CHARACTERIZING STRUCTURE OF HIGH ENTROPY ALLOYS (HEAs) USING MACHINE LEARNING

CHRISTOFF REIMER

Thesis submitted to the University of Ottawa
in partial fulfillment of the requirements for the
degree of Master of Science, Physics

Department of Physics
Faculty of Science
University of Ottawa

Declaration

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

CHRISTOFF REIMER

December 8, 2023

Acknowledgments

This thesis would not have been possible without the support of my colleagues, friends, and family and their significant contributions to morale, understanding, and funding. It is time to give thanks to a few specific parties in turn.

To my supervisor, Dr. Isaac Tamblyn, I extend my gratitude for motivation and generosity. I would also like to thank my colleagues Dr. Peyman Saidi, Christopher Beeler, Dr. Corneel Casert, and Dr. Conrard Tetsassi for the optimism, patience, and conceptual/technical support which helped make this work possible. As well, my thanks to Yuxin Chang and Hitarth Choubisa for my initial education on graph and atom concepts in Python.

In addition, this work would not have been possible without the feedback provided by the various members of CLEAN, thank you all for lending your eyes and ears over the course of these projects.

Thanks to the National Research Council Canada (NRC), the University of Ottawa, NSERC, and Dr. Isaac Tamblyn for funding my studies.

The exceptional technical support and computing resources provided by the former Compute Canada (now Digital Alliance), as well as the resources of Google Colab, also helped this work to completion.

I also appreciate Saurabh Garg, who created this thesis template and made it publicly available online for students everywhere.

Lastly, I am grateful to my family for their relentless encouragement and understanding.

Abstract

The irradiation of crystalline materials in environments such as nuclear reactors leads to the accumulation of micro and nano-scale defects with a negative impact on material properties such as strength, corrosion resistance, and dimensional stability. Point defects in the crystal lattice, the vacancy and self-interstitial, form the basis of this damage and are capable of migrating through the lattice to become part of defect clusters and sinks, or to annihilate themselves. Recently, attention has been given to High Entropy Alloys (HEAs) for fusion and fission components, as some materials of this class have shown resilience to irradiation-induced damage. The ability to predict defect diffusion and accelerate simulations of defect behaviour in HEAs using Machine Learning (ML) techniques is consequently a subject that has gathered significant interest. The goal of this work was to produce an unsupervised neural network capable of learning the interatomic dynamics within a specific HEA system from Molecular Dynamics (MD) data in order to create a Kinetic Monte Carlo (KMC) type predictor of defect diffusion paths for common point defects in crystal systems such as the vacancy and self-interstitial. Self-interstitial defect states were identified and purified from MD datasets using graph-isomorphism, and a proof-of-concept model for the HEA environment was used with several interaction setups to demonstrate the feasibility of training a Graph Convolutional Network (GCN) to predict vacancy defect transition rates in the HEA crystalline environment.

Statement of Contributions

The majority of the work herein was done by me, Christoff Reimer, with other contributions detailed below.

The code base for this work was nearly all self-made using freely available Python packages, especially NetworkX (NX), Open Visualization Tool (OVITO), and Pytorch Geometric, with additional contributions as follows. Dr. Peyman Saidi contributed files used to set up MD simulations and initial knowledge of OVITO. Dr. Corneel Casert provided example code to replicate the custom log-likelihood loss function used in [1], which was modified for this work. Code used to box-cut dumbbell defects was adapted from package components supplied by Dr. Conrard Tetsassi. Adjustments needed to establish working code for multiprocessing or ramdisk use on Compute Canada (now known as Digital Research Alliance of Canada) were worked through with the help of their support personnel. Simple functions and python tricks (ex. the argmin function) were obtained from publicly-available sources and are individually attributed in the Python code comments where applicable. In addition, Christopher Beeler contributed to the base GCN class code via a publicly-available Pytorch tutorial.

All figures used were drawn or captured by me. Most were made in the vector graphics software Inkscape or taken as screenshots from visualizations of MD files using OVITO. Views of training results and plots were produced using the Python packages *matplotlib* and *seaborn* in Google Colab. The images in Figures 3.3 and 3.4 were originally made as part of a project that is currently in preparation for publication [2].

Adam Adaptive Moment Estimation

AMD Accelerated Molecular Dynamics

ANN Artificial Neural Network

BCC Body Centred Cubic

CDF Cumulative Distribution Function

CGCNN Crystal Graph Convolutional Neural Network

CGConv Crystal Graph Convolution

CNA Common Neighbour Analysis

CNN Convolutional Neural Network

DFT Density Functional Theory

dpa displacements-per-atom

EAM Embedded Atom Method

FCC Face Centred Cubic

FCNN Fully-Connected Neural Network

GCN Graph Convolutional Network

GNN Graph Neural Network

HCP Hexagonal Close Packed

HEA High Entropy Alloy

KMC Kinetic Monte Carlo

KNN K Nearest Neighbours

LAMMPS Large-scale Atomic/Molecular Massively Parallel Simulator

MAE Mean Absolute Error

MCS Maximum Common Subgraph

MD Molecular Dynamics

ML Machine Learning

MSE Mean Squared Error

NRC National Research Council Canada

NX NetworkX

OVITO Open Visualization Tool

PBC Periodic Boundary Condition

PDF Probability Density Function

PKA Primary Knock-on Atom

ReLU Rectified Linear Unit

RF Random Forest

PMF Probability Mass Function

RMSF Root Mean Square Fluctuation

RNN Recurrent Neural Network

ROI Region of Interest

SGD Stochastic Gradient Descent

SVM Support Vector Machine

TAD Temperature-Accelerated Dynamics

0-NNI first vacancy nearest neighbours without interaction

1-NNI first vacancy nearest neighbours with interaction

2-NNI first & second vacancy nearest neighbours with interaction

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the proliferation of newer and more powerful algorithms as well as larger datasets of physical-chemical properties, advances in understanding the composition and configuration dependent properties of engineered materials have been achieved with multiple Machine Learning (ML) models including ordinary least squares (OLS) [3], Artificial Neural Networks (ANNs) [4], Recurrent Neural Networks (RNNs) [4], and Crystal Graph Convolutional Neural Networks (CGCNNs) [5]. Of these, Graph Neural Networks (GNNs) have been of great interest in searching for catalyst/energy materials using local descriptors for material representations (such as atomic position, number, and bond length/distance), as both molecules and crystalline structures map cleanly into the flexible graph representation [6,7] and graphs can readily account for Periodic Boundary Conditions (PBCs) [5,8].

While much work has been conducted on determining the basic properties of such materials and screening them, some attention has also been directed to the use of ML models as proxies or hybrid models to reduce the time-expense of computer simulations and experiments in areas such as crystal plasticity [4], 2D materials growth [9], and defect migration, particularly Kinetic Monte Carlo (KMC) [10,11].

This work focuses on one category of emerging engineered materials, High Entropy Alloys (HEAs), a class containing materials with highly desirable properties. In order to better understand these materials, some background knowledge will be summarized in Chapters 2 and 3. Some background on the techniques used in this work is also contained in Chapters 4-6.

Much of the design and synthesis of HEAs to the present was done by a combination

of phase diagram calculations and trial-and-error [12, 13], however ML is becoming an increasingly used tool for the study and design of HEAs. Properties such as phase formation [12, 13], phase volume [12], yield strength [12] have all been used to train ML models such as K Nearest Neighbours (KNN), Support Vector Machines (SVMs), and Fully-Connected Neural Networks (FCNNs), using HEA features such as atomic concentration and entropy of mixing [13]. Many-atom, maximally entropic configurations for HEA solid solution structures have also been probed by simulations based on hybrid ANN-neural evolution networks [14].

In addition, there is no shortage of work using atomistic techniques such as Kinetic Monte Carlo (KMC), Molecular Dynamics (MD), and Density Functional Theory (DFT) to study HEAs, which tend to focus on the behaviour of lattice defects [10, 15–17, 17–31]. Hybrid workflows that combine ML with atomistic techniques have also seen use in materials science, for pure metals and for HEAs. One example is the use of ML models to learn DFT-comparable *interatomic potentials* [8, 32, 33], the rules for interactions between 2 or more atoms that govern MD simulations [34]. Further examples include predictions for the growth properties of thin films from KMC data [9], or of energy barriers for defect diffusion [35]. Other recent work has shown that certain types of convolutional ANNs, provided with sufficiently informative training data, can make predictions on complex properties that approach the accuracy of DFT results [8, 36]. The area of defect migration in HEAs has specifically seen attention very recently, with ANNs being used to investigate the behaviour of the vacancy defect in a hybrid ANN-KMC technique [37].

HEAs are difficult to study using popular techniques such as DFT because either the size of the simulation cell needs to be very large to capture a representative sample of HEA behaviour or many different samples need to be averaged [17]. This thereby incurs a large computational cost for HEA research. The MD technique is also hindered by the difficulty of producing an accurate MD interatomic potential force field for HEAs [16].

Precedents for graph-based representations of lattice systems [5, 38], hybrid atomistic-ML systems [8, 9, 32, 37], trained proxies for more computationally-intense methods [36], a method for approximating diffusion dynamics via unsupervised learning [1], and a

reasonable HEA interatomic potential to use [33], led to work that had, from the knowledge gleaned in the literature search, been under-explored. The use of GNNs using local HEA atomic configurations as feature inputs. This led to the development of several projects, the most significant can be briefly summarized as follows.

The use of subgraph isomorphism to classify and extract self-interstitial defects from MD trajectories for the purpose of providing training data for high throughput graph-based ML techniques. Described in Chapter 7.

The development of a graph-based ML proxy for MD calculations, trained using MD data and based on the principles of KMC, used to both predict and accelerate the diffusion dynamics present in an Face Centred Cubic (FCC) HEA environment made using a ML developed potential [33]. Described in Chapter 8.

Chapter 2

Crystals & Defects

2.1 Alloys & Crystalline Materials

An alloy is a mixture of a metal and at least one additional element, which may be a metal or a non-metal. Alloys are ubiquitous in the human world and have been for thousands of years, early examples include bronze (Sn and Cu) and steel (Fe and C) [39]. Alloys are useful because the properties of the base metal such as ductility, electrical/thermal conductivity, and hardness can be selectively altered by alloying.

Metals, as well as many solids, exist in repeating, efficient, and ordered configurations that minimize the total free energy of the system. These configurations are called *crystals* and such materials are called crystalline. An individual crystal can also be called a grain, particularly in metallurgy. Alloys often exhibit a multitude of individual grains that have formed together in the material in different orientations and compositions. The disordered regions where grains border each other are called *grain boundaries* (Figure 2.1) and are usually only a few atoms thick [40].

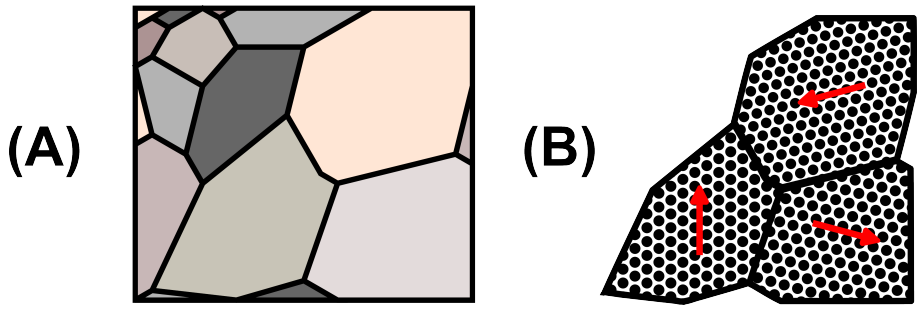


Figure 2.1: A 2D illustration of an arbitrary crystalline material. (A) Several arbitrarily sized and coloured grains. (B) A view of example atoms in three grains from the crystal in the prior image, showing different orientations present in each. Grain boundaries are represented by dark lines around the perimeter of each grain.

2.1.1 Crystals

Different crystal geometries are distinguished using their fundamental repeating substructure, called a *unit cell*. For example, the *simple cubic* unit cell defines a crystal lattice that can be abstracted as a series of cubes stacked along the principle 3D axes with atoms only at the corners of each cube (Figure 2.2) [41]. Note that although the atoms in a unit cell for a particular type of crystal are typically illustrated as whole atoms, they are counted as fractions of atoms for finding how many atoms are contained within a single unit cell of the crystal. How much of an atom is counted as being within a unit cell depends on the number of adjoining cells that an atom is a member of. In the simple cubic system, each of the 8 atoms illustrated is a member of 8 cells in total, and so the simple cubic unit cell encloses a total of 1 atom [41]. Another useful concept is *coordination number*, which is the number of equidistant atoms to each atom in the unit cell; this is equivalent to the number of *first nearest neighbours* to that atom. The size of the unit cell differs between crystals of different materials and is typically defined by *lattice parameters* a, b, and c which represent the lengths of the various cell dimensions. In cubic systems $a = b = c$ and so the lattice parameter is simply referred to as a. Lastly the *atomic packing factor* should be noted, a unitless measurement of the maximum packing efficiency of a given crystal type.

Most crystalline materials can be represented by one of a small list of known unit cells [42]. The type of unit cell a crystal has is of great importance to material properties such as ductility, strength, conductivity, as well as their macroscopic crystal appearance [40, 43]. The

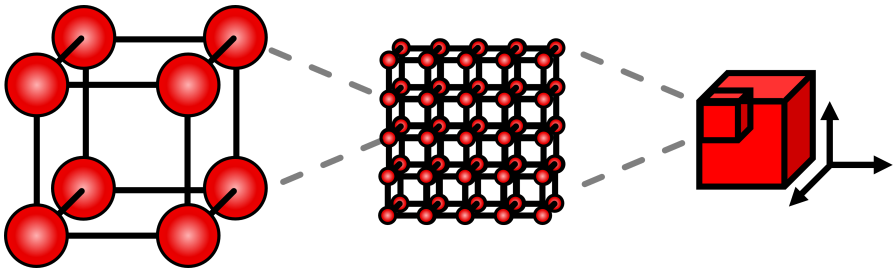


Figure 2.2: A representation of the simple cubic crystal unit cell, where an atom is found in each of the corners of a cubic lattice. Moving from left to right in the figure the unit cell is repeated, forming the basis of an arbitrary simple cubic crystalline material.

packing efficiency of the unit cell helps determine the stability of the crystal. For instance, the simple cubic structure used as an example in Figure 2.2 is too unstable to form under most conditions, and Polonium is the only element that naturally adopts this form [44]. The most common of these basic lattice structures are the Body Centred Cubic (BCC), Face Centred Cubic (FCC) (also called Cubic Close Packed [45]), and Hexagonal Close Packed (HCP) unit cells, shown in Figure 2.3 [40, 41]. Of these, the cubic forms are most relevant to this work and the FCC lattice in particular is extensively discussed in Section 2.1.1.

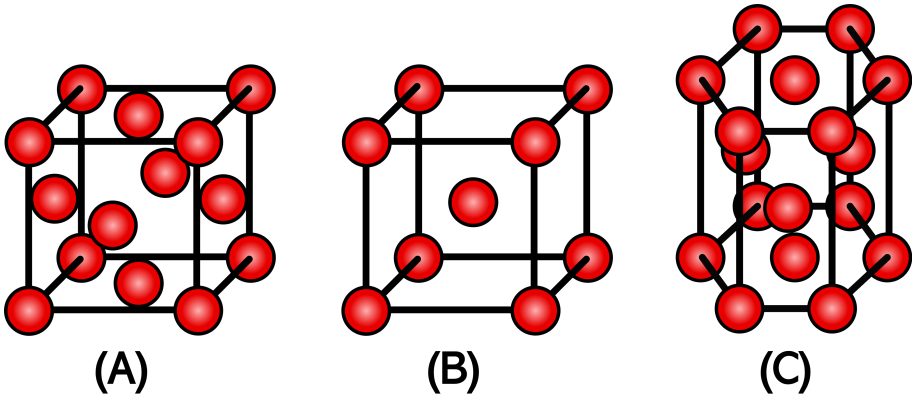


Figure 2.3: Representations of the most common crystal unit cells in nature and industry, (A) FCC, (B) BCC, (C) HCP [40]

The BCC unit cell is similar in appearance to the simple cubic cell, with the distinction that it has a single atom placed in the centre of the cell and at each corner [40]. This atom brings the total number of atoms in a BCC cell to 2 whole atoms [41]. The FCC cell is more visually complicated and is a simple cubic cell type modified by the addition of an atom placed in the middle of each of the 6 faces of the cube, resulting in 4 whole atoms per unit cell [41].

Face Centred Cubic (FCC) Crystals

As briefly described in Subsection 2.1.1, FCC crystals have a cubic unit cell with atoms positioned at each corner of a cube as well as in the centre of each of the 6 faces, with 4 whole atoms per FCC unit cell. The cell has the lattice parameter $a = 2\sqrt{2}R$ where R is the radius of the constituent particles, and each atom has a coordination number of 12 [46]. Elements with natural FCC structures on earth tend to be the more ductile metals such as gold, silver, nickel, and aluminum, among others [39, 40, 46]. The cell structure is very efficiently packed and stable, with an atomic packing factor of 74% [39, 46]. This is the greatest value that can be obtained without taking advantage of heterogeneous structures or interstitial sites [46].

A *primitive vector set* contains all the vectors necessary to completely describe the ideal lattice positions in a specific unit cell. The primitive vector set that describes the FCC unit cell as shown in Figure 2.3 (A) is noted in Equations 2.1, 2.2, and 2.3 [47], where a is the lattice parameter that defines a cube such that $a = b = c$.

$$a_1 = \frac{a}{2}\hat{y} + \frac{a}{2}\hat{z} \tag{2.1}$$

$$a_2 = \frac{a}{2}\hat{x} + \frac{a}{2}\hat{z} \tag{2.2}$$

$$a_3 = \frac{a}{2}\hat{x} + \frac{a}{2}\hat{y} \tag{2.3}$$

From the primitive vector set of the FCC cell it can be shown that any atom is equidistant from the coordinating atoms in its cell at a distance of $\frac{a}{\sqrt{2}}$. Let a single lattice site be considered at an arbitrary position. From this site there are 12 atoms remaining in the cell that can be grouped by geometry into three square planes in contact with 4 atoms each, centred on the chosen lattice site. The atoms in the XY plane can be reached by $\pm a_1 \mp a_2$, $\pm a_3$; the atoms in the YZ plane can be reached by $\pm a_1$, $\pm a_2 \mp a_3$; and the atoms in the XZ plane by $\pm a_2$, $\pm a_1 \mp a_3$. Each of the resulting vectors has the length $\frac{a}{\sqrt{2}}$.

2.1.2 Crystal Orientation

The orientation of a crystal is relevant for both mechanical and chemical properties of crystalline materials. The most obvious reason for this is that the surface density of the atoms in a crystal determines how many atoms are exposed at the interface of the crystal with surrounding media such as the atmosphere and chemical etchants [48]. Another way that orientation contributes to the properties of crystalline materials is that the relative positions of atoms in the unit cell define the directions of greatest and least energetic resistance for movement of those atoms during deformation [49].

The orientation distribution of grains in a crystalline alloy is not only relevant to chemical properties, but also affects the stress-strain behaviours of the material by rendering them anisotropic (orientation dependent) [50]. Grains can be found either in random or preferred orientations, which can be produced by mechanical treatments of the material [40].

Crystal orientations are represented as planes described by a set of three integer values called *Miller Indices*, written in the form *(hkl)*. These numbers are determined by taking the inverse of the intercept of a plane with each of the principal axes of the unit cell. For instance, a plane that intersects the x-axis of a cubic unit cell at 1 lattice parameter of distance from the origin (corner) and does the same for the y and z axes is said to have Miller Indices of $(hkl) = (1/1, 1/1, 1/1) = (111)$, illustrated in Figure 2.4 (A). In crystals with cubic unit cells, the vector *[hkl]* is normal to the crystal plane defined by *(hkl)* and moving along a vector of $a(hkl)$ ends in a point symmetrically identical to the starting position of the vector, assuming an infinite perfect crystal lattice. Planes and vectors that are identical by symmetry are grouped into families designated as *{hkl}* and <*hkl*> respectively.

One additional concept to note that is relevant to crystal planes is *slip*, a mass movement or glide of atoms in a crystal along a specific direction [40, 51]. The most common *slip directions* and *slip planes* in a crystal are grouped into symmetrically identical families called *slip systems* [51]. Slip is easier in systems that are cubic as well as those that are closely packed, hence the generally greater ductility of FCC metals compared to BCC metals [40, 51].

As the FCC cell is the cell focused upon in this body of work, the surface and properties of the FCC cell will now be described. The FCC cell, as outlined in Subsection 2.1.1 and

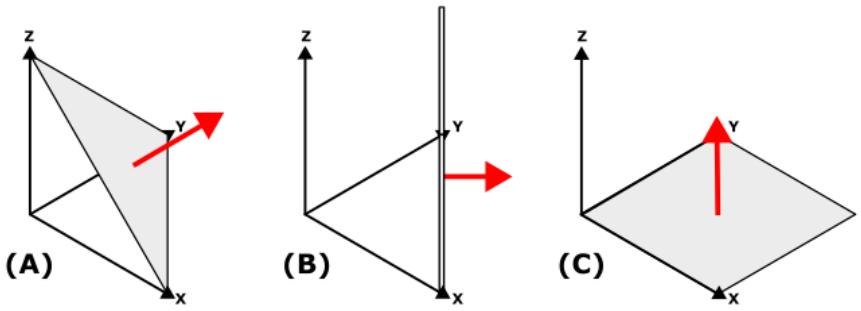


Figure 2.4: Examples of planes belonging to common families in cubic crystalline materials. (A) **{111}** (B) **{110}** (C) **{100}**. Due to their low Miller Index values, these are referred to as low index planes.

Figure 2.3 possesses parts of 12 participating atoms within a cubic volume of dimension *a*. By cutting an FCC unit cell along planes belonging to the common families (Figure 2.4), different crystal surfaces are formed (Figure 2.5 and Figure 2.6). Of these surfaces, the

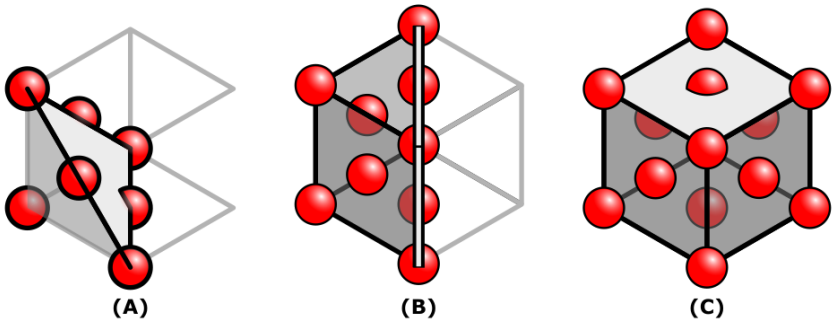


Figure 2.5: A diagram of the most common faces in the FCC crystal system. The plane surfaces are shown in light grey, with geometry on the back facing of the plane shaded and geometry on the front facing absent from the unit cell. (A) **(111)** (B) **(110)** (C) **(100)**

FCC {111} family of planes is the densest, with a 3-layer repeating symmetry instead of the 2-layer symmetry of the {110} and {100} planes. An example of the influence of plane facing on corrosion is slow etching of this plane in some circumstances due to the high density of bonds on that surface, and a higher density of pits when exposed to prolonged corrosion [48].

