zinc-binding structural motif," *Nucleic Acids Research*, vol. 29, no. 8, pp. 1703–1714, 2001.
- [22] M. M. Harding, "Geometry of metal–ligand interactions in proteins," *Acta Crystallographica Section D: Biological Crystallography*, vol. 57, no. 3, pp. 401–411, 2001.
- [23] S. S. Krishna, I. Majumdar, and N. V. Grishin, "Structural classification of zinc fingers: survey and summary," *Nucleic Acids Research*, vol. 31, no. 2, pp. 532–550, 2003.
- [24] D. S. Auld, "Zinc coordination sphere in biochemical zinc sites," in *Zinc Biochemistry, Physiology, and Homeostasis*, pp. 85–127, Springer, 2001.
- [25] Y.-m. Lee and C. Lim, "Physical basis of structural and catalytic zn-binding sites in proteins," *Journal of Molecular Biology*, vol. 379, no. 3, pp. 545–553, 2008.
- [26] B. Tamames, S. F. Sousa, J. Tamames, P. A. Fernandes, and M. J. Ramos, "Analysis of zinc-ligand bond lengths in metalloproteins: trends and patterns," *Proteins: Structure, Function, and Bioinformatics*, vol. 69, no. 3, pp. 466–475, 2007.
- [27] I. Dokmanić, M. Šikić, and S. Tomić, "Metals in proteins: correlation between the metal-ion type, coordination number and the amino-acid residues involved in the coordination," *Acta Crystallographica Section D: Biological Crystallography*, vol. 64, no. 3, pp. 257–263, 2008.
- [28] K. Goyal and S. C. Mande, "Exploiting 3D structural templates for detection of metal-binding sites in protein structures," *Proteins: Structure, Function, and Bioinformatics*, vol. 70, no. 4, pp. 1206–1218, 2008.

- [29] C. Andreini, I. Bertini, and G. Cavallaro, “Minimal functional sites allow a classification of zinc sites in proteins,” *PLoS One*, vol. 6, no. 10, p. e26325, 2011.
- [30] M. Sommerhalter, R. L. Lieberman, and A. C. Rosenzweig, “X-ray crystallography and biological metal centers: Is seeing believing?,” *Inorganic Chemistry*, vol. 44, no. 4, pp. 770–778, 2005.
- [31] M. Laitaoja, J. Valjakka, and J. Janis, “Zinc coordination spheres in protein structures,” *Inorganic Chemistry*, vol. 52, no. 19, pp. 10983–10991, 2013.
- [32] K. Hsin, Y. Sheng, M. Harding, P. Taylor, , and M. Walkinshaw, “Mespeus: a database of the geometry of metal sites in proteins,” *Journal of Applied Crystallography*, vol. 41, no. 5, pp. 963–968, 2008.
- [33] M. Jayakanthan, J. Muthukumaran, S. Chandrasekar, K. Chawla, A. Punetha, and D. Sundar, “Zifbase: a database of zinc finger proteins and associated resources,” *Bmc Genomics*, vol. 10, no. 1, p. 421, 2009.
- [34] C. Andreini, G. Cavallaro, S. Lorenzini, and A. Rosato, “Metalpdb: a database of metal sites in biological macromolecular structures,” *Nucleic Acids Research*, vol. 41, no. D1, pp. D312–D319, 2012.
- [35] V. Putignano, A. Rosato, L. Banci, and C. Andreini, “Metalpdb in 2018: a database of metal sites in biological macromolecular structures,” *Nucleic Acids Research*, vol. 46, no. D1, pp. D459–D464, 2017.
- [36] Y. Valasatava, A. Rosato, G. Cavallaro, and C. Andreini, “Metals 3, a database-mining tool for the identification of structurally similar metal sites,” *JBIC Journal of Biological Inorganic Chemistry*, vol. 19, no. 6, pp. 937–945, 2014.