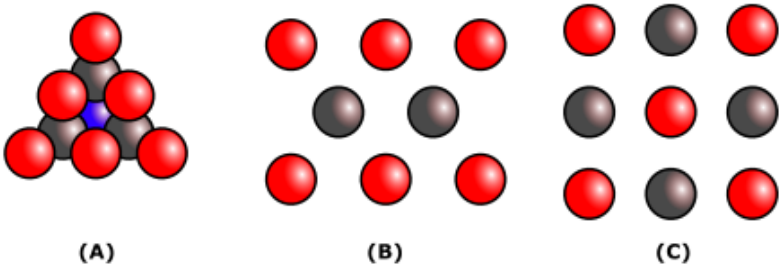


Figure 2.6: Surfaces within a single unit cell corresponding to the families of FCC planes presented in Figure 2.5. Atoms at the top layer are shaded red, atoms at the first sub-layer dark silver, atoms in the second sub-layer blue. (A) **{111}** (B) **{110}** (C) **{100}**

2.1.3 Solid Solutions

Chemically, a solution occurs when a solute material is dissolved in a solvent. At equilibrium the distribution of the solute should be uniform within the solvent, creating a homogeneous mixture. A *solid solution* is a material where both the solute and the solvent exist in the solid state and one material has generally been distributed randomly and uniformly across the volume of another [52].

The alloy brass, formed by mixing Zn and Cu together, is an example of a common solid solution, and Cu is often alloyed to mitigate its softness with elements such as Sn or Ni [52, 53]. This type of solid solution alloy is called a *substitutional* solid solution, since the solute atoms occupy lattice positions in the crystal structure of the alloy that would otherwise be occupied by the solvent atoms in the pure solvent structure (Figure 2.7). The distribution of the solute in a solid solution is not always truly random, and there may be short-ranged order, clusters within the solution (preferential clustering of certain elements) or precipitate phases with their own crystal structures (Figure 2.8) [52–56].

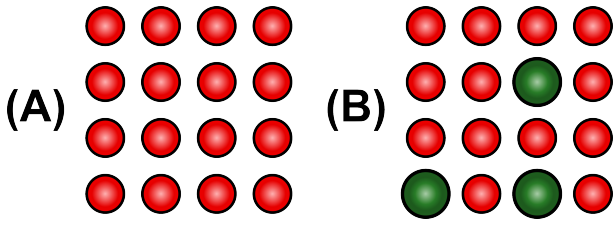


Figure 2.7: A concept of an arbitrary (A) pure metal and (B) binary (2 element) alloy with a substitutional type

In solid solutions, it is important that the solute element(s) are soluble in the solvent. An example of a good pairing is Cu and Zn, where Zn has a solubility greater than 35% [53]. However, even highly soluble pairings can form multiple phases as the solution becomes saturated, resulting in two phases with distinct differences in Cu:Zn proportions instead of a single solid solution phase [53]. For transition metals, properties such as atomic radius, crystal structure, and electronegativity play a role in determining solubility [57].

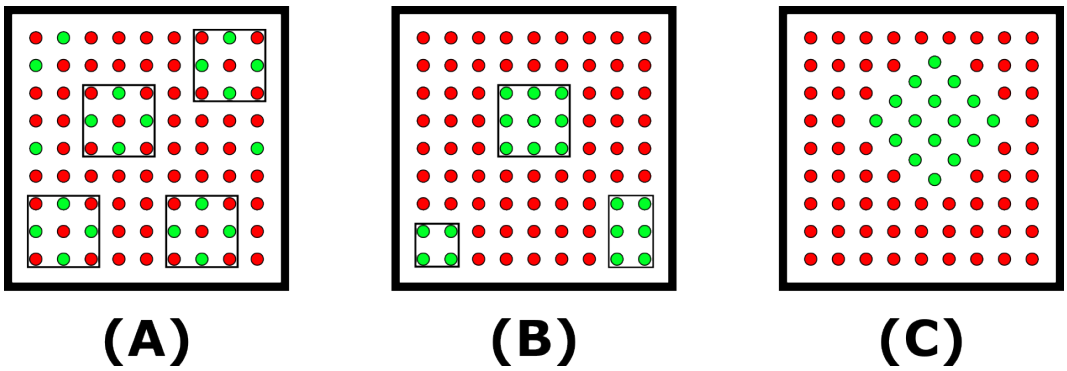


Figure 2.8: A concept of an arbitrary binary solid solution alloy exhibiting (A) Short-range order B) Short-range clustering (C) Precipitate phases

Solid solution alloys also benefit from a phenomenon called *solid solution hardening* in which the presence of varied atoms in the lattice leads to changes in the local elastic constants and stress fields around those atoms which hinder the movement of dislocations through their space [58]. A similar hardening effect occurs from the inclusion of larger particles in a lattice as well, in which case it is dubbed *precipitate* or *dispersion* hardening, depending on whether the particles are intrinsic or extrinsic to the lattice material [58]. The impairment of dislocation movement through a lattice by obstacles such as these is referred to as *pinning*.

2.1.4 Catalysis & Surface Sites

In Subsection 2.1.2, the utility of different surfaces being exposed to the extra-crystalline medium was briefly highlighted. To expand on the importance of this surface, consider *catalysts*. Catalysts are materials that are capable of enabling or accelerating the rate of chemical reactions without being consumed, by altering the energetic favourability of a reaction. Catalysts can be categorized as *homogeneous* or *heterogeneous* catalysts depending on if they occupy the same state as the reactants of the process they catalyze or not, respectively. Several valuable metals have intrinsic catalytic properties and are used as

heterogeneous catalysts, with platinum being the most well known. On the surfaces of these metallic heterogeneous catalysts, molecules preferentially adsorb to specific *surface sites* which facilitate reactions such as oxygen reduction/evolution (ORR/OER) or carbon dioxide/monoxide reduction (CO_2RR/CORR) [3, 59].

2.2 Crystal Defects

A *defect* is any deviation from the ideal structure of a crystal lattice. Defects of many different shapes, sizes, and compositions exist [31, 60], including 0D (*point*), 1D (*line*), and 2D (*interfacial*) defects [40, 60]. The dimensions in which a defect occupies a nanoscale regime define which category each belongs to (Figure 2.9). For instance, an individual lattice atom would be a 0D object, as it occupies a negligible space in each of the Cartesian dimensions. A common interfacial defect is the grain boundary, introduced in Section 2.1, [40]. Voids or bubbles in a lattice would be examples of 3D (bulk) defects, as they do not occupy a nanoscale space [40, 60]. As this work is principally concerned with point defects, greater attention will be given to their details. Two common types of point defect are *interstitials*, where an atom occupies a typically unoccupied off-lattice position, and *vacancies*, where an atom is missing from a typically-occupied on-lattice site [60]. A co-occurrence of these two defects is called a *Frenkel defect pair*, where an atom is displaced into an off-lattice site and leaves a vacancy behind it [31, 61, 62]. Point defects diffuse through their host lattice in what is usually a thermally-activated process [63]. Since defects change the

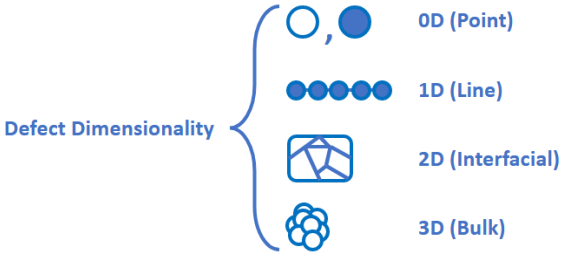


Figure 2.9: An illustration of common defect types and their nanoscale dimensionality. In order from top to bottom: Point defects (vacancy, interstitial), line defects (dislocation), interfacial defects (grain boundaries), and bulk defects (voids/bubbles) [40]

structure of a crystal lattice, they can also enhance or detract from those properties that depend on lattice structure, which makes them both of great concern and interest for defect

engineering [63–66]. Persistent groups of vacancies and interstitials can form voids and dislocation loops in crystalline materials, which can lead to changes in material properties that are detrimental to engineered applications [62, 63]. For example, defects such as voids and dislocations create stress fields in their surroundings that increase the local surface energy, serving as sites for initiation of pitting corrosion [48].

2.2.1 Intrinsic Point Defects

The existence of intrinsic point defects is partially due to the interplay of enthalpy and entropy in minimizing the free energy of a crystal structure during formation, in the same general form as Equation 3.1. The ordering of atoms into the ideal lattice structure decreases the enthalpy of the system, while as temperature rises, the entropic contribution to the stability of the system from point defects becomes increasingly favourable [63]. An additional example of intrinsic point defect formation is the spontaneous formation of Frenkel defect pairs in metal lattices near their melting temperature as the energy of lattice vibrations approaches the same order of magnitude as the defect pair formation energy [31]. However, pairs produced like this often spontaneously recombine and annihilate due to the strain they produce on their immediate lattice surroundings, provided that they are near to each other (0.5-2 nm) [31].

2.2.2 Interstitial Sites

Interstitial sites are off-lattice sites found in crystals, occupied by interstitial defect atoms. Common examples include octahedral or tetrahedral sites, distinguished by the coordination number and location of the site [60]. Both the octahedral and tetrahedral types of interstitial sites are present in BCC lattices as well as the FCC, however they are found in different locations, abundances, and orientations [60]. An atom of the same type as those in the main crystal lattice, found in an interstitial site, is referred to as a *self-interstitial* atom, or self-interstitial defect [60]. The stability of self-interstitial defects varies from system to system, and annealing (heating to allow for recrystallizing) their host system can help to transform or remove them [19, 62].

Tetrahedral Sites

The tetrahedral interstitial site involves an interstitial atom occupying the lattice in a spot between a corner atom and the three nearest face atoms. Tetrahedral sites occur in FCC lattices near each of the 8 corners of the cell, as shown in Figure 2.10 [60]. A self-interstitial defect occupying one of these sites exerts a strong influence on atomic displacement in its surroundings compared to other point defects [19].

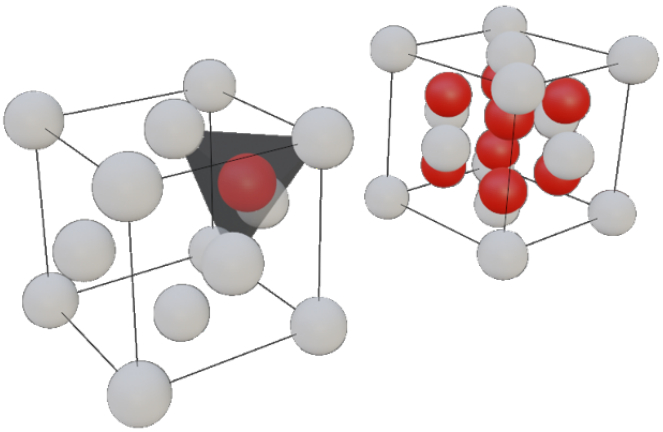


Figure 2.10: 3D view of the FCC tetrahedral defect form, where a self-interstitial atom (red) has found a tetrahedral site (shaded). To the right is an example cell with all tetrahedral sites simultaneously occupied.

Octahedral Sites

The octahedral interstitial defect form occurs in FCC lattices when the interstitial atom is in the centre of the unit cell, where the octahedral site can be defined by the space between the top and bottom face atoms in the cell and the four face atoms around the equator of the cell. Octahedral sites occur in FCC lattices in the centre of the unit cell as well as along each of the 12 edges that outline the cubic cell, as shown in Figure 2.11 [60]. There are a total of 4 sites per cell when fractional sites are accounted for. A defect residing in one of these sites results in the highest stress and changes to atomic potential energy compared to other point defects [19].

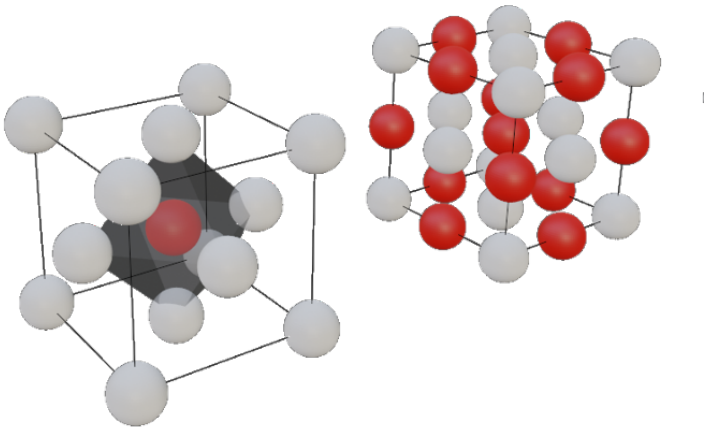


Figure 2.11: 3D view of the FCC octahedral defect form, where a self-interstitial atom (red) has found an octahedral site (shaded). To the right is an example cell with all octahedral sites simultaneously occupied.

2.2.3 Complex Lattice Defects

There are several more categories of crystal defect caused by the presence and movement of self-interstitials inside the lattice. Three noteworthy forms are: *crowdions* [19, 60, 67], *dumbbells* (AKA "split interstitials") [60, 68], and *saddle points* (transition states) [60].

Crowdion Defects

Crowdions are a linear form of point defect, in which the interstitial atom positions itself slightly off-lattice such that many other atoms in line with it are likewise pushed out of their standard lattice positions (Figure 2.12) [60]. The crowdion defect exhibits the longest ranged displacements caused by a single self-interstitial atom [19]. This form is favoured by interstitials in BCC systems [67, 69] such as W, where migration of self-interstitial atoms can occur by alternating dumbbell-crowdion transitions in which a dumbbell or dumbbell variant serves as the transition state [69]. Self-interstitials can also diffuse by crowdion motion in FCC materials [70].

Dumbbell Defects

The *Dumbbell* or split-interstitial is a common state for a self-interstitial defect in metallic systems [60], particularly FCC systems [67, 70]. The dumbbell is created when the interstitial

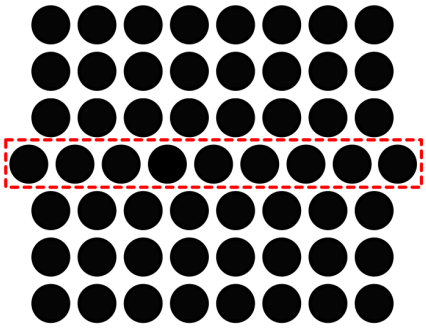


Figure 2.12: A 2D sketch of an arbitrary-length crowdion in an arbitrary 2D lattice, demonstrating the slight site shift and linearity that distinguishes this defect.

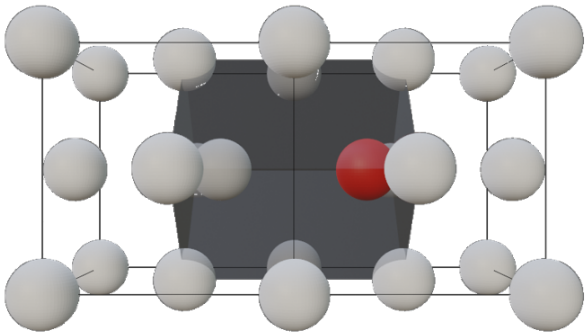


Figure 2.13: 3D view of the FCC dumbbell defect form. A self-interstitial atom (red) has split a lattice site with another atom, creating a dumbbell pair. The area enclosed by the nearest neighbouring atoms is shaded in.

shares a single lattice space with the atom formerly occupying it, splitting the position between the two atoms and creating a shape resembling a dumbbell weight-lifting device (Figure 2.13) [57, 60]. The relaxation volume due to displacement from a dumbbell self-interstitial varies with lattice composition, but is always greater than a factor of 1 atomic volume, values of 1.9 and 1.3 being known for FCC Ni and Cu respectively [57]. In FCC systems the dumbbell tends to align along the $< 100 >$ direction due to the low energy of this configuration [60]. The dumbbell is the least affected by annealing and places less stress on its lattice surroundings compared to the other self-interstitial point defects (in FCC simulations) [19]. In an FCC unit cell there are six sites for a dumbbell self-interstitial to abide in, corresponding to the positions for atoms of each face centre of the cell and oriented

along the face normals [19]. The low migration energy barrier of dumbbell defects (< 0.15 eV) makes them a highly mobile defect form [60].

Saddle Points

The saddle point is an unstable intermediate point in the transition between defect states where the slope of the migration energy landscape has a passing resemblance to a saddle, as during a transition the defect must enter a state of relatively high energy with lower energy defect states on either side of the diffusion path [57, 60, 69, 71]. The form of the defect at the saddle point varies depending on the type of transition and the crystal system [57, 69]. Saddle points are sometimes captured in simulations of self-interstitial migration [69], as seen in Chapter 7.

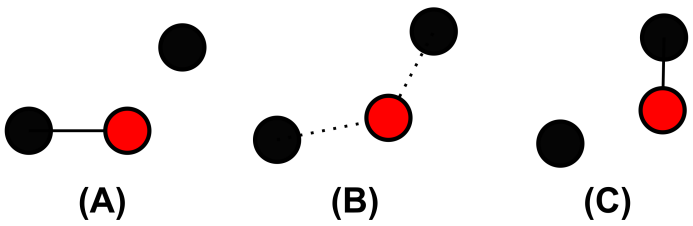


Figure 2.14: A simple representation of a dumbbell saddle point in an arbitrary 2D transition between states A and C. (A) A dumbbell self-interstitial rests in its split site position (B) One atom of the dumbbell (red) begins to interact with a neighbouring atom, caught between two possible dumbbell positions at the saddle point (C) The saddle point has passed and the dumbbell transitioned to a new configuration

2.2.4 Lattice Vacancies

A vacancy is the opposite to an interstitial in a few ways; a normally populated lattice site is instead empty, the relaxation volume is roughly 70% of the original space [57] (values of -0.2 and -0.4 reported for Cu and Pt respectively [60]), and the formation energy is lower [60]. Surrounding atoms attempt to occupy the empty site in a thermally activated process, causing the vacancy to migrate through the lattice [63]. Interstitial atoms and vacancies can annihilate, with the interstitial atom filling the vacant lattice site, a key mode of recovery for metals exposed to radiation [31]. Vacancies are less mobile than their interstitial counterparts by several orders of magnitude, owing to a larger migration energy barrier (> 0.5 eV) [60].

2.2.5 Trapping States

Due to the configurational entropy of an HEA or multicomponent alloy environment, there can exist states in the distorted lattice with very negative lattice potential energies that "trap" vacancies or self-interstitials as they diffuse through the lattice [15–17, 72, 73]. Multiple proposed explanations and types exist. One is that trapping states for vacancies are relatively low entropy and made mostly or entirely of a single type of atom [17]. In this case, the atoms are thought to pack tightly due to reduced lattice distortion from differences in atomic radius, inhibiting the escape of vacancies by increasing the migration energy barrier [17]. Recent work has supported the notion of both thermal (vacancy trapped by a favourable position) and kinetic (vacancy trapped in a favoured oscillating pathway between states) types of trapping for vacancies [73]. For FCC alloy self-interstitial dumbbells, oversized solute atoms in the lattice are thought to have a trapping effect due to their contribution to local strain [72].

2.2.6 Defect Migration

Defects migrate through their host crystals by diffusing through the lattice, displacing, replacing, or squeezing atoms in their path using a variety of mechanisms [60]. Diffusion coefficients for self-interstitials are significantly higher than vacancies [61, 74, 75], roughly 4 orders of magnitude in the example of high temperature Cu (7E-2 $\frac{cm^2}{s}$ for self-interstitials and 5E-6 $\frac{cm^2}{s}$ for vacancies) [60].

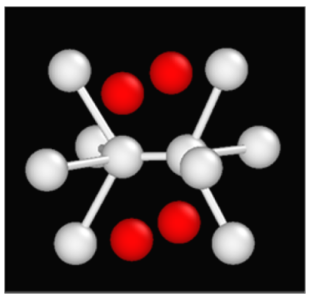


Figure 2.15: A representation of the dumbbell self-interstitial from OVITO. The middle two silver atoms are the dumbbell itself. The surrounding silver atoms on either side are the first 4 nearest neighbours to the atoms [60]. The girdle of four atoms highlighted in red are the second nearest neighbours to the dumbbell pair.

The movement of a self-interstitial defect through available lattice sites is expressed by the

diffusion formula Equation 2.4 in the absence of an external stress field, where the migration energy E_m to each site is considered equivalent and Λ_0 is a placeholder constant, δ_{ij} is the Kronecker delta, and β is the thermodynamic beta [61]. The available lattice sites for a dumbbell transition are indicated in Figure 2.15 [60].

For the vacancy, Equation 2.5 applies instead, where v_{LV} is the average lattice vibration frequency, E_v^m is the vacancy migration barrier energy, and S_v^m is the vacancy migration entropy [61]. In both formulae the constant d_o is the nearest neighbour distance of the cell.

$$D_{ij} = \frac{1}{6}\Lambda_0 d_0^2 e^{-\beta E_m}\delta_{ij} \tag{2.4}$$

$$D_V = v_{LV} d_0^2 e^{S_v^m} e^{\frac{-E_v^m}{k_B T}} \tag{2.5}$$

Defect Movement at Grain Boundaries

Over time, a defect may diffuse through the volume of the grain where it formed and approach the grain boundary. The behaviour of the grain boundary is to act as a defect sink, where migrating defects can be permanently or temporarily integrated [61,67,76,77]. This is true of some of the other extended defects in the crystal, such as dislocations and interfaces with nonmetallic inclusions [61]. The dynamics of the interaction at grain boundaries depends on a variety of variables including defect size, grain boundary orientation, and irradiation intensity [67]. Studies by a combination of MD and KMC (see Section 4.1 and Section 4.3 for more details) have shown that the geometry of the grain boundary can change the potential energy landscape of the lattice such that the favoured configurations and/or migration dynamics of nearby migrating defects changes markedly (ex. dumbbells to crowdions in FCC Cu [67]) [67,77]. Imbalance of the interstitial-vacancy equilibrium at grain boundaries is expected due to the greater mobility of interstitials compared to vacancies [67].

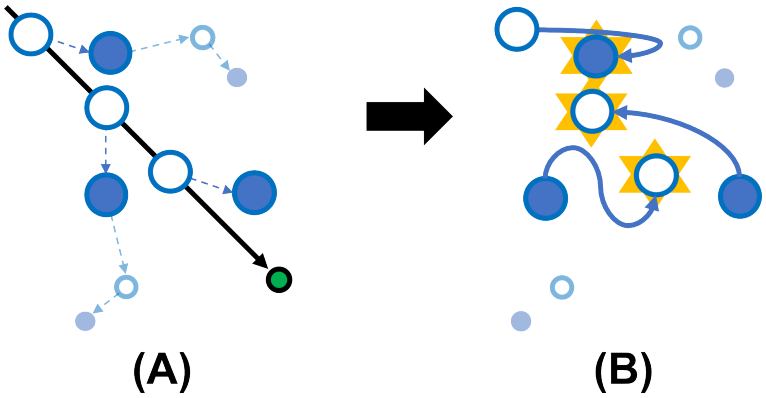


Figure 2.16: An illustration of an impinging particle (green) creating a series of primary and secondary (faded) knock-on atom displacements, followed by the recombination of some induced self-interstitials (blue) with the vacancies (white) in the recovery phase.

2.3 Radiation Damage in Metallic Structures

The irradiation of metallic structures by neutrons, electrons, protons, and other forms of particle radiation damages the lattice structure by displacing atoms in the path of the impinging radiation, as illustrated in Figure 2.16 (A) [31,60,74]. The initial atomic displacements in a collision cascade occur on a timescale of t <= 1 ps, followed by a heat spike and recovery phase roughly on the order of 1-100 ps [31,78]. The main damage produced by simple particle radiation impacts is the formation of Frenkel defect pairs, introduced in Section 2.2, which consist of self-interstitial atoms and corresponding vacancies [16,31,60]. Most of these displacement-induced defects are repaired by *athermal recombination* in the recovery phase (see Figure 2.16 (B)), using kinetic energy contributed to the lattice by the Primary Knock-on Atom (PKA) (the recoiling initially-displaced atom) and other knock-on effects [31]. Afterwards, remaining defects follow a thermally-driven migration phase [31]. This remnant fraction has a maximum value for neutrons of roughly 30% of the initially-induced defect population at higher energies (> 10 keV) [31,79]. Damage of this kind is generally reported in the form of displacements-per-atom (dpa), calculated from the initial energy contributed to the lattice by irradiation as well as the threshold energy for displacement of the constituent atoms (averaged in the case of multiple displacement energies) [31]. The defects that persist past the initial irradiation and recovery of the material are those in which the displacement of each item in the pair is beyond their spontaneous recombination distance [31,75].

The presence of unresolved self-interstitials and vacancies and their subsequent clusters (dislocation loops, voids, etc. [80]) leads to an increase in material hardness (by impeding dislocations) and can also cause dimensional instability in the form of decreases in density (due to void swelling) [19, 61, 74, 75, 81–84] and changes in shape due to irradiation creep (deformation that occurs in crystals under load) [40, 74]. Over time, the contribution from irradiation to the hardening of metals and alloys can make them more vulnerable to brittle fracture, which is of great concern for energy applications [82, 83, 85, 86].

Defects can be captured at sinks such as grain boundaries (Figure 2.17) or precipitates, which can assist in recombination of defect pairs [31, 74]. For this reason, exotic materials such as nanocrystalline alloys have attracted attention due to the proximity of grain boundaries to any possible radiation-induced defect pairs, and have shown experimental evidence of resistance to amorphization and grain growth due to irradiation [31].

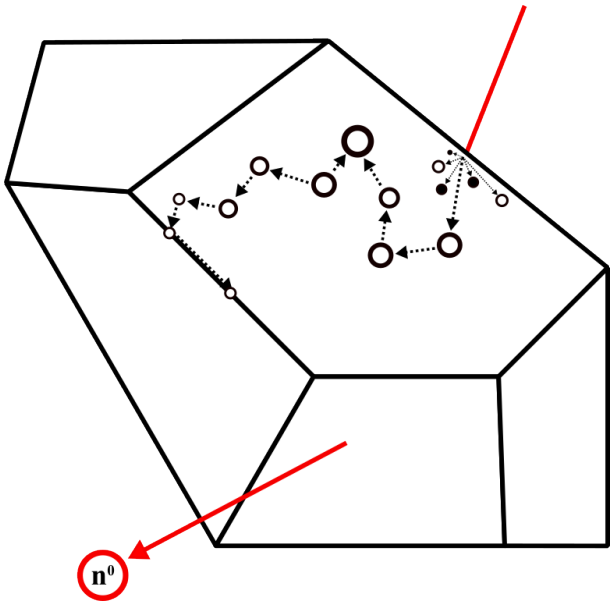


Figure 2.17: A diagram of radiation-induced defect migration. A high energy particle (a neutron for instance) collides with a crystalline lattice, transferring energy to particles along its path and displacing them from their sites, creating vacancies (hollow circles) and self-interstitial defects (full circles). Some of these persist long after the initial incident, and thermal fluctuations allow them to move through the lattice, where they can combine or, as in this figure, migrate to grain boundaries (solid black line lines) within the crystal to be captured, recombined, or released.

Chapter 3

High Entropy Alloys (HEAs)

A High Entropy Alloy (HEA) is an alloy of five or more metals mixed in equal or near equiatomic proportions that form a diverse and useful space for new materials discovery [3, 16, 87, 88]. The common elements used in HEAs tend to be familiar row 4-6 transition metals such as Fe, Cu, Ni, or Al [17], but application-specific requirements may trend towards more or less exotic compositions (e.g. nuclear materials) [16]. The definition of the term HEA has been somewhat flexible in the past and applied to alloys with less than five elements, as well as a range of mixture proportions [17, 88]. The requirement for equiatomic proportions is not strictly upheld [15], and a range of 5-35% is commonly given as acceptable [16, 88]. An alternative definition of HEA based on total configurational molar entropy S^{SS} exists, dividing multi-component alloys with solid solution phases into categories of low ($S^{SS} < 0.69R$), medium ($0.69R < S^{SS} < 1.61R$), and high ($S^{SS} > 1.61R$) entropy, where R is the gas constant [88]. This definition is challenged by some practical considerations [88] and thus the original compositional definition will be used herein.

In earlier works, HEAs were thought to stabilize in a simple solid solution lattice structure by sufficient entropy of mixing [89]. The sign of the *Gibbs free energy of mixing* of a system describes the spontaneity of the formation of mixtures; a negative free energy indicates a mixture will form spontaneously from a given system. It is dependent upon three properties of a system: temperature, *entropy of mixing*, and *enthalpy of mixing*. The entropy of mixing is the change in entropy of a system when multiple substances are allowed to mix freely and reach equilibrium. The enthalpy of mixing is described by changes in the pressure, volume, and/or energy of a system in response to mixture (zero in the case of non-interacting substances). The formula for the free energy of mixing is shown in Equation 3.1, where

ΔG_{mix} is the Gibbs Free Energy of mixing, ΔH_{mix} is the enthalpy of mixing, T is the temperature, and ΔS_{mix} is the entropy of mixing [87,89].

$$\Delta G_{mix} = \Delta H_{mix} - T * \Delta S_{mix} \tag{3.1}$$

More recent work has shown that Equation 3.1 is not sufficient to explain the stabilization of the solid solution, and that HEAs form with greater phase variety than was initially anticipated [13,87,89], prompting alternate conditions for stabilization of HEA solid solutions [90]. For example, even within a single HEA that demonstrates a dominant solid solution phase, e.g. AlCoCuNiTiZn, grains can exist with BCC or FCC solid solution phases [87]. Thus not all HEAs can be expected to form pure solid solution structures [87], though solid solutions are often considered a desirable ideal and used in HEA models [15–17,33,88].

In order to maximize the entropy of mixing, HEAs are commonly synthesized using either plasma sintering techniques or mechanical alloying (Figure 3.1) [87]. The phase of the HEAs obtained from these can vary, including simple phases such as FCC and BCC as well as amorphous metallic glasses [87]. Obtaining HEA materials mostly composed of a solid solution phase is an ongoing subject of research [89,91], as is exploring the potential of multi-phase HEAs [92].

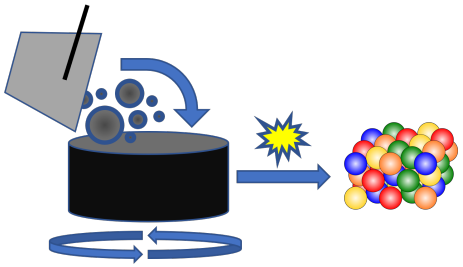


Figure 3.1: An illustration of the mechanical alloying process using a ball-milling technique, a common HEA synthesis path. Finely powdered compounds and hard balls of various sizes are added to a rotating drum, where collisions between/with the balls alloy the different powders together.

Part of the value of HEAs is the opportunity to tailor compositions to meet material needs as well as discover synergistic combinations of elements with novel properties [16,56,88]. This is the case in lightweight HEAs (density <= 3 g/cm3), where mechanical alloying allows HEAs to be made using light reactive metals such as Na, Mg, etc [87]. It is worth noting

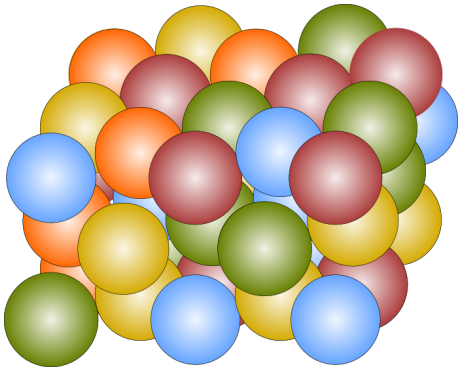


Figure 3.2: A hypothetical HEA composed of 5 elements in roughly equal proportions randomly positioned in a FCC (111) solid solution phase slab.

that considerations of interatomic solubility are still important to the composition design process, as discussed briefly in Subsection 2.1.3 [91].

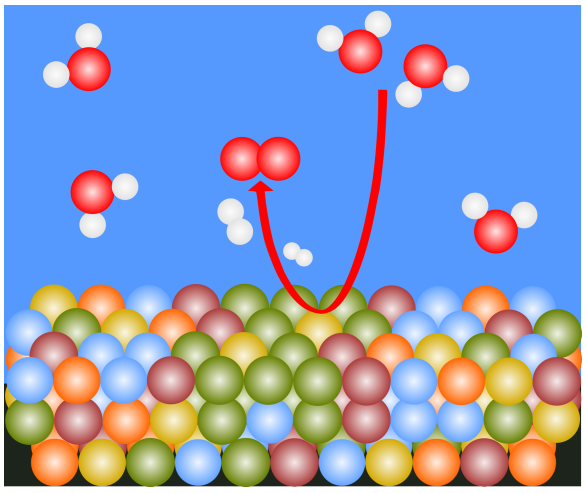


Figure 3.3: An illustration of an arbitrary 5-element HEA in solution, catalyzing an Oxygen Evolution Reaction (OER) from nearby water molecules. In this graphic, a site near a yellow atom on the surface was the most favourable, adsorbing nearby water molecules and splitting them into gaseous oxygen and hydrogen.

Catalyst applications of HEAs are also of interest, as the large configurational diversity of HEAs creates a distribution of surface adsorption energies, a useful space for discovery of novel catalytic environments (see Figure 3.3, Figure 3.4) [3, 56, 59]. In addition to their use for discovery, the alloying of extra elements to form an HEA has been shown to provide improved yield in the case of known catalytic bimetallic materials with poor baseline miscibility [56].

Lastly and most pertinent to this work, beneficial attributes of a number of HEAs include

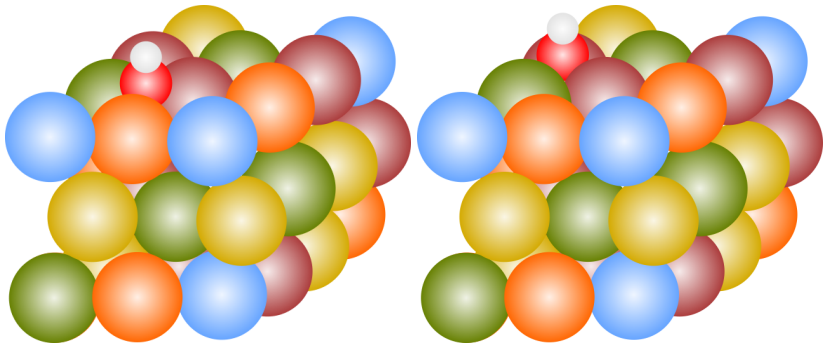


Figure 3.4: The two types of hollow site on the FCC {111} surface, the FCC (left) and HCP (right) sites. Sites such as this have a large composition space in HEAs when considering the interaction of 9 and 7 nearby atoms with adsorbates respectively.

properties such as high strength [87, 89, 92], low neutron activation [89, 91], resistance to radiation damage [89], and high thermal stability [87, 89].

3.1 Radiation Damage in HEAs

The prospect of creating high strength tailored alloys out of low-activation materials has generated interest in the exploration of HEAs for next-generation nuclear energy applications such as fission fuel cladding and fusion reactor lining, as conventional materials may produce excessive waste or be mechanically unsuitable [16, 89, 91]. These applications will subject materials to extremes of both temperature (> 600 C for fission) and neutron irradiation (14.1 MeV for fusion) [16].

The basis for discussion of irradiation damage resistance in HEAs is the series of explanations for reported observations of phase stability under irradiation via "sluggish diffusion" and amorphization-recrystallization "self-healing" [81, 89, 93, 94]. The sluggish diffusion property of HEAs (particularly slow vacancy diffusion compared to pure substances and conventional alloys) may account for reduced defect accumulation [17, 81]. Previous work on HEAs and radiation damage has examined both their purported self-healing properties as well as their phase stability under irradiation. Work with HEAs such as Zr-Hf-Nb, Co-Cr-Cu-Fe-Ni, and Al-Co-Cr-Fe-Ni under charged particle bombardment indicated persistence of main phases (BCC, FCC, and BCC in order) in the material up to at least 10, 40, and 45

dpa respectively, with little or no observable grain coarsening [81]. Low void swelling was also reported [81].

The existence of unusually slow diffusion is contested, at least insofar as it applies exclusively to HEAs, and it has been proposed that a highly diverse lattice displacement energy landscape is a general property of multicomponent alloys which contributes to slow diffusion [16, 73]. In addition, claims of extraordinary radiation resistance have not been conclusively shown to be a trait of HEAs as a whole, or to be limited to HEAs compared to other multi-component alloys, and as such many explanations are deficient [15, 16, 73, 88]. The chemical disorder in HEAs also makes studying crystal defects more difficult, as defect shape and formation/migration energy are dependent on the local environment [17, 95]. The impact of lattice distortion on defect formation and behaviour has been credited with some of the properties of interest in HEAs and is currently a subject of scrutiny [16, 88]. For example, it has been shown in work with simulated HEAs that at room temperature the distortion of the lattice due to thermal fluctuations can be several times stronger than the inherent lattice distortion due to the size difference of the constituent elements [33]. It has also been proposed that a chemically diverse lattice environment changes the distribution of migration energies for defects, leading to an overlap of the distributions and a favourable impact on irradiation defect annihilation [96].

In spite of the lack of consensus on the diffusion behaviour and radiation resistance of HEAs, as well as the massive complexity of the compositional design space, the interest generated by the study of compositional entropy and its impact on defect diffusion and phase behaviour continues to drive the field of HEA materials science, as does the promise of tailor-made alloys with enhanced properties [16, 88, 93].

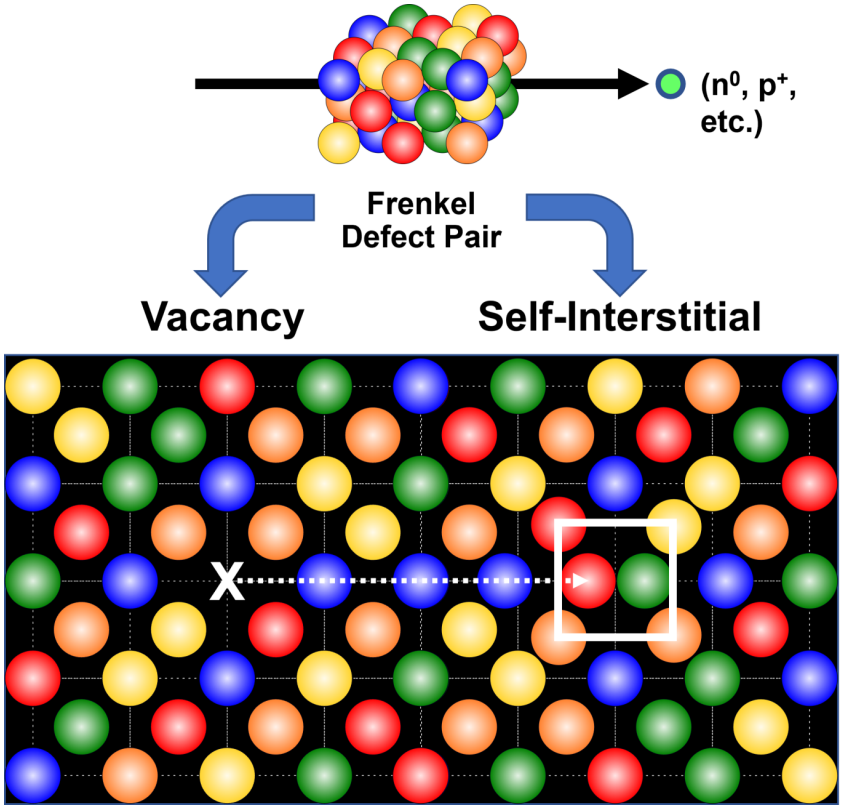


Figure 3.5: An illustration of the formation of a Frenkel defect pair in a HEA due to incident radiation. Incident radiation moves through the crystal lattice, producing a disrupted lattice in its wake where atoms were knocked from their sites (bottom left) to arrive elsewhere (bottom right). The empty site is the vacancy defect and the itinerant atom is the self-interstitial component of the Frenkel defect pair.

3.2 Example - Configurational Entropy

Imagine a Region of Interest (ROI) of 12 1st nearest-neighbour atoms and a vacancy in an ideal FCC metal lattice. In an ideal pure metal with a vacancy defect, each of these atoms has an identical probability of being chosen for a transition and diffusion is uniformly random, as informed by Equation 2.5. In the HEA environment equivalent, the elemental configuration in the local environment affects the migration rate as well [16, 28, 60, 61, 97]. The pure environment has a single possible configuration of atoms, whereas an HEA with just 3 atoms has $3^{12} = 531441$ configurations. Many are identical by symmetry (for instance, each possible state has at least 6 simple mirror duplicates, one for each face of a cube).

Chapter 4

Material Simulations

4.1 Molecular Dynamics (MD)

Molecular Dynamics is a statistical mechanics technique commonly used in materials science for deterministic simulations of the evolution of complex atomic systems using an interatomic potential and classical Newtonian equations of motion [17, 25, 34, 98]. This allows for accurate simulations of the behaviour of atomic systems composed of thousands to millions of atoms on the timescale of atomic vibrations (fs) between steps, so long as quantum effects and electronic motion are relatively unimportant to the system dynamics [25, 71, 98]. The practical weaknesses of MD are its computational expense and the shortness of the timescales it can accurately capture [98]. The latter failing is an unfortunate result of the fact that an analytical solution to the motion of atomistic systems is not feasible, and so in order to numerically solve the problem of their evolution, short timesteps are required to give the best accuracy [98].

With positions for all atoms in the system known as well as their masses, and assuming that interactions between particles are pairwise and additive, the force on each atom in the MD system can be calculated from the negative gradient of the potential energy (i.e. Lennard-Jones potential [34]) of the particle system $-\nabla E$ using Equation 4.1, where $X(t)$ are the particle positions [98]. Additional many-body terms are also used in MD potentials to more accurately represent interactions in particular systems (i.e. crystalline solids [34]) [34, 71].

$$F(X) = -\nabla E(X(t)) \tag{4.1}$$

The computational cost in terms of the worst-case time ("Big O" notation) of MD is O(N) for short-range interactions and is quite similar for longer-range coulomb interactions in various models (the most expensive of which costs O(N log N)) where N is the number of atoms being simulated in the system [71]. The tremendous gains in computational speed and memory in the past several decades are largely responsible for the rising popularity of MD to the present, a fact that will be discussed further with Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) [71, 98]. The time scales accessible to MD simulations are also limited by the number of atoms being simulated. As N increases, the time required to run MD can make observations over certain time scales infeasible [71].

The force field potential for MD is the most vital component, since the accuracy determines the quality of the simulation [17, 34, 71, 98]. Recent advances in technology as well as ML techniques have also improved this aspect of MD by allowing larger and more complex models of the atomic potentials to be generated, which in turn increase the applicability of MD simulations to materials research [71]. Interatomic potentials for HEA systems are rarer for systems composed of multiple elements and one technique for creating them is to combine existing potentials from binary and ternary systems to cover all possible interactions in the system (e.g. For hypothetical HEA FeCuNi, use potentials for alloys FeCu, FeNi, and CuNi) [17]. As discussed in Chapter 1, ML models have become increasingly effective for predicting appropriate interatomic potentials for HEA systems [8, 17, 33].

4.2 Markov Chains

A *Markov Chain* is a sequence of random variables in which the selection of the next item depends solely on the current state of the system and is not conditional on the history of previous selections [99]. For example, the next step in the path of a free rubber ball in a very dense and busy crowd depends only on its present location, not on where it had previously been in that crowd.

The environment of a crystal lattice with a defect is best described as an *infrequent event system*, where transitions between states are rare in comparison to the thermal fluctuations in the system [25]. Due to the long residence times in each state and vibration of the system,

the evolution of the system through multiple states constitutes a Markov Chain, as history of previous transitions is erased [25]. Note this assumption does not hold in systems that have been disrupted by radiation damage for a short time until the system has regained thermal equilibrium (See Section 2.3) [25].

4.3 Kinetic Monte Carlo (KMC)

Monte Carlo methods are a class of algorithms that make use of independently-generated random numbers to approximate the behaviour of deterministic systems [25]. Repeated random numbers are drawn to make decisions under certain constraints, causing an approximated probabilistic system to evolve as if from the true probability distribution function of the system instead of from the random number process, by way of the *law of large numbers* [100, 101]. The most well known Monte Carlo method is the Metropolis algorithm; a different example of this class that is prevalent in materials science is KMC, a rejection-free Monte Carlo method applicable to processes such as point defect diffusion and others that occur with a set of possible outcomes and frequencies [10, 11, 28]. It has seen use in materials science for a variety of tasks, particularly simulating radiation damage [25, 28, 31], in part due to its applicability to bridge the timescale gap between methods such as MD and the longer-term evolution of irradiation damage (10^{-12} to 10^{-3} s) [28]. A major assumption of KMC is that when a system transitions to a new state, the memory of the prior state is lost, due to remaining for a long time relative to the lattice vibrations [25]. This makes the trajectory of states ω formed from KMC a valid Markov Chain [25]. Each of the possible transitions from state i to state j is defined by a rate constant k_{ij} and the summation for all j is called the *escape rate* k_{tot} [25]. Typically, the rate constants are based on studies of the potential energy landscape between state *i* and possible state *j* [25]. KMC is performed by taking a system and defining a finite list of states it could evolve into, as well as the rate at which it is likely to choose each of these states [25]. Then a Markov Chain of states, a trajectory ω, is made by using independent uniform random values to pick a new state at each step based on k_{tot} [25]. To illustrate how KMC works in greater detail, the derivation shown by Voter is summarized in Subsection 4.3.1 [25].