- [37] V. Sobolev and M. Edelman, “Web tools for predicting metal binding sites in proteins,” *Israel Journal of Chemistry*, vol. 53, no. 3-4, pp. 166–172, 2013.
- [38] C. Andreini, G. Cavallaro, and S. Lorenzini, “Findgeo: a tool for determining metal coordination geometry,” *Bioinformatics*, vol. 28, no. 12, pp. 1658–1660, 2012.

- [39] H. Zheng, D. R. Cooper, P. J. Porebski, I. G. Shabalin, K. B. Handing, and W. Minor, “*CheckMyMetal*: a macromolecular metal-binding validation tool,” *Acta Crystallographica Section D*, vol. 73, pp. 223–233, Mar 2017.
- [40] B. L. Vallee and R. Williams, “Metalloenzymes: the entatic nature of their active sites,” *Proceedings of the National Academy of Sciences*, vol. 59, no. 2, pp. 498–505, 1968.
- [41] R. J. WILLIAMS, “The symbiosis of metal and protein functions,” *European Journal of Biochemistry*, vol. 150, no. 2, pp. 231–248, 1985.
- [42] B. L. Vallee and A. Galdes, “The metallobiochemistry of zinc enzymes,” *Adv Enzymol Relat Areas Mol Biol*, vol. 56, pp. 283–430, 1984.
- [43] H. Vahrenkamp, “Why does nature use zinc—a personal view,” *Dalton Transactions*, no. 42, pp. 4751–4759, 2007.
- [44] W. Maret and Y. Li, “Coordination dynamics of zinc in proteins,” *Chemical Reviews*, vol. 109, no. 10, pp. 4682–4707, 2009.
- [45] A. Krkezel and W. Maret, “The biological inorganic chemistry of zinc ions,” *Archives of Biochemistry and Biophysics*, vol. 611, pp. 3–19, 2016.
- [46] R. J. Williams, “Zinc: what is its role in biology?,” *Endeavour*, vol. 8, no. 2, pp. 65–70, 1984.
- [47] I. Bertini, C. Luchinat, and R. Monnanni, “Zinc enzymes,” *Journal of Chemical Education*, vol. 62, no. 11, p. 924, 1985.
- [48] D. W. Christianson and W. N. Lipscomb, “X-ray crystallographic investigation of substrate binding to carboxypeptidase A at subzero temperature,” *Proceedings of the National Academy of Sciences*, vol. 83, no. 20, pp. 7568–7572, 1986.
- [49] D. W. Christianson and C. A. Fierke, “Carbonic anhydrase: evolution of the zinc binding site by nature and by design,” *Accounts of Chemical Research*, vol. 29, no. 7, pp. 331–339, 1996.

- [50] S. F. Sousa, A. B. Lopes, P. A. Fernandes, and M. J. Ramos, “The zinc proteome: a tale of stability and functionality,” *Dalton Transactions*, no. 38, pp. 7946–7956, 2009.
- [51] U. Ryde, “Carboxylate binding modes in zinc proteins: a theoretical study,” *Biophysical Journal*, vol. 77, no. 5, pp. 2777–2787, 1999.
- [52] S. F. Sousa, P. A. Fernandes, and M. J. Ramos, “The carboxylate shift in zinc enzymes: A computational study,” *Journal of the American Chemical Society*, vol. 129, no. 5, pp. 1378–1385, 2007.
- [53] S. R. Udagedara, D. M. L. Porta, C. Spehar, G. Purohit, M. J. Hein, M. E. Fatmous, G. P. C. Garcia, K. Ganio, C. A. McDevitt, and M. J. Maher, “Structural and functional characterizations of the C-terminal domains of CzcD proteins,” *Journal of Inorganic Biochemistry*, vol. 208, p. 111087, July 2020.
- [54] M. Babor, H. M. Greenblatt, M. Edelman, and V. Sobolev, “Flexibility of metal binding sites in proteins on a database scale,” *Proteins: Structure, Function, and Bioinformatics*, vol. 59, no. 2, pp. 221–230, 2005.
- [55] W. Maret, “Metalloproteomics, metalloproteomes, and the annotation of metalloproteins,” *Metallomics*, vol. 2, no. 2, pp. 117–125, 2010.
- [56] T. Kochanczyk, A. Drozd, and A. Krkezel, “Relationship between the architecture of zinc coordination and zinc binding affinity in proteins—insights into zinc regulation,” *Metallomics*, vol. 7, no. 2, pp. 244–257, 2015.
- [57] S. K. Burley, C. Bhikadiya, C. Bi, S. Bittrich, L. Chen, G. V. Crichlow, C. H. Christie, K. Dalenberg, L. D. Costanzo, J. M. Duarte, S. Dutta, Z. Feng, S. Ganesan, D. S. Goodsell, S. Ghosh, R. K. Green, V. Guranović, D. Guzenko, B. P. Hudson, C. L. Lawson, Y. Liang, R. Lowe, H. Namkoong, E. Peisach, I. Persikova, C. Randle, A. Rose, Y. Rose, A. Sali, J. Segura, M. Sekharan, C. Shao, Y.-P. Tao, M. Voigt, J. D. Westbrook, J. Y. Young, C. Zardecki, and M. Zhuravleva, “RCSB protein data bank: powerful new tools for exploring 3D structures of biological macromolecules for basic and

- applied research and education in fundamental biology, biomedicine, biotechnology, bioengineering and energy sciences,” *Nucleic Acids Research*, vol. 49, pp. D437–D451, Nov. 2020.
- [58] D. Chicco and G. Jurman, “The advantages of the matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation,” *BMC Genomics*, vol. 21, Jan. 2020.
- [59] J. W. H. Schymkowitz, F. Rousseau, I. C. Martins, J. Ferkinghoff-Borg, F. Stricher, and L. Serrano, “Prediction of water and metal binding sites and their affinities by using the Fold-X force field,” *Proceedings of the National Academy of Sciences*, vol. 102, pp. 10147–10152, July 2005.
- [60] A. C. Wallace, N. Borkakoti, and J. M. Thornton, “Tess: a geometric hashing algorithm for deriving 3D coordinate templates for searching structural databases. application to enzyme active sites,” *Protein Science*, vol. 6, no. 11, pp. 2308–2323, 1997.
- [61] H. Deng, G. Chen, W. Yang, and J. J. Yang, “Predicting calcium-binding sites in proteins—a graph theory and geometry approach,” *Proteins: Structure, Function, and Bioinformatics*, vol. 64, no. 1, pp. 34–42, 2006.
- [62] W. Zhao, M. Xu, Z. Liang, B. Ding, L. Niu, H. Liu, and M. Teng, “Structure-based *de novo* prediction of zinc-binding sites in proteins of unknown function,” *Bioinformatics*, vol. 27, no. 9, pp. 1262–1268, 2011.
- [63] Z. Liu, Y. Wang, C. Zhou, Y. Xue, W. Zhao, and H. Liu, “Computationally characterizing and comprehensive analysis of zinc-binding sites in proteins,” *Biochimica et Biophysica Acta (BBA) — Proteins and Proteomics*, vol. 1844, no. 1, Part B, pp. 171 – 180, 2014. Computational Proteomics, Systems Biology and Clinical Implications.
- [64] W. He, Z. Liang, M. Teng, and L. Niu, “mFASD: a structure-based algorithm for discriminating different types of metal-binding sites,” *Bioinformatics*, vol. 31, pp. 1938–1944, 02 2015.

- [65] J. Joseph A. Cotruvo and J. Stubbe, “Metallation and mismetallation of iron and manganese proteins in vitro and in vivo: the class i ribonucleotide reductases as a case study,” *Metallomics*, vol. 4, no. 10, p. 1020, 2012.