4.3.1 Example - Reasoning Behind KMC

Due to the memory-less property of the KMC procedure there is the same probability of the system making a transition between states in any given moment [25]. Given this assumption, the transition from one state to another is a random and independent event similar to atomic decay where the probability of the system being in its original state at some time t is $p(t)_{Survival} = e^{-k_{tot}t}$. The distribution $p(t)$ for this system represents the *time of first escape*, as the system can only escape from its current state once. The probability that the state has changed by time t is $p(t)_{escape}$ (the complement of the survival probability function), which represents the integral of the distribution of escape times from time 0 to time t. The distribution $p(t)$ can be obtained by taking the derivative $f'(x)$, yielding $p(t) = k_{tot}e^{-k_{tot}t}$, an exponential distribution with $\lambda = k_{tot}$. A vital component of the KMC method is that this form extends to each of the individual pathways as well, i.e. $p(t)_{ij} = k_{ij}e^{-k_{ij}t}$ with $\lambda = k_{ij}$.

In order to evolve a KMC system in time, a value t_{draw} is drawn from the exponential distribution of times associated with the current state i. This is done by non-uniform random number generation with the quantile function of $p(t)$. For a random variable X with a continuous monotonic Cumulative Distribution Function (CDF), the quantile function (inverse cumulative function) $Q(p)$ gives the support x for which the probability is equal to p, or $Q(p) = x|P(X \leq x) = p$. Taking the cumulative distribution of $p(t) = k_{tot}e^{-k_{tot}t}$ from $t = 0$ to an arbitrary time t yields the form $p(t) = 1 - e^{-k_{tot}t}$, the cumulative distribution of the exponential distribution. To obtain the quantile function for use in obtaining appropriately distributed random times, the function is inverted by solving for the arbitrary time t which is dependent upon both p and k_{tot}, $t = \frac{-ln(1-p)}{k_{tot}}$. Now the KMC system can evolve according to $t_{draw} = -\frac{1}{k_{tot}}ln(r)$, where r is a uniform random variable on the interval $(0, 1]$ (0 is not included in the range since $ln(0)$ is undefined) [25].

Next the KMC procedure involves stacking the rates k_{ij} that make up k_{tot} atop each other to form intervals of length k_{ij} across a range $[0, k_{tot}]$ [25]. A uniformly-distributed random number (also called r) can be generated and multiplied by k_{tot} to achieve the desired effect of picking a transition to the state j indicated by the rate constant k_{ij} of the interval that r exists in Figure 4.1 [25].

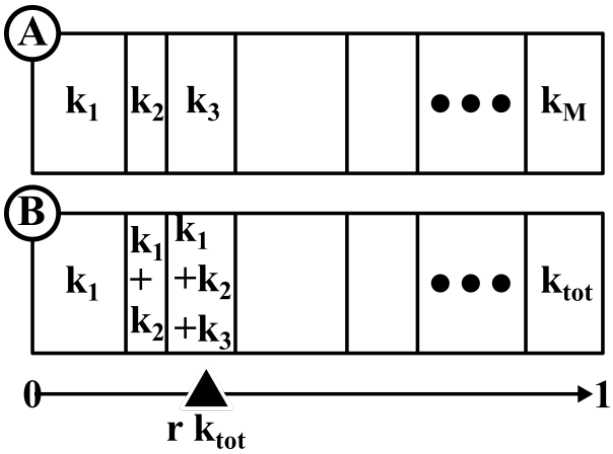


Figure 4.1: An overview for the KMC process. (A) The domains occupied by each rate constant, with a width k_i. (B) The programming implementation of the technique, using sums to delineate boundaries between different k_i domains [25]. The rk_{tot} covers the range of values from (0,1) and is the means by which the next state is determined.

4.3.2 Example - KMC Draw

An arbitrary hypothetical state i in a trajectory has possible transitions to 4 new states j, which have associated rates k_{ij} (in 1/fs for this case) for transition of K = [0.05, 0.25, 0.15, 0.1]. The k_{tot} for this environment is then $\sum_{j=1}^{N} = 0.55\ fs^{-1}$. Next a random uniform variable r is drawn, let it take a value of $r = 0.5$ here. This is used to get the time evolution of the system for this step by inserting it into the draw time equation $t_{draw} = -\frac{1}{0.55} ln(0.5) \doteq 1.26 fs$. Another r is drawn, this time let it be $r = 0.1$. The intervals that determine the next state using this r are based on the 4 transition rates such that the path of k_2 will be chosen, as 0.1 is within its interval of $[k_1, k_1 + k_2] = [0.05, 0.30]$. The system will then evolve to a new state by the path associated with k_2 and the trajectory will move forward by 1.26 fs. A new k_{tot} will be calculated for the new state, and the process repeated to arbitrary length depending on the goals of the simulation.

4.4 Combining MD & KMC

A weakness of KMC is that the accuracy and usefulness of the technique is highly dependent on a thorough understanding of the possible states and interactions in the system. In this way, unobserved or unknown behaviours of the real system can compromise predictions

made with this technique KMC [25]. The shortness of MD steps poses a problem for observing systems effectively over the course of longer timescales than ns [34]. Work on simulated materials at multiple timescales or levels of detail benefit from hybrid strategies involving both KMC and MD, generally such that MD provides the initial data that is used to determine appropriate rates for KMC [28, 29, 31, 102].

Chapter 5

Graphs

In order to better understand structures and their relationships to physical properties, it is necessary to introduce the concept of a *graph*. An *undirected graph* is a pairing of vertices V and edges E, represented in notation here as $G = (V, E)$, with a set of edges composed of unordered pairs of vertices (e.g. edges xy and yx describe exactly the same edge) [103]. This means that all edges are unique as well. Generally, no self-loops are allowed in simple undirected graphs (e.g. edges xx or yy are forbidden) [103]. The *degree* of a vertex is simply the number of edges including that vertex in a graph G [103]. The number of vertices in a graph is n and is also called the *order* of that graph [103]. An *independent set* is a set of vertices such that none are mutually adjacent [103]. The opposite concept is the graph *clique*, a group of vertices which are all adjacent to one another [103].

Graphs can be grouped into several categories based on the nature of their node and edge features. Figure 5.1 displays several of these. This work only considers graphs that belong to the undirected, heterogeneous, and static categories.

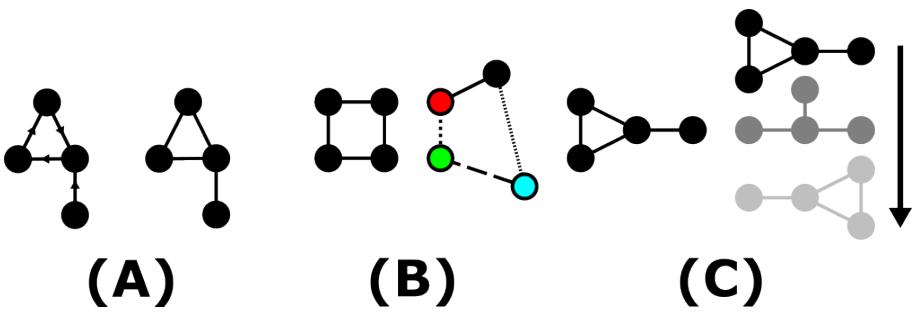


Figure 5.1: Common graph types: (A) Graphs in which edges have directions are called *directed*, and those that do not are *undirected* (B) Graphs with identical node and edge types are *homogeneous*, otherwise they are *heterogeneous* (C) Graphs with structure that does not change in time are *static*, whereas graphs that evolve over time are *dynamic*.

5.1 Subgraphs

One more key concept is that of *subgraphs*. A subgraph is a graph H that is produced from a graph *G* by removing vertices and edges, such that all the vertices and edges in *H* are present in *G*, *V(H)* $\subseteq V(G)$ and *E(H)* $\subseteq E(G)$ respectively [103]. There are several types of subgraph, but the one of greatest relevance to this work is the induced subgraph, which shares all the same edges incident on a vertex as those in the parent graph, except edges removed by deletion of vertices, $E(H) = \{xy \mid x,\ y \in H,\ xy \in E(G)\}$ [103]. For visual elucidation of each of the graph concepts, please refer to Figure 5.2, and Figure 5.3.

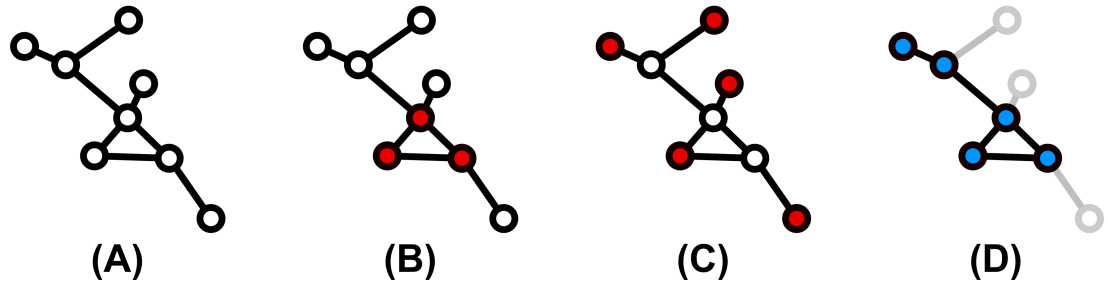


Figure 5.2: Graph concepts in visual form: (A) A graph *G*, composed of 8 vertices and 8 edges, where the highest order vertex has order 4 (B) The Maximum Clique of *G*, composed of 3 vertices (C) The Maximum Independent Set of *G*, composed of 5 vertices, and (D) An induced subgraph *H* of *G*.

In order to clearly discuss how subgraphs play a role in this work, three additional concepts need to be explained, adjacency, isomorphism, and similarity. The concept of *adjacency* is fairly self-explanatory, and is usually represented in the form of a binary *adjacency matrix*, where zeros and ones in the i^{th} row represent which nodes are connected to the j^{th} node in the graph. See the following figure for a trivial example. Examining the 1st row shows that node one is connected to the 2nd and 3rd nodes of the graph [103].

Isomorphism exists between graphs *G* and *H* if there is a bijection (1-1 mapping) of vertices *V(G) –> V(H)* that preserves adjacency [103]. This analogous to saying *G* = *H*. Please see Figure 5.4 for example, where the nodes 1,2,3,4 of graph *G* have an exclusive bijection to nodes 4,3,2,1 respectively that would reproduce an identical adjacency matrix.

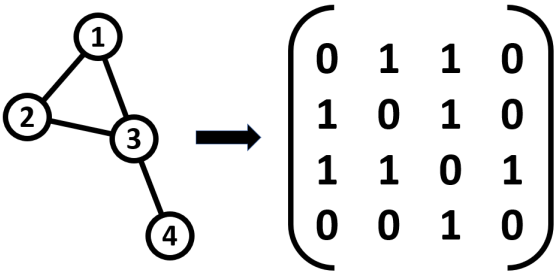


Figure 5.3: Graph concepts in visual form: A 4x4 binary matrix corresponding to a small graph of order 4. Values of 1 indicate edges existing between vertices with corresponding row and column numbers (e.g. row 1 and column 2 and 3, etc.)

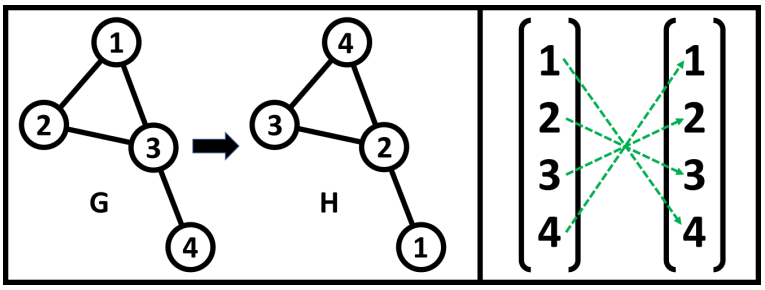


Figure 5.4: Graph concepts in visual form: Two graphs G and H (left) which are isomorphic under the conditions of the bijection indicated (right)

5.2 Graph Similarity

The term *graph similarity* describes a selection of techniques for determining how close two graphs are to being isomorphic with each other [104–107]. The concepts of bijection (Figure 5.4) and subgraphs (Figure 5.2) (D) provide the beginning of a more concrete definition. One way to determine similarity aside from only using graph isomorphism is to compare graphs to check for *common subgraphs* [104]. These are subgraphs that are isomorphic between all graphs being compared [104]. The largest subgraph that is present across the compared graphs is called the Maximum Common Subgraph (MCS) [104].

5.3 Computational Graphs

A *computational graph* is a form of directed graph where each node represents a value (which can be scalar, tensor, vector, etc) and edges indicate a functional relationship between the

value of the parent node and the child node that they connect [108, 109]. Different nodes can use different functions or operators to determine their value (Figure 5.5). Computational graph representation and nomenclature are commonly used in discussion of ANN structures [108, 109].

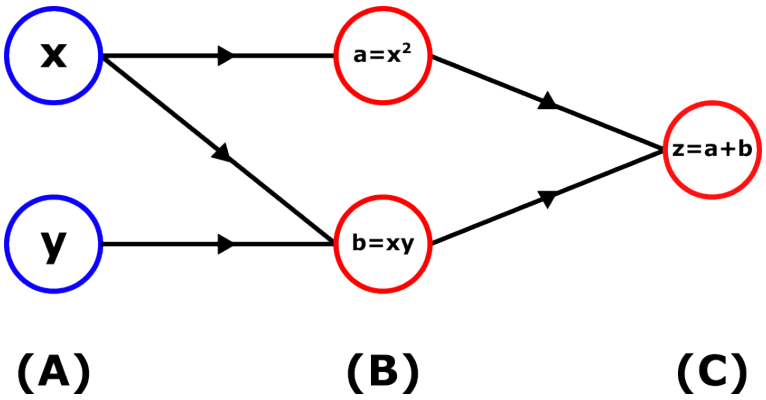


Figure 5.5: An example of an arbitrary computational graph. Nodes are shown as circles, with nodes that hold variable values and nodes that perform operations having perimeters of blue and red respectively. Information in this graph flows from the left to the right nodes.

Chapter 6

Machine Learning (ML)

The field of Machine Learning (ML) is based upon the use of mathematical models that find some mapping between a set of inputs (features) and the corresponding outputs (labels, targets) [110, 111]. In order to grasp the core concepts of ML, some background material will be addressed, beginning with statistical models.

6.1 Statistical Models

A *statistical model* is a probability distribution, Φ, constructed by imposing constraints on a distribution, Ξ, that has generated a known random data sample, ξ [112]. A *parametric model* is a statistical model that is defined by a vector of parameters, θ, from the set of parameter space, Θ [112]. A parametric model may be used, as in this work (Section 8.3), to extract an estimator for the true value of the parameters that generated ξ, $\theta_o \in \Theta$ [112].

6.2 Conditional Models

Parametric models can be used to describe the probability distribution responsible for the dependency between coupled data points (such as the relationship between x and y in $y = mx+b$) [112]. A *conditional model* (or discriminative model) is a statistical model in which input and output vectors are separated in the sample data set as $\xi = [xy]$ and the outputs are used to impose conditional restrictions on the estimate of the generating distribution (i.e. the parent distribution is treated from a standpoint of $P(y|X)$) [110, 112]. The conditional model can then be used in two ways, for *regression* or *classification* tasks (Figure 6.1) [110, 112]. Regression tasks are broadly categorized as either linear or nonlinear depending on what

form the relationship between the dependent and independent variables is believed to take [112]. The relation between x and y in the slope equation is a trivial example of a linear regression type of problem composed of only parameters m and b [112]. Classification tasks also typically fall into two categories, being either binary classification, (the logistic or logit model), or multiclassification [112].

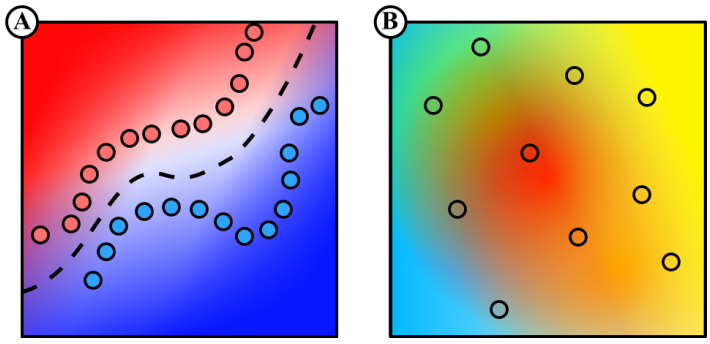


Figure 6.1: (A) Classification task decision boundary (dotted line), drawn across 2D space with 2 colour-coded classes represented by a set of circular targets (B) A regression task, showing a continuous distribution of predictions based on 10 targets in 2D.

6.3 Linear Regression

Linear regression is predicated on the assumption that the output variables have some linear relationship to the inputs that can be expressed in the form $y_i = x_i\beta + \epsilon_i$, where y_i and x_i are the input vector and output scalar respectively, β is a vector of coefficients that serve as parameters for the regression, and ϵ_i is an inherent and unknowable uncertainty in the system [110]. An instance x_i from a data sample for linear regression can be represented as a vector of K values, $x_i = [a_1 ... a_K]$, and the parameters for the regression β as $\beta = [\beta_1 ... \beta_K]^T$ which provide a relation to the output y_i of $y_i = a_1\beta_1 + ... + a_K\beta_K + \epsilon$ [113].

With N data points $(y_i\ x_i)$ a design matrix X (not to be confused with the random variable X) can be constructed in which each column represents the *kth* characteristic value from the data and each row contains all the information about the input of the *nth* sample [113]. Similarly, the output data and parameters can be composed into column vectors as shown in Equation 6.1 [113]. The linear regression problem then becomes $Y = X\beta + \epsilon$, which can be solved to yield an estimator of the optimal parameter vector β to fit the data in the

sample [113].

$$Y = \begin{bmatrix} y_1 \\ ... \\ y_N \end{bmatrix} \quad X = \begin{bmatrix} x_1 \\ ... \\ x_N \end{bmatrix} = \begin{bmatrix} x_{11} \, ... \, x_{1K} \\ ... \, ... \, ... \\ x_{N1} \, ... \, x_{NK} \end{bmatrix} \quad \beta = \begin{bmatrix} \beta_1 \\ ... \\ \beta_K \end{bmatrix} \tag{6.1}$$

6.4 Logistic Regression

In cases with discrete classes of outcome, *logistic regression* is a common technique. In order to understand logistic regression, it is necessary to discuss the basic case of a binary outcome and then extend it to additional classes.

6.4.1 Binary Logistic Regression

For this conditional model, there is assumed to be some solution similar to the linear regression problem with a linear combination of parameter estimates and known inputs $x_i\tilde{\beta}$ [114]. The difference occurs with the choice of transformation applied. Instead of a linear transformation, the sigmoid function is used, which restricts the range of output to $[0, 1]$, a binary classification [114]. The conditional probability mass function for this discrete model then becomes a Bernoulli distribution $p_{Y_i|X_i=x_i}(y_i) = \frac{1}{1+e^{-x_i\beta}}$ if the output is 1 and $p_{Y_i|X_i=x_i}(y_i) = 1 - \frac{1}{1+e^{-x_i\beta}}$ if the output is 0 (note the form of this is the sigmoid function and its complement) [110, 115]. With these assumptions, the estimate of the output is $\tilde{y_i} = S(x_i\tilde{\beta})$, which represents the probability of the true output being 1 [114].

6.4.2 Multinomial Logistic Regression

In logistic regression, the outputs for any given input to the model were binary. When this is not the case, the distribution changes from the Bernoulli to the Multinoulli distribution, as there are now more than two outcomes, yet they remain discrete and follow many of the same rules [115, 116]. This is the multiclassification problem and is solved using multinomial logistic regression. The form of the expected Multinoulli distribution generalizes the Bernoulli distribution to multiple discrete outputs, where there are K probabilities p_i such that $\sum_{i=1}^{K} p_i = 1$ [115, 116]. Now suppose a random vector X produces a vector input

$(x_1, ..., x_k)$ representative of a single sample, where all x are 0 save for 1 that indicates the output (e.g. (00001) or (10) for the binary special case) [115, 116]. Under this condition the distribution from the random vector X takes the form Equation 6.2. The conditional model based on this distribution is called softmax, and the eponymous activation function in neural network architectures works on this principle [115, 116].

$$p_X(x_1, ..., x_K) = \begin{bmatrix} \prod_{i=1}^{K} p_i^{x_i} \ for \ (x_1, ..., x_K) \\ 0 \ for \ else \end{bmatrix} \tag{6.2}$$

6.5 Likelihood Optimization

A likelihood function gives the probability of an observed outcome coming from a Probability Density Function (PDF) or statistical model as a condition of the set of model parameters [1, 117]. In order to obtain the likelihood function L from the PDF/Probability Mass Function (PMF) or model, P, the random variable samples x_n are held constant and P becomes a function of θ, $P(x_n; \theta) \rightarrow L(\theta; x_n)$ (See Subsection 6.5.1) [118].

The utility of this function is that the parameters of a model that produce its global maximum are those that create a unbiased estimator of the true parameters of the system [118–120]. This is called the *maximum likelihood method*, and is a common way to extract information about the parameters in a PDF [1, 117, 119, 121]. For several reasons the log of the likelihood function is most often used instead of the original function. For instance, the log function converts the product of independently drawn samples into a sum, which is better for stability, as products tend towards extreme values more severely than sums and thus are less reliable where numerical precision might come into play [119]. Due to the predominant use of minimization techniques in computing, the log-likelihood is often changed into the negative log likelihood by applying a negative transformation [119]. This maximum log-likelihood / minimum negative log likelihood problem is shown in Equation 6.3, where ξ is a random vector coming from the distribution characterized by parameter θ belonging to the parameter space Θ and $\hat{\theta}$ is an estimator of θ produced by solving the optimization problem on the log-likelihood function $l(\theta, \xi)$ [119].

$$\hat{\theta} = \arg \max_{\theta \in \Theta} l(\theta, \xi) \rightarrow \hat{\theta} = \arg \min_{\theta \in \Theta} [-l(\theta, \xi)] \tag{6.3}$$

6.5.1 Example - Poisson Distribution

Consider a Poisson distribution function, $P(x; \lambda) = \frac{\lambda^x e^{-\lambda}}{x!}$ and a single observation x = *10*. Following the process outlined, the likelihood function in this situation becomes $L(\lambda; x = 10) = \frac{\lambda^{10} e^{-\lambda}}{10!}$. When performing this action with multiple variables, under the assumption that all observations arise from the same distribution and are independent of each other, the joint PDF of the observations is the product of individual PDFs, *p(x; t)*, for each value of *x* via $P(x_1 ... x_n; t) = \prod_{i=1}^{n} p(x_i; t)$, where t is an arbitrary parameter and x are the observations [119, 122].

Continuing to use the Poisson system example, let there be two observations, *x* = *10* and *x* = *8*. The total likelihood function is now the product of the individual ones, which yields a new joint likelihood function of $L(x = 10, 8; \lambda) = \frac{\lambda^{18} e^{-2\lambda}}{8!10!}$. Taking the log of this returns a log-likelihood function of $l(x = 10, 8; \lambda) = -(ln(8!) + ln(10!)) + 18ln(\lambda) - 2(\lambda)$ which yields a derivative with respect to λ of $\frac{d}{d\lambda} l = \frac{18}{\lambda} - 2$. Setting this function to zero then returns the maximum likelihood estimator for the system, $\lambda = 9$. In the Poisson system, the expected value of the distribution is the parameter λ, so this result makes sense in that the mean is an estimator of that same value [122].

6.6 Predictive Models

Many ML models predict outputs using inputs that have not been observed from a given sample yet, these are called predictive models [112]. Techniques such as decision trees, random forests, and ANNs are examples of ML that can fit in this category [112]. The two major categories of predictive models are supervised and unsupervised learning, the former will briefly be discussed. In supervised learning, the data provided to the model possesses features paired to known labels/targets [111]. A supervised learning model is akin to a conditional model that finds parameters that optimize a mapping between input and output variables and then extrapolates outputs for new inputs outside of the initial sample [111].

Supervised models are divided into solving classification problems or regression problems, just like conditional ones [111].

6.6.1 Datasets

Datasets are often split into two or three blocks so that some data can be held in reserve to evaluate the performance of a model with respect to novel data. The first and largest block is the training dataset, used to train a model. The remaining data, generally a fifth or less of the original dataset, is reserved to help evaluate the model. If this data is used to determine the accuracy/quality of the final trained model it is referred to as the *testing set*, and if it is used to provide feedback on the training progress during learning it is called the *validation set*. Common divisions of datasets for these groups include 80/20, and 80/10/10 for train/test and train/validation/test respectively.

6.7 Loss Functions

The discussion has summarized how predictive models work, and now it will cover how the predictions $\tilde{y}_t$ of those models are judged in comparison to the true target values y_t [111]. The comparison of these values is called the *loss*, and the function that describes it varies from model to model, with the three most common categories being squared, absolute, or cross-entropy functions [111]. Note that in the context of ML, the terms cost, loss, and objective function are often used interchangeably [123, 124]. Cost and loss can also respectively refer to the mean value of the loss function across a dataset in contrast to the loss over a single example [124]. The performance of an ML model is not ultimately dependent on the success of minimizing the loss function, though it is a good indication of how the model may work. The way trained models are judged is by *performance metrics* such as classification accuracy.

The absolute and squared error loss functions are the simplest in form and appear respectively as Equation 6.4 and Equation 6.5 [111].

$$L_t = |y_t - \tilde{y}_t| \tag{6.4}$$

$$L_t = (y_t - \tilde{y}_t)^2 \tag{6.5}$$

These two are used for problems that predict continuous values, such as regression. For classification problems, the cross-entropy loss is commonly used, and takes the general form Equation 6.6

$$L_t = -\sum_{k=1}^{K} y_{t,k} log(\tilde{y_{t,k}}) \tag{6.6}$$

where there are K options or classes for the target variable [110, 111].

The risk R can be defined as the expected value of the loss function with respect to the joint distribution that the sample data comes from, $R = E[L_t]$ [111]. Since this distribution is not generally known, the risk is approximated as the sample mean of the loss over the data provided to the predictive model, $\tilde{R}$ [111]. The average loss function across n input data x is sometimes known as the *cost function* and has names specific to the three types discussed earlier, the Mean Squared Error (MSE), MAE, and negative average log-likelihood respectively [111]. The MAE and MSE loss functions are commonly known as L1 and L2 loss respectively. Picking the parameterization of a predictive model that produces the least average loss drives the optimization process of a predictive model [111].

6.7.1 Example - Cost Function

Suppose there is a dataset $X = x_1, x_2, x_3$ such that each x vector has the form $x_i = [x_{i1}, x_{i2}, ..., x_{in}]$. It is given as input to a predictive model which produces the outputs $Y = [\tilde{y_1}, \tilde{y_2}, \tilde{y_3}]$, where $\tilde{y_1} = 3, \tilde{y_2} = 2, \tilde{y_3} = 3$. The true values will be arbitrarily chosen as $y_1 = 4$, $y_2 = 2$, and $y_3 = 5$. Suppose as well that this model uses an L1 loss such as Equation 6.7 to generate the cost for the current set of predictions Y.

$$C(\tilde{y_i}, y_i; n) = \frac{1}{n} \sum_{i=1}^{n} L_i(\tilde{y_i}, y_i) = \frac{1}{n} \sum_{i=1}^{n} |y_i - \tilde{y_i}| \tag{6.7}$$

In this case, the cost for these examples according to Equation 6.7 will be: $\frac{1}{3}(|4-3| + |2-2| + |5-3|) = \frac{1}{3}(1+0+2) = \frac{1}{3}3 = 1$ which yields an absolute error of 1 from the cost function for this model.

6.8 Artificial Neural Networks (ANNs)

ANNs are ML models used to learn relationships present in input data through the use of iterative updates to functions that can be represented with computational graphs [109, 125]. These functions take the form of matrix multiplications and nonlinear transformations of the input data. The parameters governing the functions are called *weights*, and the nonlinear functions used to transform the data in ANNs are called *activation functions* (Section 6.9). The learning process in an ANN occurs in several distinct stages that are collectively described as *training*. As an ANN trains, input data undergoes *forward propagation*, moving through a procedural series of *layers* composed of nodes that perform functions in parallel on their input before passing the resulting data to subsequent layers, leading ultimately to a final output. The next stage in training an ANN is *backpropagation*. Here the ANN output is evaluated by a *loss function* which makes an estimate of the loss (error) of the network performance. Backpropagation refers to the fact that the contributions to the loss from the parameters in the network are calculated from the output value backwards for each layer [124]. By taking the *gradient* of the loss function with respect to the parameters of the ANN, updates are made to each of the parameters by adjusting their value to traverse in the direction that minimizes the loss function [124]. The size of the update steps is controlled by a hyperparameter called the *learning rate*.

One additional term to be familiar with in the context of neural networks is the *bias* term, b. This is a value added at each node to allow the network to learn functions that do not have to pass directly through the origin (i.e. In the case of a linear function $y = mx + b$, the bias term means that the origin $(0, 0)$ is not the default value if $x, y = 0, 0$ [123]. At the node level, the bias term is added after taking the dot product of the weights and input values, $\sum_{i=1}^{n}(w_i \cdot x_i) + b$ [126].

The update applied to the weights and biases of an ANN at the end of backpropagation is defined in Equation 6.8, where η is the learning rate, C is the cost function, and w_k and b_l are individual weights and biases respectively [124]. After updating the values of the weights, the training process repeats itself, going through forward and backpropagation for a user-defined number of training cycles, or *epochs*.

$$w'_k = w_k - \eta \frac{\partial C}{\partial w_k} \qquad b'_l = b_l - \eta \frac{\partial C}{\partial b_l} \tag{6.8}$$

The structure of an ANN can be readily described using computational graphs (Section 5.3) to illustrate some of the aforementioned processes (See Figures 6.2, 6.3, and 6.4) [108,109,123].

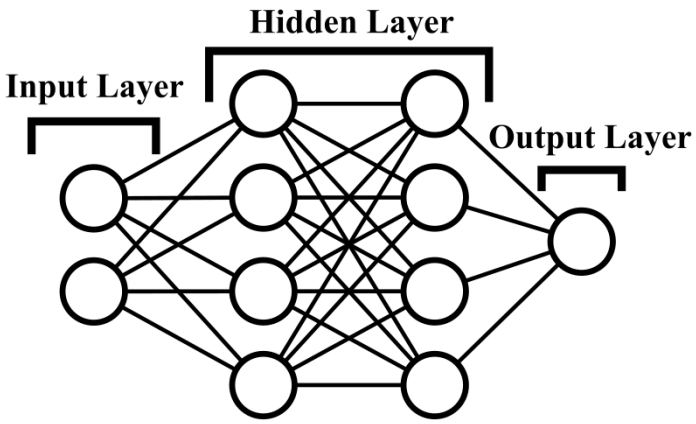


Figure 6.2: An example of a fully-connected ANN with 2 input nodes, 2 hidden layers with 4 nodes each, and a single output node. This net has a depth of 3. In an ANN, a series of randomly-initialized weights w_i is generated, one for each each of the connections between nodes in adjacent layers [123].

$$\mathbf{f}^{(i)}(\mathbf{x};\theta) = \mathbf{f}^{(i)}(\mathbf{x};\mathbf{W},\mathbf{b}) = a^{(i)}(\mathbf{W}^{(i)}\mathbf{f}^{(i-1)}(\mathbf{x};\theta) + \mathbf{b}^{(i)}) \tag{6.9}$$

The general form for forward propagation in an ANN is shown in Equation 6.9, where $\mathbf{f}^{(i)}$ is the output of the i^{th} layer of the ANN defined by parameters θ and input data $\mathbf{x}$, a^i is the activation function for the layer, $\mathbf{b^i}$ is the bias for the layer, and $\mathbf{f}^{(i-1)}$ is the output of the previous layer [127]. Thus, the output y for forward propagation of the for the three-layer network from Figure 6.5 is $y = f(f(f(\mathbf{x} \cdot \mathbf{W_1}) \cdot \mathbf{W_2}) \cdot \mathbf{W_3})$.

6.8.1 Layers

It will be helpful to provide a brief overview of the types of layers and their nomenclature. The *input layer* takes in data **x** and performs no operations on it. The next type of layer that the input is passed to is generally a *hidden layer*, which transforms **x** by multiplication with the weight matrix for that layer [123]. Hidden layers often then apply an element-wise

nonlinear *activation function* to their output to the next layer [123]. Many common networks are *fully connected nets*, which means that all nodes in a layer send their output to all nodes in the next layer [123].

The weights in a layer are randomly-initialized at the start of training a model [123]. The weights for a given layer are written as a matrix $\mathbf{W^i}$, where i indicates the layer that the weights belong to. The processing of a series of inputs through a single hidden layer in a typical ANN can be written as $y = f(\mathbf{x} \times \mathbf{W} + \mathbf{b})$, where f is an activation function, $\boldsymbol{x}$ is the input data, $\boldsymbol{W}$ is the weight matrix for the layer, and $\boldsymbol{b}$ is the bias term.

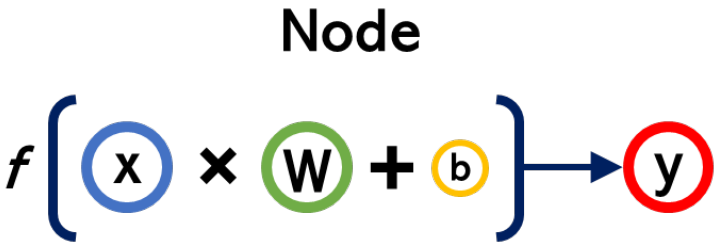


Figure 6.3: A simplified node, showing the typical process of an input matrix $\boldsymbol{x}$ being multiplied by a weight matrix $\boldsymbol{W}$ and having the node bias $\boldsymbol{b}$ added to the product. An activation function f provides an additional (optional) transform of the data [123]. Note that in this single-node example the weight matrix is a subset of the weight matrix for a full network layer.

When the input data has been transformed through the hidden layers, it will be sent through a final *output layer* which provides an estimate of the value or classification(s) that the net was constructed for [123]. For this purpose, the node(s) in this layer often apply functions that serve as classifiers or otherwise transform data into a form that is more easily interpreted to users. As mentioned previously, the output of layers in an ANN can be written as a composite function of all the previous layers, and the number of functions that form the composite defines the *depth* of an ANN [123].

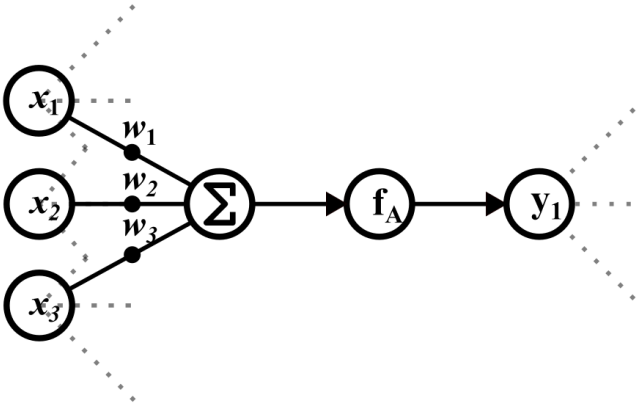


Figure 6.4: A node taking three inputs x_i and applying weights w_i to them before summing the result, which is then passed through an activation function f_A to produce output y_1.

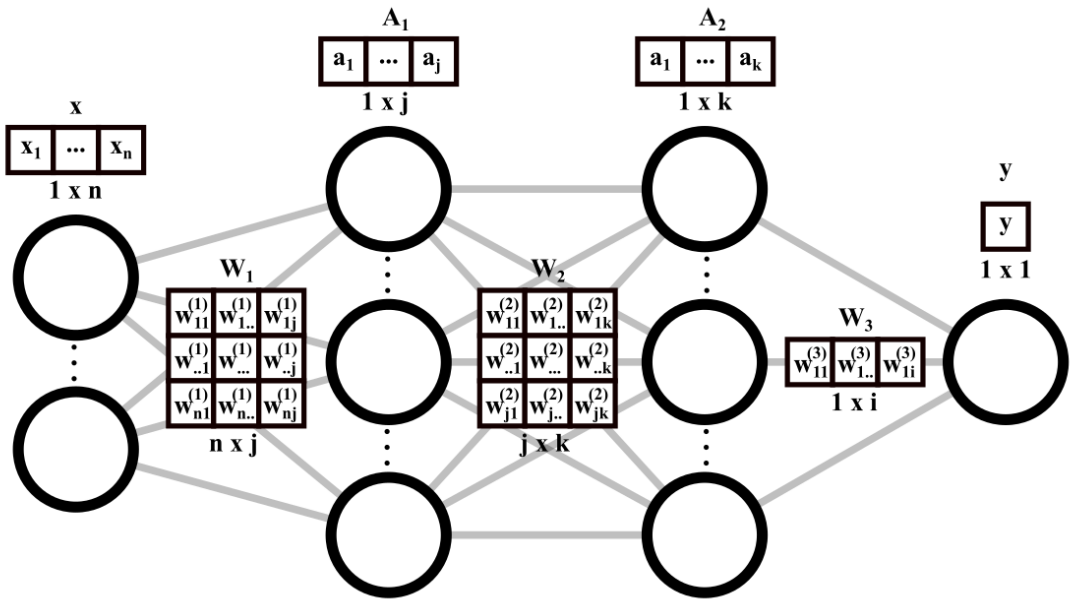


Figure 6.5: A detailed representation of a forward propagation in a typical ANN. The bias terms are omitted for simplicity.

6.9 Activation Functions

The functions applied to transform the output of the nodes in an ANN are referred to as *activation functions*. These transformations are typically nonlinear functions, and their output allows an ANN model to represent nonlinear relationships between input variables [123]. Summaries of the common functions will be given, followed by more detailed examinations of those relevant in this work and the reasons for their selection.

All figures for these functions were produced by modifying publicly available code [128].

Linear

The linear activation function (Figure 6.6) is equivalent to an identity function in an ANN, since the node activation is directly proportional to the output via the relationship $g(z) = z$. Due to the fact that this activation function cannot describe nonlinear behaviour, it is ill-suited for exploring data with nonlinear relationships. However, This activation function is commonly applied as the final activation function for ANNs built for regression tasks, as its output range is $[-\infty, \infty]$ [123].

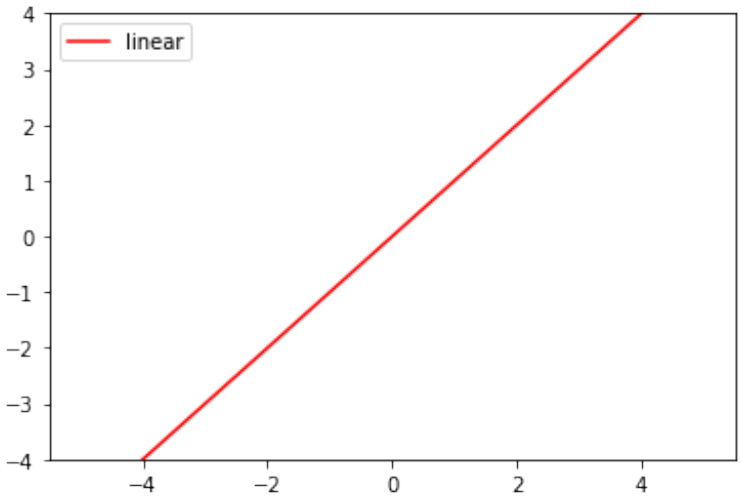


Figure 6.6: A plot of the linear activation function

ReLU

The Rectified Linear Unit (ReLU) activation function (Figure 6.7) is one of the most commonly used, and applies the transformation $g(z) = max\{0, z\}$ to the input data [123].

This function has the limitation that it does not have a continuous derivative, which could result in an error during backpropagation [123]. However this is rarely the case, as the probability of the value of the ReLU function being exactly zero is remote [123]. ReLU has another noteable limitation, which is that as soon as values provided to a ReLU activated node are negative, the output is floored to zero and the derivative of the function in the gradient likewise becomes zero. This 'kills' input from that node by omitting mappings for negative values and preventing adjustment to the node for the remainder of training [126]. This can result in poor learning and wasted neurons and is dubbed the 'Dying ReLU' problem.

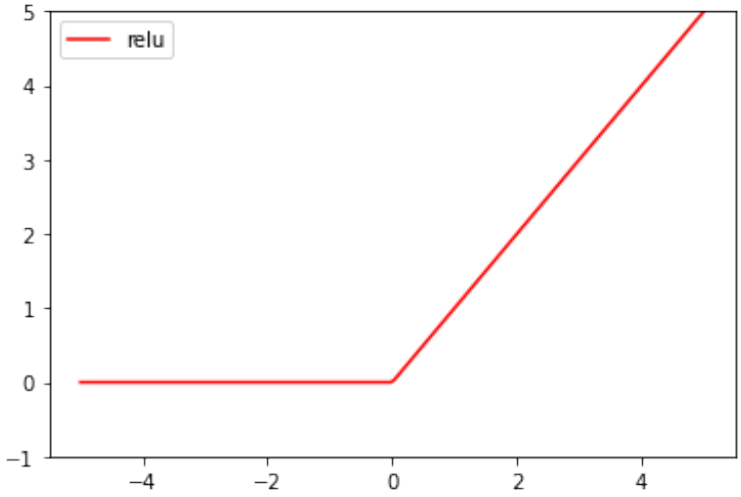


Figure 6.7: A plot of the ReLU activation function

6.10 Gradient Descent

Gradient descent is a type of *optimizer*, an algorithm used to minimize the loss function and update the parameters of an ANN [123]. To understand this in greater depth, consider the *gradient*. The gradient of some function $f(x_n)$ is a vector composed of partial derivatives across all x_n. The gradient operator is represented by the symbol ∇. When evaluated at a point $p = (x_n, x_{n-1}...x_1)$, the gradient returns the slope of $f(x_n)$ with respect to each of the x_n. In the process of backpropagation, the gradient is evaluated on the loss function during each epoch to inform the direction in which parameter values can be changed to minimize the loss. One of the requirements for backpropagation with gradient descent is that the

activation functions must be differentiable, at least for part of their domain, otherwise it would be impossible to obtain a measurement of the gradient and make informed updates to the weights and biases. ReLU for instance is not differentiable, but can be divided into a set of two differentiable domains with relative ease [123]. The negative of the gradient is used for this purpose, since the intent of gradient descent is generally to minimize the loss [126].

Example - Gradient Function

For a univariate function such as $f(x) = 2x^3$, the gradient is equal to the operation $\frac{df}{dx}$ and thus $\nabla f(x) = 6x^2$ is the gradient. The usefulness of the gradient is more apparent for multivariate situations, for instance let there exist a function $f(x, y, z) = x^2 - y^3 + 2z$. The gradient of this function is Equation 6.10.

$$\nabla f(x, y, z) = \begin{bmatrix} \frac{\partial f}{\partial x} = 2x \\ \frac{\partial f}{\partial y} = -3y^2 \\ \frac{\partial f}{\partial z} = 2 \end{bmatrix} \tag{6.10}$$

Evaluating this function at position $p = (1, 1, 1)$ yields a vector $(2, -3, 2)$ that indicates the best directions to minimize $f(x, y, z)$ are $(-, +, -)$. Therefore, the next step in seeking the minimum of this function could be the point $p = (0.5, 2, 0.5)$. A similar process occurs in neural networks with respect to the loss function.

6.10.1 Getting the Gradient in Backpropagation

In order to traverse the gradient, during backpropagation the individual contributions from each layer and weight must be obtained. Since the forward propagation during training can be represented as a composite function, this allows use of the calculus chain rule, $\frac{d}{dx}[f(g(x))] = f'(g(x)) \cdot g'(x)$, to determine the gradient with respect to all parameters in the net [123]. The overall cost function for the weights and biases can be obtained by taking the average of the partial derivatives of each example for each parameter. For instance, if the partial derivative of the cost function with respect to a particular weight w_i was

wanted, it could be written as: $\frac{\partial C}{\partial w_i} = \sum_x \frac{\partial C_x}{\partial w_i}$, where x represents the data samples used in training [124]. Via backpropagation, the network can obtain each $\frac{\partial C}{\partial w_i}$ and $\frac{\partial C}{\partial b_i}$ [124].

During backpropagation, the process for a single weight w^l_{jk} where l is the layer number, j is the current node number, a^{l-1}_k is the activation from the previous layer in the network, and k is the number of the previous node, is as follows. For a known pre-activation signal of $z^l_j = \sum_{k=1}^{m} w^l_{jk} a^{l-1}_k + b^l_j$, the partial derivative with respect to a particular weight, $\frac{\partial z^l_j}{\partial w^l_{jk}}$ can be shown to have the solution $\frac{\partial z^l_j}{\partial w^l_{jk}} a^{l-1}_k$ which is the activation of the previous layer [124]. This allows the contribution of this weight to the overall gradient, $\frac{\partial C}{\partial w^l_{jk}}$, to be written as Equation 6.11 [124].

$$\frac{\partial C}{\partial w^l_{jk}} = \frac{\partial C}{\partial z^l_j} a^{l-1}_k \tag{6.11}$$

Similarly for the backpropagation of the bias values, for a single bias b^l_j, the chain rule dictates that the value of the derivative of the cost function with respect to b^l_j is Equation 6.12 [124].

$$\frac{\partial C}{\partial b^l_j} = \frac{\partial C}{\partial z^l_j} \tag{6.12}$$

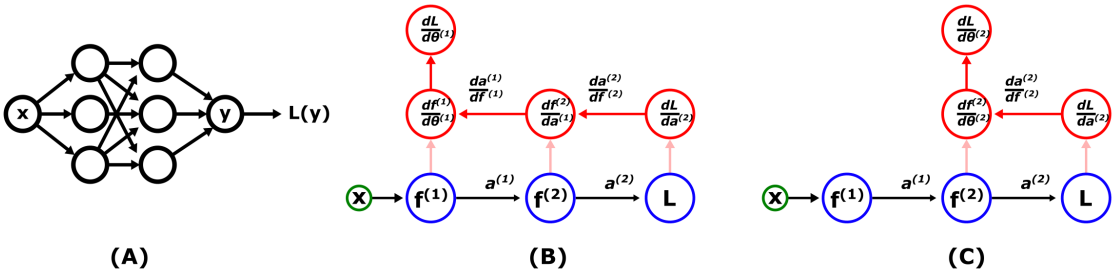


Figure 6.8: A computational graph representing (A) an arbitrary network with a depth of 2 (B) the backpropagation process that would get the gradient component for the parameters of the first layer in the network (C) the backpropagation process that would get the gradient component for the parameters of the second layer in the network

6.10.2 Learning Rate

In optimizing the value of a parameter in an ANN, the value of the learning rate determines the size of the steps taken towards the minimum of the system, which can lead to difficulty. A small learning rate can lead to extended training times as the network needs to run far longer to achieve a viable solution [123]. In scenarios with a large learning rate, the results

can be far worse, as the network may overshoot the target value and find itself diverging from it with each subsequent step, worsening the fit of the net [123]. The ideal situation is a learning rate that is between these two cases (Figure 6.9).