- [66] G. Sciortino, E. Garribba, J. R.-G. Pedregal, and J.-D. Maréchal, “Simple coordination geometry descriptors allow to accurately predict metal-binding sites in proteins,” *ACS Omega*, vol. 4, pp. 3726–3731, Feb. 2019.
- [67] J. S. Sodhi, K. Bryson, L. J. McGuffin, J. J. Ward, L. Wernisch, and D. T. Jones, “Predicting metal-binding site residues in low-resolution structural models,” *Journal of Molecular Biology*, vol. 342, no. 1, pp. 307–320, 2004.
- [68] J. C. Ebert and R. B. Altman, “Robust recognition of zinc binding sites in proteins,” *Protein Science*, vol. 17, no. 1, pp. 54–65, 2008.
- [69] A. J. Bordner, “Predicting small ligand binding sites in proteins using backbone structure,” *Bioinformatics*, vol. 24, pp. 2865–2871, 10 2008.
- [70] C. Zheng, M. Wang, K. Takemoto, T. Akutsu, Z. Zhang, and J. Song, “An integrative computational framework based on a two-step random forest algorithm improves prediction of zinc-binding sites in proteins,” *PLOS ONE*, vol. 7, pp. 1–15, 11 2012.
- [71] “An improved prediction model for zinc-binding sites in proteins based on Bayesian method,” *Tehnicki Vjesnik - Technical Gazette*, vol. 26, Oct. 2019.
- [72] R. Nan, I. Farabella, F. F. Schumacher, A. Miller, J. Gor, A. C. Martin, D. T. Jones, I. Lengyel, and S. J. Perkins, “Zinc binding to the tyr402 and his402 allotypes of complement factor H: possible implications for age-related macular degeneration,” *Journal of Molecular Biology*, vol. 408, no. 4, pp. 714–735, 2011.
- [73] M. C. Mayer, D. Kaden, L. Schauenburg, M. A. Hancock, P. Voigt, D. Roeser, C. Barucker, M. E. Than, M. Schaefer, and G. Multhaup, “Novel zinc-binding site in the E2 domain regulates amyloid precursor-like protein 1 (APLP1) oligomerization,” *Journal of Biological Chemistry*, pp. jbc-M114, 2014.

- [74] M. S. Barbosa, D. R. Lowy, and J. T. Schiller, "Papillomavirus polypeptides E6 and E7 are zinc-binding proteins.," *Journal of Virology*, vol. 63, no. 3, pp. 1404–1407, 1989.
- [75] W. Bishop, P. Kirschmeier, S. George, S. Cramer, W. Hendrickson, *et al.*, "Identification and characterization of zinc binding sites in protein kinase C," *Science*, vol. 254, no. 5039, pp. 1776–1779, 1991.
- [76] T. E. Nash and M. R. Mowatt, "Variant-specific surface proteins of *Giardia lamblia* are zinc-binding proteins," *Proceedings of the National Academy of Sciences*, vol. 90, no. 12, pp. 5489–5493, 1993.
- [77] A. I. Bush, G. Multhaup, R. D. Moir, T. G. Williamson, D. H. Small, B. Rumble, P. Pollwein, K. Beyreuther, and C. Masters, "A novel zinc (ii) binding site modulates the function of the beta A4 amyloid protein precursor of Alzheimer's disease.," *Journal of Biological Chemistry*, vol. 268, no. 22, pp. 16109–16112, 1993.
- [78] Y. Furukawa, C. Lim, T. Tosha, K. Yoshida, T. Hagai, S. Akiyama, S. Watanabe, K. Nakagome, and Y. Shiro, "Identification of a novel zinc-binding protein, C1orf123, as an interactor with a heavy metal-associated domain," *PloS One*, vol. 13, no. 9, p. e0204355, 2018.
- [79] C. Andreini, I. Bertini, and A. Rosato, "A hint to search for metalloproteins in gene banks," *Bioinformatics*, vol. 20, no. 9, pp. 1373–1380, 2004.