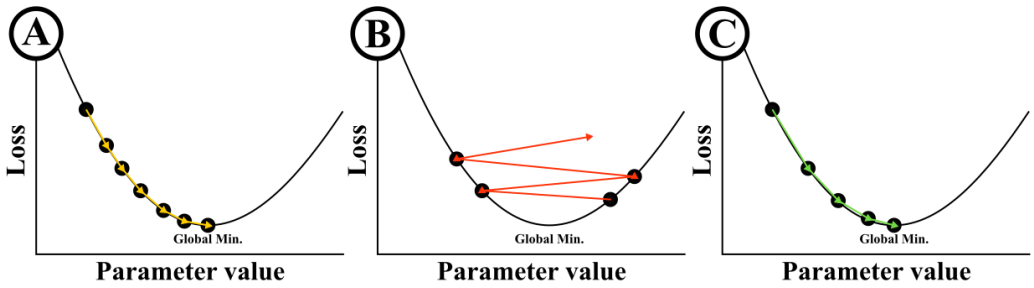


Figure 6.9: Three possible outcomes of the gradient descent process for slow, fast, and ideal learning rates, respectively. (A) Learning takes a long time and many passes (B) Learning diverges as the updates mistakenly ascend the gradient (C) An acceptable solution is achieved in a reasonable number of iterations as the network converges quickly

The general form of gradient descent is $\theta = \theta - \alpha \cdot \nabla J(\theta)$, where θ represents the parameters of the model, α is the learning rate, and $J(\theta)$ represents the cost function as a function of those parameters [124].

6.10.3 Vanishing & Exploding Gradients

When optimizing a model using gradient descent, there are several detrimental phenomena that arise from the impact of the local shape of the gradient on ANN performance [126].

When a neuron receives a set of weights that causes an activation response in a section of its range which has a very flat slope, the neuron may begin to learn too slowly, causing the network to learn poorly. When this happens, the output of this node may cause the output of subsequent layers to experience the same effect, rapidly diminishing the ability of the net to learn as the loss becomes a plateau. This is called the *vanishing gradient* problem [126].

The opposite case can also occur, when the activation output at a neuron is in a steep part of the range and experiences a large update which may accumulate to cause massive and inaccurate parameter adjustments, causing the loss to increase rapidly [126]. This is called the *exploding gradient* problem [126].

6.11 Other Optimization Functions

Gradient descent is a simple and effective way to optimize the parameters of an ANN, but there are some difficulties with larger datasets, where computing the gradient based on the entire input would result in a long calculation time. The computational cost of gradient descent is *O(m)*, where m is the number of examples used in the input dataset [126]. In addition, gradient descent may become trapped in local minima during the optimization process instead of finding the optimal solution [126].

In addition to the form of gradient descent discussed previously, which is sometimes called *deterministic* or *batch gradient descent*, there are other forms of optimizer algorithms for ANNs that govern how and when the weights of the network are updated [123]. For the most part, these are modifications to gradient descent that are intended to improve training requirements and/or model performance.

6.11.1 Stochastic Variations of Gradient Descent

Two of the most common forms of gradient descent are Stochastic Gradient Descent (SGD), which updates the weights after the calculation of the loss for each training example, and *mini batch* gradient descent, which updates after processing the loss for a subset of examples instead [123, 126]. In this way, approximating the gradient of the loss function with respect to the model parameters by using smaller sample sizes helps with memory requirements and training times, as the model does not need to store the gradient for the whole dataset, just a few examples at a time. By making more frequent updates compared to pure gradient descent, these techniques are able to achieve a faster convergence to a solution, at the cost of model accuracy [124]. The use of fewer examples at a time in gradient descent can benefit the generalization of a model, but can also slow training time down due to the number of batches that need to be processed [124].

For the pure SGD case, the optimization process is conducted using Equation 6.13 where x_i, y_i are the training input and output respectively [124].

$$\theta = \theta - \alpha \cdot \nabla J(\theta; x_i; y_i) \tag{6.13}$$

Mini-batch gradient descent is represented by Equation 6.14 where B_i is a batch of training examples [124].

$$\theta = \theta - \alpha \cdot \nabla J(\theta; B_i) \tag{6.14}$$

6.11.2 Momentum

The challenge of determining good parameter updates such that a consistent ANN solution can converge in reasonable time led to the development of *momentum*, a dynamic regulation of adjustments to the parameters informed by a decaying mean of stored gradient measurements from previous updates [126]. Momentum is represented by Equation 6.15 [123].

$$v = \gamma v + \alpha \cdot \nabla J(\theta) \tag{6.15}$$

which alters the parameter update to: $\theta = \theta - v$, where v, called *velocity*, is the moving average of previous gradients [123]. The hyperparameter γ determines how strongly the update is adjusted. Typically this value is just below 1 [123]. The result of using momentum in an optimizer is that large parameter updates are encouraged along particularly steep sections of the parameter space as 'momentum' increases. This also means that when fluctuations in parameter updates occur, the momentum term uses the previous update to adjust against the current, mitigating them and encouraging a smooth convergence (Figure 6.10) [123]. This keeps updates optimizing in the correct direction in the optimization space with a lower variance and good pacing [123]. Momentum is effective for improving learning on gradients that are small, noisy, or have high curvature [123].

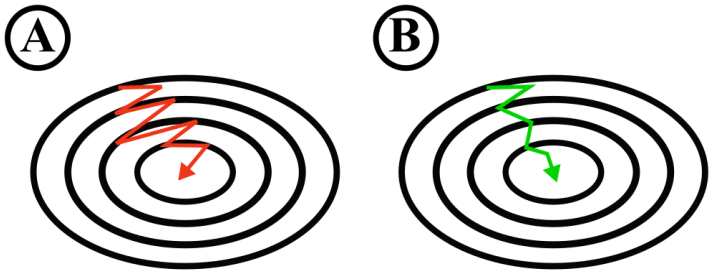


Figure 6.10: Momentum allows learning to spend less time in unfavourable directions (A) and instead smooths the variance of the gradient updates over time (B) as the network heads in the most useful direction in the optimization space

6.11.3 Adam

Of the alternate approaches to standard gradient descent, none discussed so far have altered the learning rate itself. The class of *adaptive learning rate optimization* algorithms does just this [123]. Of the more popular of these is Adaptive Moment Estimation (Adam), an evolution of SGD that uses the first and second moments of the gradient to adjust the learning rate [123, 129]. The first and second moments are the exponentially-decaying mean (as from Subsection 6.11.2), and variance of the previous gradient measurements [123, 129]. This optimization technique is robust, reduces hyperparameter tuning, and performs well on noisy/sparse gradient problems [129].

6.12 Convolutional Neural Networks (CNNs)

The previous sections have introduced the basics of neural networks and this understanding can now be used to introduce more advanced forms that pertain to this work. The first of these is the Convolutional Neural Network (CNN) [130]. In a typical FCNN, input data is given in the form of a flattened vector, a form which does not preserve the topology of the input [130]. Many data types such as photos, videos, and audio contain spatial/temporal relationships between adjacent elements. [130]. A CNN is a ML architecture that works with this information via *convolution*, a process similar to cross-correlation in signal processing [130]. The nomenclature for CNNs describes 2D input in terms of *width*, *height*. Input in 3D format is given the added characteristic *depth* [130]. Depth is often referred to by the term *channels* instead [130]. Note that for CNNs the input data must be given in a *Euclidean* form, such as a grid or cube.

6.12.1 Example - CNN Input

To familiarize the reader with the shape and characteristics of CNN input data, the case of a simple arbitrary 2D photo composed of red/green/blue (RGB) valued pixels is shown in Figure 6.11. Let the photo be of the size 9×9 pixels, where each pixel has 3 RGB values associated with it. When passing this data to a CNN, each colour is assigned its own channel, giving the network input in the form of a $9 \times 9 \times 3$ matrix.

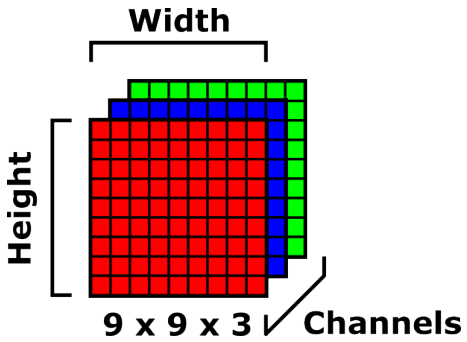


Figure 6.11: Helpful terms visually defined for a CNN.

6.12.2 Filters & Convolutions

A *filter*, often dubbed a *kernel* in the CNN context, is a matrix used to extract features from data through convolution [130, 131]. Convolutions produce larger output values where the features of the input data match those of the filter [131]. For some tasks the filters needed to best identify relationships in the input data are not known, so in a *convolutional layer*, the weights for a layer in a model are used as filter values, thus making the filter(s) trainable [131]. When a CNN trains, the filters in each convolutional layer are therefore adjusted to optimize feature recognition [131]. Note that due to the trainable nature of the weights in the filters, the process of convolution in a CNN is identical to cross-correlation in signal processing [130].

Convolution in the context of CNNs begins by taking the dot product of the kernel and a subset of the matrix representation of the input data with the same dimensions [130, 131]. This subsampling is done across the input matrix in intervals defined by a parameter called *stride* (Ex. A stride of 1 means that the input is subsampled at every possible allowed filter position with respect to the filter size) [131]. Convolution produces output with reduced dimensionality compared to the input, unless the input is *padded* with additional zero values at the margins to maintain the dimensions [130, 131]. The output size of a CNN is O, as shown in Equation 6.16, where I is the input, K is the size of the kernel, P is the extent of padding, and S is the stride [130]. Note the presence of the floor function to ensure a whole number output.

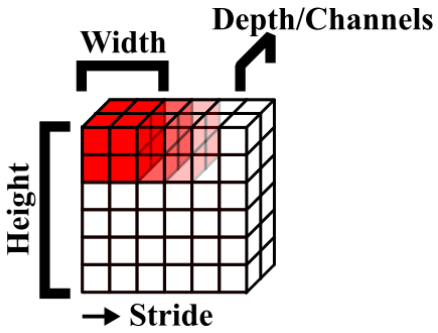


Figure 6.12: An illustration of 2D convolution on input with multiple channels. Note that for 2D convolutions, the filter kernel has a depth fully matching the depth of the input.

6.12.3 Example - Applying a Filter to a 2D Matrix

For a simplified example of convolution, construct an arbitrary $3 \times 3 \times 1$ matrix and a single $2 \times 2 \times 1$ filter (Figure 6.13). The filter is applied as a dot product between the filter and subsets of the input feature matrix defined by the stride of the filter, which for this example is 1 unit. This convolution creates an output feature map of the size 2×2.

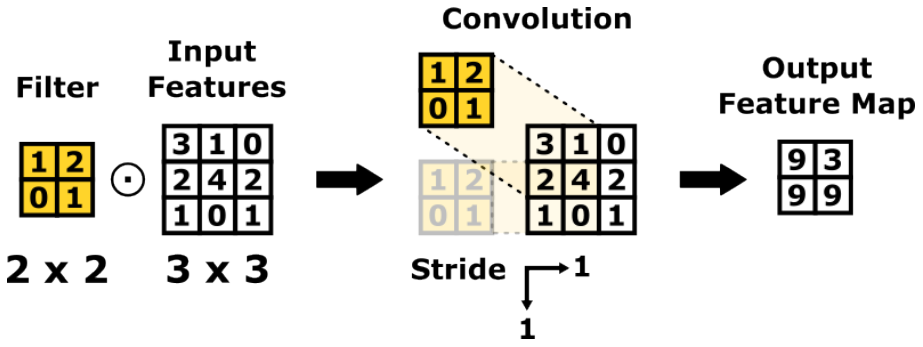


Figure 6.13: Convolution process defined for an arbitrarily constructed 2D input to a CNN.

$$O = \left\lfloor \frac{I - K + 2P}{S} \right\rfloor + 1 \tag{6.16}$$

6.12.4 Properties of Convolutional Networks

In typical FCNNs, dense layers are fully connected and each output from a previous layer contributes to the input of every node in the subsequent layer(s) [123]. One of the beneficial properties of CNNs is that convolutional layers produce *sparse weights/connectivity*, meaning that the contribution of each node in a layer only applies to a few of the subsequent

nodes [123]. This reduces the number of parameters that need to be held in the net at each step and the number of operations that need to be performed in the net [123]. Another significant property of CNNs is *parameter sharing*; unlike fully-connected layers the weights in CNNs are not singly-used, instead they are shared across the input of the convolutional layer due to the construction and usage of the kernel [123]. This also reduces the number of parameters needed in the model. In Equation 6.17 a form is given for a convolution operation over a single subset of a 2D input *x* with a filter shape *m x n* composed of trainable weights $w_{i,j}$ [127]. The output *y* is a scalar value and *b* is the bias value. The bias and filter are shared for all subsets of the data as the filter is applied to differing subsets of input according to the stride value [124]. Lastly, convolutional layers are *equivariant* to translation, meaning that if a feature in the input is shifted, the learned representation of that feature will also shift the same way [123].

$$y = b + \sum_{i=1}^{n} \sum_{j=1}^{m} x_{i,j} w_{i,j} \tag{6.17}$$

$$\mathbf{f}^{(i)}(\mathbf{X}; \mathbf{W}, \mathbf{b}) = a^{(i)}(\mathbf{W}^{(i)} \mathbf{f}^{(i-1)(\mathbf{X};\mathbf{W},\mathbf{b})} + \mathbf{b}^{(i)}) \tag{6.18}$$

A convolutional layer in a CNN is described by Equation 6.18, where **X, W, b** are the input data, trainable filter weights, and bias terms respectively (As in Section 6.8) and bold terms indicate data in matrix form. Activation functions can be applied element-wise to the output of convolutional layers [123, 131]. The number of subsequent convolutional layers in a CNN determines the level of abstraction of the feature representation in the output mapping, extracting higher-level features from the input (ex. faces, patterns, large shapes) [123]. Variants of CNN exist such as deformable convolutional nets which can recognize features in spite of geometric transformations of the input [132].

6.13 Graph Neural Networks (GNNs)

For CNNs, an implicit assumption is that there are spatial relationships between adjacent pixels. Framing the CNN in terms of graph operations, the neighbourhood $\mathcal{N}$ of elements

u surrounding element v is obtained from an input graph G with nodes $V(G)$ by finding edges (u, v) present in $E(G)$ (Equation 6.19) [38]. This information is passed through filters and aggregated to extract features from the input [38]. The GCN is a generalized form of CNN that extends this explicitly to non-euclidean input data [133].

$$\mathcal{N}(v) = u \in V(G)|(v, u) \in E(G) \tag{6.19}$$

Like CNNs, GCNs take input that contains correlations between adjacent data elements. Unlike the CNN each element is not limited to association with its nearest neighbours in a grid/matrix. Euclidean data elements are generalized into nodes and edges of a graph, allowing for greater connectivity and complexity. This format is particularly useful for modeling irregular objects as inputs, such as social networks and molecules [38]. The figures Figure 6.14 and Figure 6.15 demonstrate the similarities and differences between CNNs and GCNs respectively.

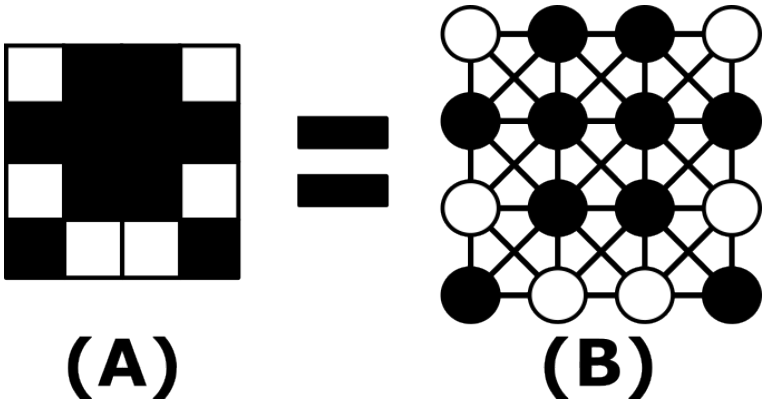


Figure 6.14: An illustration of similarities between CNNs and GCNs. In (A), an arbitrary 4×4 CNN input is shown, with binary pixel values (black & white) in a 1-channel image. In (B), a GCN input is shown that shares the same input and connectivity between nodes as the CNN in (A).

Graphs for GCN input are represented by a set of vertices and a set of edges, *G = (V, E)* just as in Chapter 5, where the total number of nodes in a graph will be denoted by *N* and the number of edges by N^e [133]. The vital properties of a graph are broken down for GCN into a handful of matrices. The adjacency matrix is represented here by $A \in \mathbb{R}^{N \times N}$ [133]. The information contained in all the nodes in a graph is represented using a *node feature matrix*, denoted $X \in \mathbb{R}^{N \times F}$, where *F* is the size of the feature vector for each node. In some

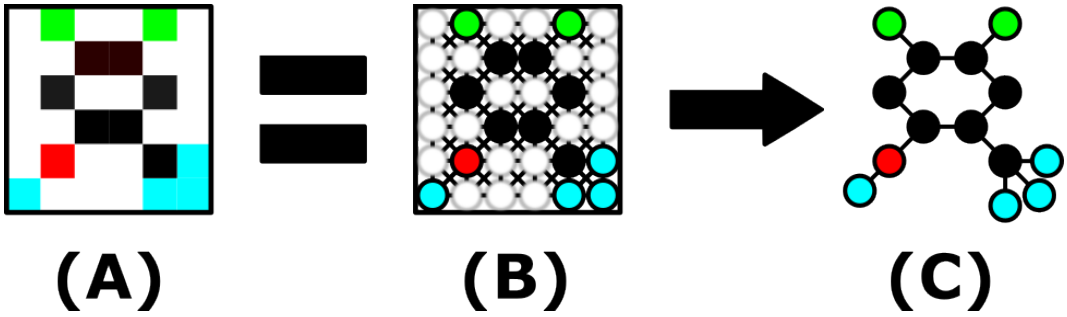


Figure 6.15: An illustration representing the differences between CNNs and GCNs using a 6×6 input with 5 different values (colours) for pixels, where white pixels are a value of 0. The CNN input (A), assumes connections between every pixel and the 8 nearest neighbours. The GCN input in (B) is equivalent to (A), with the connections between pixels/nodes made explicit with edge lines. The GCN case is not limited to rectangular input. In this example, it is assumed that the pixels assigned to a zero value (white) are merely empty and the input can instead be represented as a smaller set of nodes with a slightly different connectivity (C) without loss of information. This example was constructed arbitrarily to mimic the shape of an organic compound.

networks, edges also have their own features, and these are represented with a *edge feature matrix*, $e_{ij} \in \mathbb{R}^{N^e}$.

In a CNN there is a fixed structure and orientation for the kernel and input. This is not true of graphs, where the notion of orientation does not exist, only the nodes and their connections. In order to adapt convolution to the graph environment, a diagonal *degree matrix*, D, stores the degree (as defined in Chapter 5) for each node [134].

A useful concept for GCN is the *reference operator*, R, also called the *graph shift* or *structure* operator [135, 136]. This refers to any matrix with the same shape and sparseness as A. Adapting a filter that can operate on with varying node degree requires the use of R according to Equation 6.20 and demonstrated in Equation 6.21 and Subsection 6.13.1. The construction of a learnable filter is shown in Equation 6.20 using a reference operator R and weight matrix $\Theta \in \mathbb{R}^{F^{(i)} \times F^{(i+1)}}$, where F is the size of the feature vector for each node in layer i of the network [135]. In the case of applying a reference operator to a single node (Equation 6.21), most of the values of R are zero valued, so it is sensible to condense the notation to simply the neighbourhood set of node i, $\mathcal{N}$(i), as shown in Figure 6.16 [135, 136]

$$X^{(i+1)} = RX^{(i)}\Theta \tag{6.20}$$

$$(\mathbf{RX})_i = \sum_{j=1}^{N} r_{ij} \cdot \mathbf{x_j} = \sum_{j \in \mathcal{N}(i)} r_{ij} \cdot \mathbf{x_j} \tag{6.21}$$

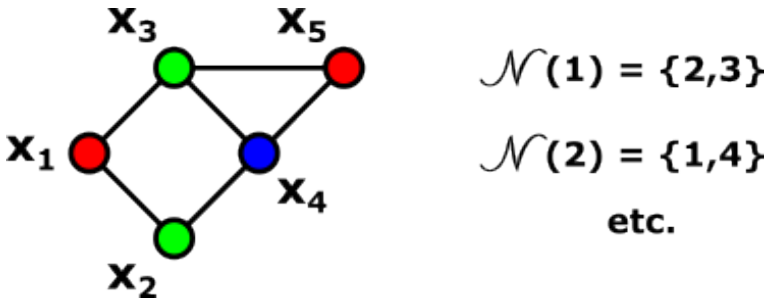


Figure 6.16: A visual representation of the nodes and edges for an arbitrary graph with several node types denoted by colours. Node neighbourhoods are indicated by calligraphic N.

6.13.1 Example - Applying the Reference Operator

As an example of the use of *R*, use the graph from Figure 6.16 and apply Equation 6.21 to it. To start, define the following matrices appropriate to the selected graph. For this graph, there are no special edge features, so the edge feature matrix has a shape $N \times 1$ and will not be used in this example.