- [80] A. Passerini, C. Andreni, S. Menchetti, A. Rosato, and P. Frasconi, "Predicting zinc binding at the proteome level," *BMC Bioinformatics*, vol. 8, no. 1, p. 29, 2007.
- [81] K. Nakata, "Prediction of zinc finger dna binding protein," *Bioinformatics*, vol. 11, no. 2, pp. 125–131, 1995.
- [82] C.-T. Lin, K.-L. Lin, C.-H. Yang, I.-F. Chung, C.-D. Huang, and Y.-S. Yang, "Protein metal binding residue prediction based on neural networks," *International journal of neural systems*, vol. 15, no. 01n02, pp. 71–84, 2005.
- [83] A. Passerini, M. Punta, A. Ceroni, B. Rost, and P. Frasconi, "Identifying cysteines and histidines in transition-metal-binding sites using support

- vector machines and neural networks,” *Proteins: Structure, Function, and Bioinformatics*, vol. 65, no. 2, pp. 305–316, 2006.
- [84] H. Lin, L. Han, H. Zhang, C. Zheng, B. Xie, Z. W. Cao, and Y. Z. Chen, “Prediction of the functional class of metal-binding proteins from sequence derived physicochemical properties by support vector machine approach,” in *BMC bioinformatics*, vol. 7, p. S13, BioMed Central, 2006.
- [85] N. Shu, T. Zhou, and S. Hovmöller, “Prediction of zinc-binding sites in proteins from sequence,” *Bioinformatics*, vol. 24, no. 6, pp. 775–782, 2008.
- [86] A. Srivastava and M. Kumar, “Prediction of zinc binding sites in proteins using sequence derived information,” *Journal of Biomolecular Structure and Dynamics*, pp. 1–11, 2018.
- [87] S. Kumar, “Prediction of metal ion binding sites in proteins from amino acid sequences by using simplified amino acid alphabets and random forest model,” *Genomics & Informatics*, vol. 15, pp. 162–169, Dec. 2017.
- [88] Z. Chen, Y. Wang, Y.-F. Zhai, J. Song, and Z. Zhang, “ZincExplorer: an accurate hybrid method to improve the prediction of zinc-binding sites from protein sequences,” *Molecular BioSystems*, vol. 9, no. 9, p. 2213, 2013.
- [89] H. Li, D. Pi, Y. Wu, and C. Chen, “Integrative method based on linear regression for the prediction of zinc-binding sites in proteins,” *IEEE Access*, vol. 5, pp. 14647–14656, 2017.
- [90] M. Karimi, D. Wu, Z. Wang, and Y. Shen, “DeepAffinity: interpretable deep learning of compound–protein affinity through unified recurrent and convolutional neural networks,” *Bioinformatics*, vol. 35, pp. 3329–3338, Feb. 2019.
- [91] I. Haberal and H. Ogul, “Prediction of protein metal binding sites using deep neural networks,” *Molecular Informatics*, vol. 38, no. 7, p. 1800169, 2019.
- [92] M. Heinzinger, A. Elnaggar, Y. Wang, C. Dallago, D. Nechaev, F. Matthes, and B. Rost, “Modeling the language of life – deep learning protein sequences,” Apr. 2019.



- [93] Zhao, Xu, and Zhao, “SXGBsite: Prediction of protein–ligand binding sites using sequence information and extreme gradient boosting,” *Genes*, vol. 10, p. 965, Nov. 2019.
- [94] X. Hu, Z. Feng, X. Zhang, L. Liu, and S. Wang, “The identification of metal ion ligand-binding residues by adding the reclassified relative solvent accessibility,” *Frontiers in Genetics*, vol. 11, Mar. 2020.
- [95] W. Li and A. Godzik, “Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences,” *Bioinformatics*, vol. 22, pp. 1658–1659, May 2006.
- [96] Z. Zhang, S. Schwartz, L. Wagner, and W. Miller, “A greedy algorithm for aligning DNA sequences,” *Journal of Computational Biology*, vol. 7, pp. 203–214, Feb. 2000.