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}, \mathbf{R} = \begin{bmatrix} 0 & r_{12} & r_{13} & 0 & 0 \\ r_{21} & 0 & 0 & r_{24} & 0 \\ r_{31} & 0 & 0 & r_{34} & 0 \\ 0 & r_{24} & r_{34} & 0 & r_{45} \\ 0 & 0 & r_{35} & r_{45} & 0 \end{bmatrix}, \mathbf{X} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \tag{6.22}$$

$$\mathbf{RX} = \begin{bmatrix} 0 & r_{12} & r_{13} & 0 & 0 \\ r_{21} & 0 & 0 & r_{24} & 0 \\ r_{31} & 0 & 0 & r_{34} & 0 \\ 0 & r_{24} & r_{34} & 0 & r_{45} \\ 0 & 0 & r_{35} & r_{45} & 0 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} r_{12} + r_{13} & 0 & 0 \\ r_{21} & 0 & r_{24} \\ r_{31} + r_{35} & 0 & 0 \\ r_{45} & r_{42} + r_{43} & 0 \\ 0 & r_{54} & 0 \end{bmatrix} \tag{6.23}$$

Multiplication of a reference operator with a node feature matrix results in a new node feature matrix that describes a graph of the same shape, i.e. $X^{i+1} \in \mathbb{R}^{N \times F}$, where the nonzero entries are weighted sums of the neighbouring nodes for each of the i nodes in G. Having demonstrated an example of the reference operator with arbitrary values, the following example shows a GCN filter operation by multiplying the previous result by an arbitrary weight matrix.

$$\mathbf{X}' = \mathbf{RX\Theta} = \begin{bmatrix} r_{12} + r_{13} & 0 & 0 \\ r_{21} & 0 & r_{24} \\ r_{31} + r_{35} & 0 & 0 \\ r_{45} & r_{42} + r_{43} & 0 \\ 0 & r_{45} & 0 \end{bmatrix} \times \begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} & w_{15} \\ w_{21} & w_{22} & w_{23} & w_{24} & w_{25} \\ w_{31} & w_{32} & w_{33} & w_{34} & w_{35} \end{bmatrix} = \begin{bmatrix} (r_{12} + r_{13})\mathbf{w_1} \\ r_{21}\mathbf{w_1} + r_{24}\mathbf{w_3} \\ (r_{31} + r_{35})\mathbf{w_1} \\ r_{45}\mathbf{w_1} + (r_{42} + r_{43})\mathbf{w_2} \\ r_{54}\mathbf{w_2} \end{bmatrix} \tag{6.24}$$

Each weight applies to a neighbourhood of a specific node, since it acts on the dot product of the reference operator and the feature matrix corresponding to node i. Multiple convolutional layers in a GCN applied to the same graph function similarly to successive layers in a CNN, capturing larger-scale interactions in the input data and pulling in weighted features from neighbours of neighbours and so on. The common GCN layer (Equation 6.26) is constructed with a reference operator based on Equation 6.25, where the identity matrix $\boldsymbol{I}$ is added to the adjacency matrix $\boldsymbol{A}$ Equation 6.25 [134]. This accounts for the 0th degree convolution via allowing nodes to have their own existing features taken into account via *self loops* in the graph [134].

$$\mathbf{R} = \mathbf{D}^{-1/2}(\mathbf{I} + \mathbf{A})\mathbf{D}^{-1/2} \tag{6.25}$$

$$\mathbf{X}' = \mathbf{D}^{-1/2}(\mathbf{I} + \mathbf{A})\mathbf{D}^{-1/2}\mathbf{X}\mathbf{\Theta} \tag{6.26}$$

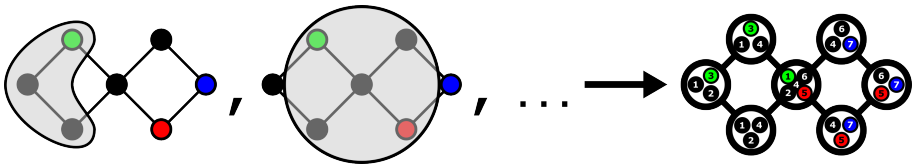


Figure 6.17: An example of a single convolution on an arbitrary graph. Each node's immediate neighbourhood is captured (shaded grey area) and the values of the node features are aggregated (right side) and weighted using trainable weights to create a graph with the same adjacency

6.13.2 Graph Network Tasks

One of the qualities that distinguishes between graph-based networks is the feature type of the graph that the output is based on Figure 6.18.

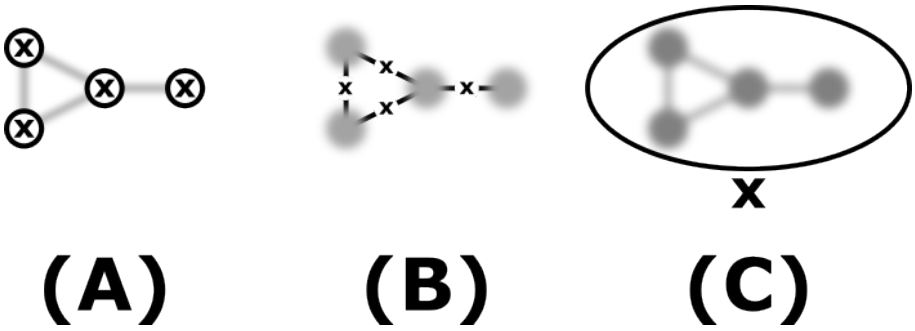


Figure 6.18: An overview of common tasks using GCNs. Since graphs are composed of separate parts in the form of nodes and edges, the values of node (A) and edge features (B) can both be predicted in regression or classification tasks using GCNs, as well as whole graphs (C) [133].

6.14 One-Hot Encoding

One-hot encoding is a popular feature representation technique applied to categorical data to convert it to numerical data in cases where there are relatively few categories to represent [137]. One-hot encoding prevents an ANN model from incorrectly determining an ordinal relationship exists between the categorical variables in the input data (e.g. to

prevent the model from interpreting atoms 1 and 2 as being more similar than atoms 1 and 5). In order to understand one-hot encoding better, consider the concept of *dummy* variables. In regression problems, these are variables D that take binary integer values $[0, 1]$ and represent categorical features of the data.

For example, a matrix of dummy variables could represent the colour of a stop light at an intersection as cars pass through.

6.14.1 Example - One-Hot Encoding

At a stoplight, 4 cars pass through during an arbitrary timespan. Using dummy variables, represent which colour the light was at a given time in row format. columns 1, 2, 3 represent categories red, yellow, and green respectively. $\mathbf{X} = \begin{bmatrix} 0\ 0\ 1 \\ 0\ 1\ 0 \\ 0\ 0\ 1 \\ 1\ 0\ 0 \end{bmatrix}$

Chapter 7

Classification & Feature Extraction of Molecular Dynamics Interstitial Defect Sites in High Entropy Alloys via Subgraph Isomorphism

Christoff Reimer, Peyman Saidi, Conrard Tetsassi, Stephen Whitelam, Mark Daymond, Laurent Beland, Isaac Tamblyn

7.1 Abstract

In this work, MD, LAMMPS, OVITO, and subgraph isomorphism were combined for screening the diffusion of self-interstitial point defects through a HEA FCC crystal structure. This method has the potential to create a training dataset for use with GNNs for rapid classification of MD trajectories, or to be used as-is to catalogue the state of a defect in any given frame of an input trajectory.

7.2 Introduction

Frenkel pairs, consisting of interstitials and vacancies, are the initial products of irradiation in crystalline materials. Their evolution to form extended defects such as dislocation loops or to promote swelling, segregation, and creep culminates in severe degradation of material properties [25, 31, 83, 138]. Due to the lower migration energy of self-interstitials compared to vacancies, and consequently their higher mobility, they are more prone to be removed at sinks such as grain boundaries [74, 84]. Self-interstitial migration towards

existing dislocations produces creep by extending the dislocation along the direction of material load [139]; their diffusion also causes radiation-induced mixing and mass-transport, leading to solute segregation and precipitation in alloys which affect corrosion resistance and hardness [138, 139]; and their presence/mobility plays a role in the rate of void swelling by controlling the rate of recombination with vacancies in different temperature/neutron flux regimes [74, 84].

Self-interstitials induced by irradiation can take on various shapes over time depending on local compositions and temperatures, including dumbbells, tetrahedra, octahedra, and crowdions. A common form is the dumbbell split-interstitial form, where two atoms share one lattice site [61]. Due to the anisotropy of this configuration, it responds to applied external stress placed on its host lattice [68]. As the dumbbell diffuses through the lattice, it may change compositions, exchanging one of its pair of atoms for another during a transition, often rotating in the process, swapping the relative positions and orientation of the pair (Figure 7.1) [61, 138]. Dumbbell migration barriers have also been shown to vary in alloys compared to their values in a pure system [138]. Movement of the dumbbell defect can also proceed by crowdion diffusion from a canted dumbbell (at least in BCC metals) [140].

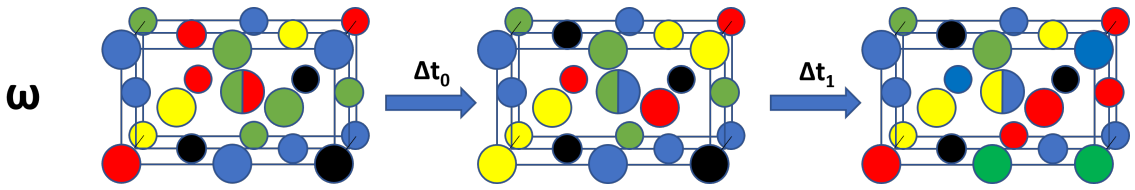


Figure 7.1: Illustration of an arbitrary dumbbell trajectory in a FCC HEA, where ω represents the whole series of states in the trajectory and each Δt_i represents a residence/escape time for a given state i (states 0 and 1 in this example). The circle split into two colours in the centre represents a dumbbell pair with a colour and orientation corresponding to the type/position of the involved atoms. The nearby atoms of the unit cells along the dumbbell bond axis are also shown within traced cells.

From the many local environments present in the lattice HEAs exhibit very complex diffusion dynamics and a wide range of migration rates for point defects [16, 97]. This is often given as a reason for the observation of interesting properties such as irradiation resistance seen in some HEAs, a subject of ongoing critique and research [16, 73]. Regardless, the rich migration energy landscape provided by the HEA environment poses a useful challenge for

predictive models of defect migration and served as motivation for this project.

Due to the compositional variation of the self-interstitial defect, the preference for the dumbbell defect form, and the popularity of studying these defects using MD; techniques for the extraction, labelling, and prediction of defects from MD trajectories is of use in materials science. The use of ML models such as GCN was considered for this task; however in light of the significant complexity of the dumbbell defect the task of classifying and extracting point defect states in MD trajectories proved to be its own work, with applications in studying defect evolution over time as well as the contribution of different local compositions/configurations to migration rates. A "fingerprint" technique was ultimately developed and applied to the data, distinguished from the recent FaVAD automated defect fingerprinting technique by using graph fingerprinting instead of vector fingerprints [141].

7.3 Methods

In this work the simulated HEA cell was of a classic ideal type, exhibiting a solid solution in FCC phase and composed of equiatomic fractions. A single interstitial defect was created by placing an additional atom into the system. The system was then allowed to evolve at 900 K, with PBCs. The interatomic potential used for MD came from a model developed by Farkas et al, built for a Fe-Ni-Cr-Co-Cu HEA using an Embedded Atom Method (EAM) using individual atom potentials that were fit until they produced a stable FCC solid solution structure [33]. Simulations were performed using the LAMMPS MD package and all visualizations or modifications of the simulation files were made using the OVITO Python package.

Feature extraction of the configuration of the self-interstitial was performed using a combination of graph comparison and the OVITO Python interface. First, all FCC atoms in the system were removed via Common Neighbour Analysis (CNA), with the aim that the remainder represented the site of greatest distortion of the crystal structure and was thus the region of interest. Then, the connectivity of the remaining atoms was determined using a threshold distance of 2.5 Angstroms to decide if atoms were neighbours. This distance was decided as it allowed a margin for highly distorted systems to be captured. A graph

representation of the non-FCC atoms that had a nonzero number of neighbours was then constructed using NX.

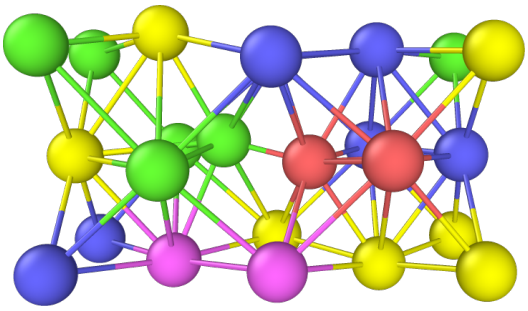


Figure 7.2: A dumbbell defect and surrounding environment in an FCC lattice, extracted from MD simulations with OVITO, containing only atoms within the two unit cells along the dumbbell bond axis.

The shortened bond distance of the split-site made it easier to track the pair in the split site, as in nearly all dumbbell states the shortest bond distance found was that of the dumbbell pair (Figure 7.2). However, even after using OVITO to remove all FCC atoms in each MD step via the CNA modifier, the distortion in the lattice caused by the presence of the defect left a few atoms behind that prevented a clean extraction of the dumbbell or other defect. To eliminate erroneous atoms captured as part of the distorted system, reduced and idealized graph representations of point defect structures were constructed for the forms expected in the MD trajectories (Figure 7.3). For instance, the dumbbell fingerprint (Figure 7.3 (D)) was constructed from two atoms joined by a single bond, where each atom is nearest to four atoms of the adjoining FCC unit cells disrupted by the defect. These 'fingerprint' graphs were then used to classify the MD states by subgraph isomorphism in order to extract the dumbbell defect frames. Subgraphs for comparison with the fingerprints were obtained via algorithm 1.

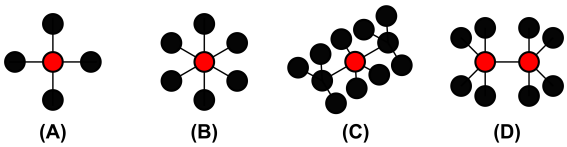


Figure 7.3: Example graph 'fingerprints' for (A) Tetrahedral (B) Octahedral (C) Saddle (D) Dumbbell sites. These were matched against subgraphs decomposed from non-FCC atoms present in a MD-simulated HEA.

The larger defect fingerprints such as the saddle and dumbbell all contain subgraphs isomorphic with smaller graphs such as the tetrahedral fingerprint, even without extra

atoms involved. In order to prevent this and similar problems from interfering with the classification, the comparison of different subgraphs was organized first by separating the classification tree into two branches based on the number of nodes present in graph representations of the MD defect states. Then the largest subgraphs of a compatible size were checked for isomorphism with each relevant fingerprint in decreasing graph order (See algorithm 2 and Figure 7.4 for more details). Due to overlapping subgraphs between frames containing dumbbell and crowdion defects, geometric considerations were added to fine-tune the final classifications, as shown in Figure 7.4 (E).

Algorithm 1 Decompose *G(V,E)* into all *G[S]*

for i in range(1, $numAtoms$+1) **do**
  $subgraphDict$[i] = []
for n in range(1, $numAtoms$+1) **do**
  for $subnodes$ in (x for x in itertools.combinations(G, n)) **do**
    if nx.isconnected(G.subgraph($subnodes$) **then**
      $subgraphDict$[n].append($subnodes$)
Remove repeats
return $subgraphDict$

Algorithm 2 Given a graph of a distorted lattice, *G(V,E)* and a set of point defect graph fingerprints *D*, find the first instance of $G[S] \simeq F(V, E)$ under the following conditions

Require: $S \subseteq V(G)$
for *G[S]* $\in G$ **do** *G[S]* are organized in a dict keyed by decreasing order
  Use $B \subseteq D$ to match expected order of defect graph with fewer comparisons
  for $F(V, E) \in B$ **do** $F(V, E)$ are sorted in D by decreasing order
    if $|S(G)| >= |V(F)|$ **then**
      if $S(G) \simeq F(V)$ **then**
        Apply additional checks to accept/reject a match (ex: bond angle)
        return dict key "**Type**" of $F(V, E)$
return **None**

After classification of MD states was complete, a sample was taken of each unique dumbbell state in order to build up a dataset of dumbbell transitions. In order to extract the isolated dumbbell defects with their neighbours along the 2 unit cells aligned with the dumbbell axis, algorithm 3) was used. In summary, the edges of a box defined from the centre of the dumbbell bond were extended by $\pm 5/4a$, $\pm 3/4a$ on the long axis and short axes respectively, as determined by the alignment of the dumbbell bond with the principle axes of the system.

Algorithm 3 Cut out a ROI of 2 unit cells along the system axis parallel to the dumbbell bond, centred on the dumbbell centre of mass or origin point identified by the variable *ori*

Require: $ori \in [0, 1, 2], bounds = [[5/4, 3/4, 3/4], [3/4, 5/4, 3/4], [3/4, 3/4, 5/4]]$

```
x, y, z = bounds[ori]
for all c, p in enumerate(system_particle_coords) do
    if −x < p[0] < x and −y < p[1] < y and −z < p[2] < z then
        box.append(c)
OVITO remove all atoms not in box
OVITO cluster analysis –> create Topology
return Topology
```

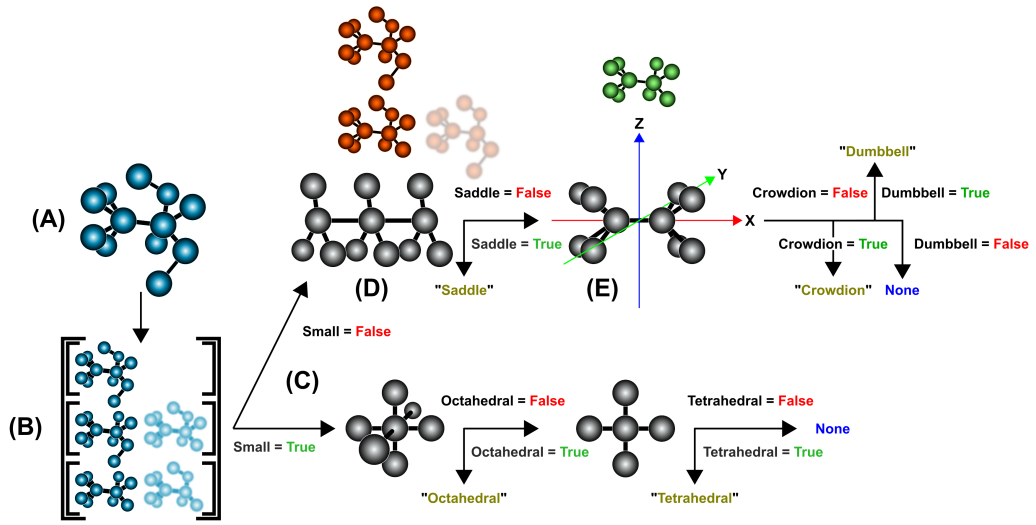


Figure 7.4: Illustrated companion to algorithm 2. Fingerprint graphs from Figure 7.3 are drawn in grey. The process shown is for an arbitrary dumbbell defect aligned with the x-axis. In (A) the defect (blue) is pulled from its surroundings in OVITO. In (B) the defect is decomposed into a list of subgraph representations, organized by order (number of nodes), with five examples shown. In (C) the list of subgraphs is sent through the first branch of the classification, which compares the order of the defect from (A) with a threshold value. If the order is above the threshold it is checked against the fingerprints for the larger defect states, otherwise the list of subgraphs is considered part of a small defect. In (D) the list of subgraphs from (B) is checked against the first fingerprint on the large defect branch, the saddle fingerprint from Figure 7.3, by decreasing graph order. The list in the example has only one member of the same order as the fingerprint, the parent graph from (A), which fails the isomorphism test (top subgraph in red). All remaining subgraphs in the list are automatically disqualified due to insufficient order (two examples shown in red). In (E) the dumbbell fingerprint is checked against the subgraph list. This comparison is performed similarly to the first, with added geometric constraints due to overlap in the graph structure of the crowdion and dumbbell defect representations. If there is a subgraph isomorphism match for the dumbbell defect (shown in green), the alignment of the central bond is compared with the system axes, returning a classification of "crowdion" if diagonal with respect to the axes and a classification of "dumbbell" otherwise. If there is no match between the graph fingerprint and any subgraph from (B), a non-value is returned and recorded as an unknown state.

7.4 Results

The potential energy of the HEA system was recorded at each step of the trajectory and plotted in Figure 7.5, showing an increase in energy after the initial timestep and then a gradual decrease. This behaviour is attributed to atomic-scale strain imposed on the system by the initial placement of the interstitial and the subsequent relaxation of the lattice over time. However, as the energy was recorded after relaxing each step of the initial MD trajectory, an observation of longer trajectories is necessary to confirm this.

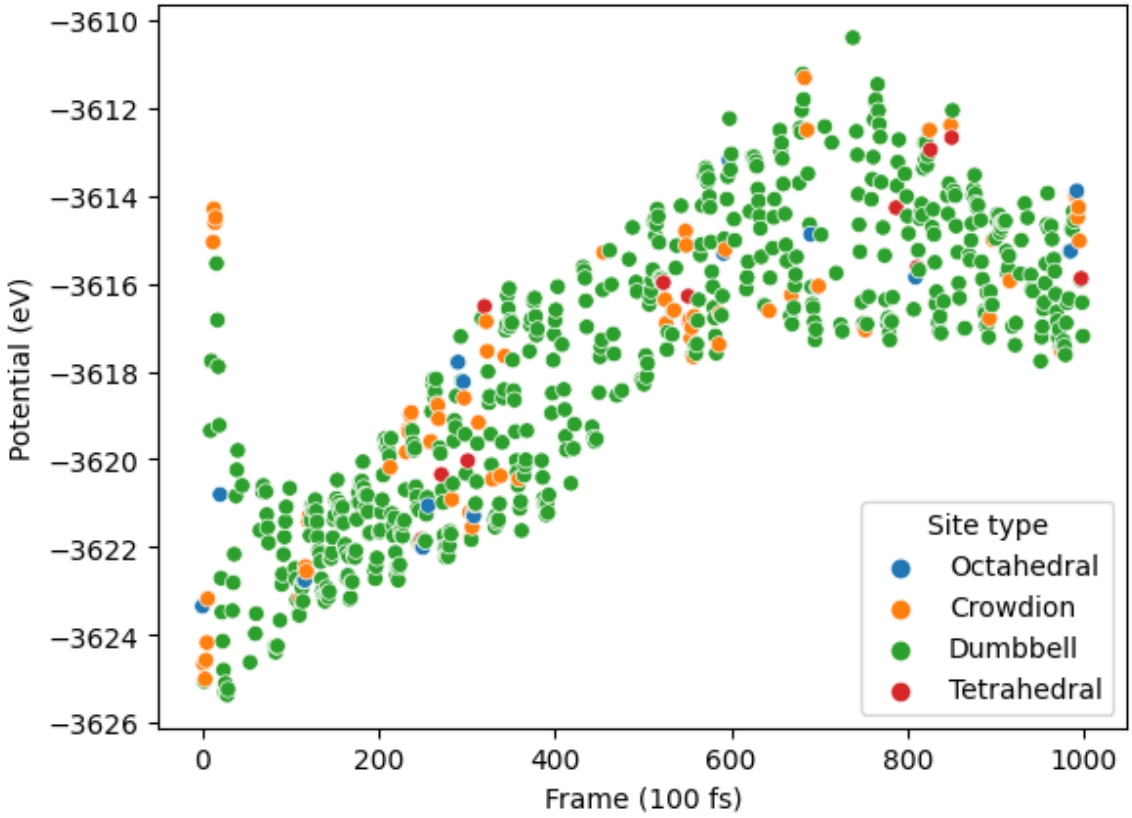


Figure 7.5: Potential energy of the MD HEA system (in eV) as a function of time (in 100 fs/frame steps), colour-coded by classifications assigned via algorithm 1 from the fingerprints in Figure 7.3

In keeping with the reported behaviour of FCC lattices, the majority of defect states catalogued were dumbbells, at just over half (577) of the 1000 frames used in this trial. Few of the less stable octahedral and tetrahedral defect states were noted in the trajectory, and many states were classified as saddle points. The abundance of saddle point classifications is likely due to the low migration energy and high mobility of the dumbbell defect offering

ample instances where incomplete transitions could be captured by MD [60]. Molecular dynamics data indicated that crowdions typically caused the most significant atom displacement, matching with expectations [19]. This was used to help classify them separately from dumbbells, as MD states identified manually as crowdions generally exhibited a larger set of atoms collected for algorithm 2 after removing the non-distorted FCC atoms from the test cell.