- [97] D. S. Foundation, “Django (Version 2.2).” <https://djangoproject.com>, 2019.
- [98] F. Engineering, “GraphQL: A data query language.” <https://code.facebook.com/posts/1691455094417024>, 2015.
- [99] S. Akbary, “Graphene python.” <https://docs.graphene-python.org/>, 2020.
- [100] A. S. Rose and P. W. Hildebrand, “NGL viewer: a web application for molecular visualization,” *Nucleic Acids Research*, vol. 43, no. W1, pp. W576–W579, 2015.
- [101] S. M. Ireland and A. C. R. Martin, “atomium—a python structure parser,” *Bioinformatics*, vol. 36, pp. 2750–2754, Feb. 2020.
- [102] S. R. Hall, F. H. Allen, and I. D. Brown, “The crystallographic information file (CIF): a new standard archive file for crystallography,” *Acta Crystallographica Section A Foundations of Crystallography*, vol. 47, pp. 655–685, Nov. 1991.
- [103] K. Reitz, “Requests: HTTP for humans.” <https://requests.readthedocs.io/>, 2020.

- [104] S. M. Ireland and A. C. R. Martin, “ZincBind—the database of zinc binding sites,” *Database*, vol. 2019, Jan. 2019.
- [105] T. Smith and M. Waterman, “Identification of common molecular subsequences,” *Journal of Molecular Biology*, vol. 147, pp. 195–197, Mar. 1981.
- [106] S. M. Ireland and A. C. R. Martin, “Zincbindpredict—prediction of zinc binding sites in proteins,” *Molecules*, vol. 26, p. 966, Feb. 2021.
- [107] W. C. Wimley and S. H. White, “Experimentally determined hydrophobicity scale for proteins at membrane interfaces,” *Nature Structural Biology*, vol. 3, pp. 842–848, 1996.
- [108] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
- [109] A. D. Yates, P. Achuthan, W. Akanni, J. Allen, J. Allen, J. Alvarez-Jarreta, M. R. Amode, I. M. Armean, A. G. Azov, R. Bennett, J. Bhai, K. Billis, S. Boddu, J. C. Marugán, C. Cummins, C. Davidson, K. Dodiya, R. Fatima, A. Gall, C. G. Giron, L. Gil, T. Grego, L. Haggerty, E. Haskell, T. Hourlier, O. G. Izuogu, S. H. Janacek, T. Juettemann, M. Kay, I. Lavidas, T. Le, D. Lemos, J. G. Martinez, T. Maurel, M. McDowall, A. McMahon, S. Mohanan, B. Moore, M. Nuhn, D. N. Oheh, A. Parker, A. Parton, M. Patricio, M. P. Sakthivel, A. I. A. Salam, B. M. Schmitt, H. Schuilenburg, D. Sheppard, M. Sycheva, M. Szuba, K. Taylor, A. Thormann, G. Threadgold, A. Vullo, B. Walts, A. Winterbottom, A. Zadissa, M. Chakiachvili, B. Flint, A. Frankish, S. E. Hunt, G. Iisley, M. Kostadima, N. Langridge, J. E. Loveland, F. J. Martin, J. Morales, J. M. Mudge, M. Muffato, E. Perry, M. Ruffier, S. J. Trevanion, F. Cunningham, K. L. Howe, D. R. Zerbino, and P. Flicek, “Ensembl 2020,” *Nucleic Acids Research*, Nov. 2019.
- [110] S. M. Ireland, “ZincBindDB GitHub Repository.” <https://github.com/samirelanduk/zincbinddb>, 2018.

- 
- [111] S. M. Ireland, “ZincBindDB Docker Image.”  
[https://hub.docker.com/repository/docker/samirelanduk/zincbinddb\\_django/](https://hub.docker.com/repository/docker/samirelanduk/zincbinddb_django/),  
2021.