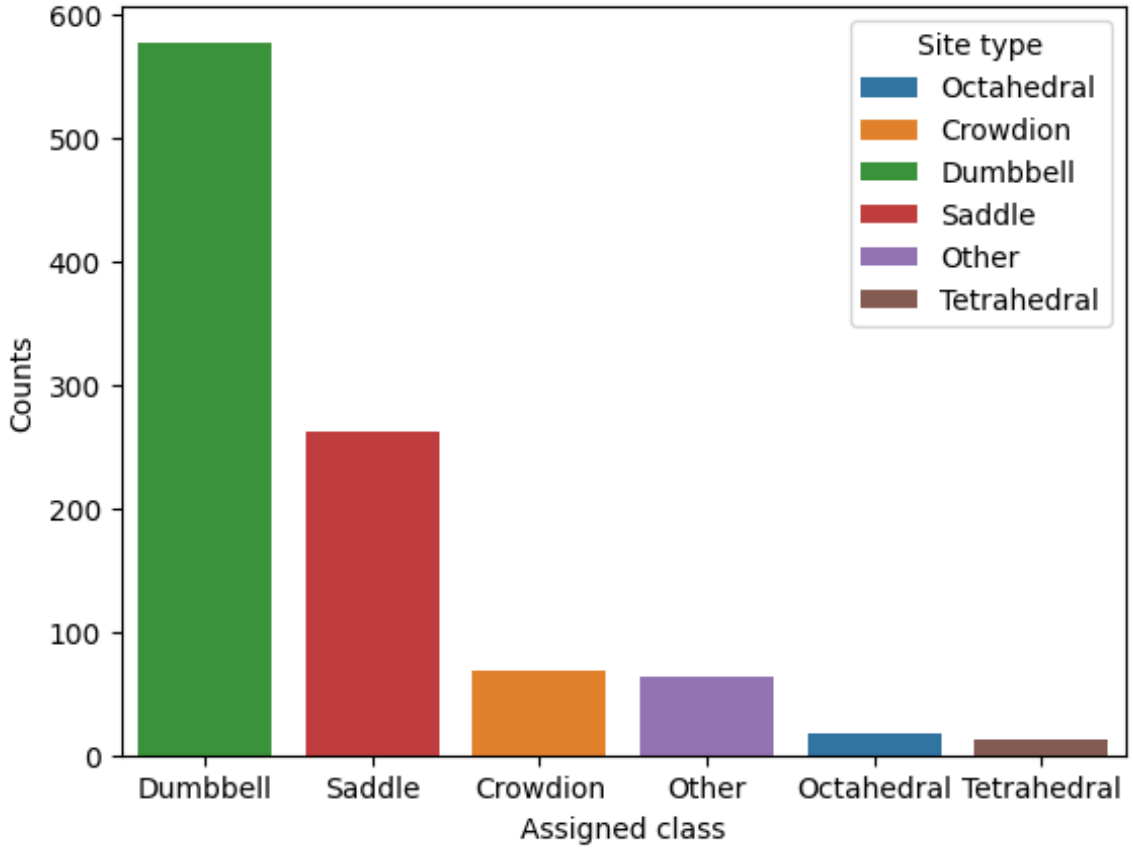


Figure 7.6: Each state of the MD system was classified into one of several categories based on the expected forms (dumbbell, saddle, crowdion, octahedral, and tetrahedral), plus an additional category for results that did not match any of the graph fingerprints from Figure 7.3. These unknown cases were probed and appeared to correspond to exceptionally distorted saddle and crowdion state transitions.

In the analysis of 1000 MD frames, 577 frames were classified as dumbbell form self-interstitials by algorithm 1 and boxed by algorithm 3. Of these, 76 continuous dumbbell-dumbbell transitions and 54 unique pair compositions were detected. Further examination of the data is required to determine what fraction of the remaining transitions belonged to other diffusion mechanisms, such as the canted dumbbell-crowdion diffusion reported in

BCC structures [140].

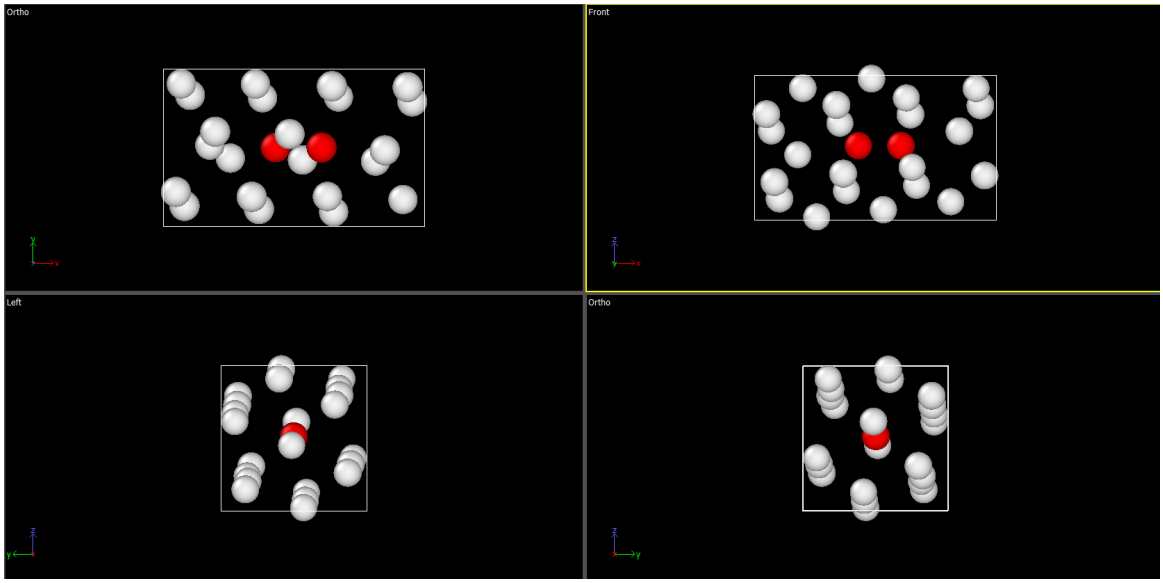


Figure 7.7: A visualization in OVITO of the boxing of a single dumbbell defect and surroundings, with all other atoms in the system deleted. The dumbbell pair at the split interstitial site are marked in red

After reaching a satisfactory level of classification accuracy with the use of graphs on the MD data, and having the capacity to box and extract dumbbell defects, the use of graph-formatted dumbbell defects as training data for ANNs such as Transformers or convolutional nets would not proceed and instead the lessons learned from this work would go towards studying the vacancy defect, the simpler counterpart to the dumbbell defect. The main next development for this project is likely to involve the use of identified defect graphs as the training dataset in a GCN classification model, which is anticipated to accelerate analysis of defects from MD data. Aside from streamlining the algorithms in this work, improving the orientation-detection and box-cutting abilities of algorithm 3 would be another recommended next step to allow for cutting defects out of systems not aligned directly with the principle axes.

7.5 Conclusion

This work demonstrates the practical ability to classify point defects of a variety of types in MD trajectories by the use of subgraph isomorphism, providing a means of studying the formation, migration, and change of point defects during diffusion through the FCC crystal.

Ultimately, due to complicated dynamics resulting from the anisotropy of the dumbbell interstitial [68] as well as its interactions with neighbouring atoms at differing distances from the defect centre, attention was diverted to instead examine the movement of the vacancy half of the Frenkel pair defect and its movement characteristics. Future directions and additions to this work could include: training a GCN classification model by feeding extracted graphs of non-lattice conforming atoms using true labels based on the results of subgraph fingerprinting, speeding up the efficiency of the process; adding a graph of the vacancy environment to the Python/OVITO fingerprinting tree to encompass all the common point defects for generalization; extending the code to handle systems containing multiple point defects by way of OVITO's cluster analysis modifier; and adding a check for the dumbbell shortest bond to support/confirm the results of dumbbell classifications. Any or all of these would elevate and generalize the usefulness of this work to research involving point defects and MD in FCC systems, both in HEAs and in pure crystals.

Chapter 8

Prediction of Vacancy Defect Diffusion Paths in High Entropy Alloys via Machine Learning on Molecular Dynamics Data

Christoff Reimer, Peyman Saidi, Conrard Tetsassi, Corneel Casert, Chris Beeler, Stephen Whitelam, Mark Daymond, Laurent Beland, Isaac Tamblyn

8.1 Abstract

In this work, GCN, MD, and KMC techniques were drawn from to create a proxy model for predicting the diffusion of vacancy defects through a HEA FCC crystal structure. This model can generate synthetic trajectories by applying predictions to graph versions of the system and saving the updated results in MD file formats. The feasibility of this approach was initially tested using a toy system built with abstracted rules for vacancy defect migration based on the interactions of nearest-neighbours of the vacancy surface, in increasing complexity. In order to test this model on a more realistic system, a ML interatomic potential for a Fe-Ni-Cr-Co-Cu HEA was sourced and applied in LAMMPS to generate datasets for use in training and testing. Techniques developed in previous work with the dumbbell defect form were adapted to effectively cull topological and compositional data from the dataset and convert it to a graph format. To check the accuracy of the learned dynamics of the model on the MD data, work is ongoing to examine the diffusion behaviour of the synthetic trajectories compared to the genuine trajectories, in lieu of missing empiric transition rates.

8.2 Introduction

Future structural materials in nuclear power plants are desired that are more resistant to the effects of irradiation-induced defects and produce less harmful waste products than current alloys, prompting a search of materials design spaces for suitable alternatives, including HEAs [89, 91]. This type of search is often aided by ML models [142], and there is an awareness in literature of the potential of ML to reduce the use of techniques such as KMC and increase the speed at which atomic behaviour can be modelled and predicted [9, 11]. For instance, incited by the cost-limited application of DFT to only a few hundred/thousand atoms, DFT-GCN models have been trained that can produce interatomic potentials of nearly equal quality for properties such as vacancy formation energy in Fe [8]. Multiple ML techniques, including SVM, Random Forest (RF), KNN, and ANN have been used in conjunction with KMC to predict growth of thin films in order to bypass the difficulty of running thousands of KMC calculations to populate a dataset to model film growth [81]. A hybrid MD and KMC technique has also been applied to a HEA to probe temperature-dependent chemical ordering [143].

The movement of defects within the crystal system has previously been explored using LAMMPS and OVITO, studying how annealing might change the microscopic stresses imposed on a pure metallic crystal system [19]. This work aimed instead to use GCNs on LAMMPS and OVITO data in order to obtain a model with the capacity to predict point defect movement dynamics that would previously have been performed using only KMC and/or MD (Figure 8.2). The configurational entropy of HEAs makes analysis of overall properties difficult, and taking inspiration from other HEA local feature representations, this work opted for an approach using a simple extracted local environment to train a ML model to predict properties of the HEA system [3]. Instead of predicting site-specific properties of the surface [3], global properties of crystals [5], or interatomic potentials [8], this work predicts escape rates from a vacancy defect for each atom in a local environment (See Figure 8.1).

Without empiric rates to compare against for a true MD simulation, an alternative instead of generating migration energy barriers (as recent work has done [37]) was required in

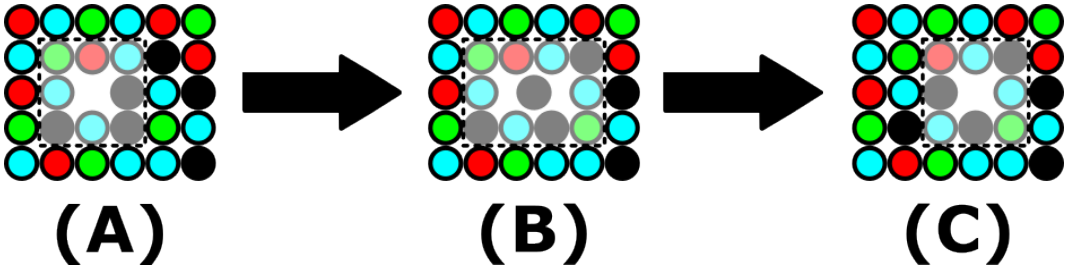


Figure 8.1: An example of a vacancy defect transition in an arbitrary 2D HEA crystal system: A) The initial state of the vacancy (shaded region) and its neighbourhood of atoms. (B) An atom from the immediate neighbourhood of the vacancy moves in to fill the vacancy during a transition (saddle point configuration). (C) The final vacancy state and neighbourhood, after the successful transition.

order to assess model quality. It has previously been demonstrated that a single trajectory of states was sufficient to train a ML model that displayed good prediction of dynamics for stochastic 3D binary spin systems and was able to extrapolate additional traits of the system implicitly [1]. Furthermore, diffusion techniques such as Root Mean Square Fluctuation (RMSF) have been previously used to determine the similarity between system dynamics in organic molecules [144] and to observe point defect migration in crystals [10]. Combining the dynamics learning approach from Casert et al to train a GCN model with analysis of diffusion behaviour is the approach implemented in this work.

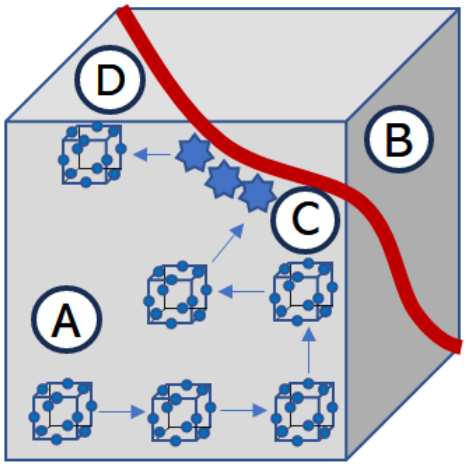


Figure 8.2: An illustration of a proposed application of this model and algorithm in defect diffusion simulations. A) A vacancy defect, migrated through a simulated system according to ML learned dynamics, B) A grain boundary close to the migrating defect, C) A transition from KMC-like transitions back to MD for the duration of the defect's residence at the grain boundary, D) A resumed trajectory using the learned dynamics

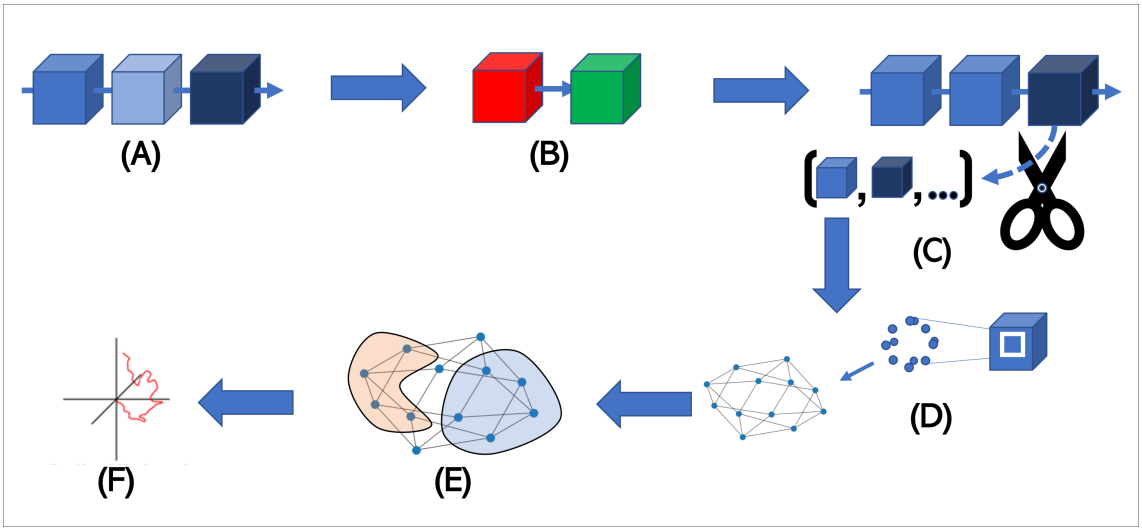


Figure 8.3: An illustration of the workflow involved in this project. (A) A trajectory composed of many states is obtained by evolving a system with MD. (B) The trajectory data is minimized to reduce noise and make analysis easier. (C) Analysis of the trajectory dictates where to cut out example frames for the state of the system. (D) Vacancy defects are isolated from their representative cutout frames and converted into ML-compatible formats. (E) The graphed defects are used to train a GCN which looks at how each atom is influenced by its neighbours. (F) The trained GCN is used to generate trajectories with its learned dynamics, and the result is compared with the dynamics governing MD trajectories of the system.

8.3 Methods

In order to test this concept, a few assumptions were made regarding the interaction of the defect with its surroundings. The first of these is that the crystal environment would be perfect, or nearly so, aside from the presence of the vacancy defect. The next was that only the 1st and 2nd neighbours surrounding the vacancy would have a significant impact on the diffusion path [25], with just the 1st neighbours being capable of filling the vacancy. It was also assumed that there was only one defect in the system at a time for the purposes of data generation and training. However, the code implemented from the learned dynamics was made with a limited capacity to generate multiple vacancy trajectories simultaneously in the same system. A toy proof-of-concept model was built to provide a simpler test case in addition to the model trained on MD data. To assess the viability of learning from interactions with different groups of nearby atoms, the toy model proof-of-concept was tested on toy data generated according to three models: first vacancy nearest neighbours without interaction (0-NNI), first vacancy nearest neighbours with interaction (1-NNI),

and first & second vacancy nearest neighbours with interaction (2-NNI), described in Subsection 8.3.1.

8.3.1 Data Generation

The bulk of the data used in this work was generated using LAMMPS and post-processed with the OVITO Python package. LAMMPS is an open-source parallel implementation of classical MD that was used for all MD in this work [71]. The nature of the equations that govern MD simulations (see figure for eq 13.3) is such that they are easily parallelized using MPI and the workload of MD distributed over multiple cores, up to millions in principle [71]. LAMMPS operates in either 2D or 3D space using orthogonal or triclinic bases and a variety of boundary conditions including fixed, shrink-wrapped, and PBCs, depending on user preference [71]. LAMMPS parallelization breaks the total MD simulated domain into unique and non-overlapping subdomains that each have roughly the same number of atoms (if the domain has reasonably consistent density) which keep track of both their own atoms as well as those within a cutoff distance in neighbouring subdomains [71]. Keeping the extraneous atom information allows a subdomain to calculate interactions with its atoms that occur within its neighbours without needing to communicate with them for atomic information constantly [71]. Atoms that move across borders between subdomains or periodic boundary conditions change ownership every few time steps when the lists of atomic neighbours are recalculated [71]. The construction of this neighbour list occurs in O(M) time where M is the number of atoms that a processor is keeping track of (including nearby neighbour atoms outside its subdomain) [71]. A concluding note on LAMMPS is that the units that LAMMPS uses depend on a keyword selected by the user in an input file. There are biologically-relevant units under the 'real' keyword, and others with 'electron', and the most relevant to these purposes are those associated with the key 'metal', which sets time to ps, lengths to Angstroms, and energies to eV [71].

The MD simulations used PBCs, *metal* units, and EAM (for alloys) as the pair interaction style. The interatomic potential was sourced from a model developed by Farkas et al to create equiatomic solid solutions strongly resembling realistic FCC HEA behaviour [33]. A lattice unit cell dimension of 3.552 Angstroms was used and extended 3 times in each cartesian

axis direction (3 on each side of the origin), producing a FCC HEA system containing 864 atoms of five types (Fe, Ni, Cr, Co, Cu). To create a vacancy, a single atom was deleted from the system, after which the system was allowed to evolve in steps of 1 fs. Data was exported from the system in the form of dump files at intervals of 100 fs to provide a set of files constituting a continuous trajectory. The temperature used was 1500 K, since this allowed for a rate of vacancy migration in the MD simulation of roughly 1E-2 to 1E-3 vacancy transitions per fs of simulation time, which was practical for the purposes of a proof-of-concept model.

After the initial MD trajectories had been produced, they were converted to data files as described in the following section and the files were each minimized using LAMMPS in order to reduce noise in the system from atomic vibrations to facilitate extraction of the defect site. The minimization process parameters and force field are identical to the previous ones used in the MD simulation, minus the system construction steps. The LAMMPS minimization command takes in four inputs that decide when the minimization of the objective function is complete: energy tolerance, force tolerance, maximum number of iterations, and the maximum number of evaluations. The values of each of these were set to 1E-4, 1E-7 eV/Å, 1000, and 10000 respectively, where energy tolerance is the unitless relative change in energy between iterations [145]. Minimization in LAMMPS involves iteratively minimizing the objective function (See Equation 8.1), which is the total potential energy as a function of the position of each of the N atoms in the system and is composed of the sum of the bonded interaction energies, long-range interaction energies, and various other constraints that have less obvious physical analogues [71, 145].

$$\begin{aligned} E(r_1, r_2, ...r_N) = &\sum_{i,j} E_{pair}(r_i, r_j) + \sum_{ij} E_{bond}(r_i, r_j) + \sum_{ijk} E_{angle}(r_i, r_j, r_k) \\ &+ \sum_{ijkl} E_{dihedral}(r_i, r_j, r_k, r_l) + \sum_{ijkl} E_{improper}(r_i, r_j, r_k, r_l) + \sum_{i} E_{fix}(r_i) \end{aligned} \tag{8.1}$$

In order to both interactively view and post-process molecular dynamics data, the open source software OVITO was used extensively [146]. The key feature of OVITO is the data

pipeline, an object that allows for multiple operations to be dynamically applied in real time to the data and adjusted, such as cluster analysis, slicing, filtering, colour-coding, etc [146]. The resulting frame(s) of data can be saved at the other end of the pipeline as new files, viewed through a detailed user interface, or saved for a figure in publication [146]. Operations on a data pipeline can be rearranged and this immediately updates the visualization of the data objects in the pipeline, as they are non-destructive in nature (i.e. they do not apply permanent changes to the underlying data in the pipeline) [147]. Flexibility and modularity are key to OVITO, and users can, with the help of the Python scripting reference for OVITO, even create custom operations and atomic properties for the pipeline themselves [147].

Initial OVITO pipelines were created from the input LAMMPS MD files and used simply to convert the dump files en masse to data files suitable for additional simulation work algorithm 4. The data files were then minimized in another brief LAMMPS MD simulation. The final frame of each of these minimized frame trajectories was then saved and a complete minimized version of the original trajectory of the LAMMPS simulation was made using the OVITO Python interface [147]. This process makes it so that the vacancy in the system can very easily be detected in OVITO by using the pipeline modifiers *Common neighbor analysis*, *Select type* (FCC) and *Delete selected* in sequence. In the conceptual case of multiple vacancies in the system, a *Cluster analysis* modifier would be appropriate as well, to select atoms belonging to each vacancy region of interest and tag them accordingly in the output. When necessary, a custom pipeline function was used to assign elements as a particle attribute, as atom labelling was done by arbitrary numbers in dump files and not by element names or atomic numbers as might be expected.

A suitable facsimile of the MD vacancy defect for use in a proof-of-concept model was made by mining vacancy topology data in OVITO. The data constructed a NX blank template graph that was uniform-randomly assigned 12 node numbers and an atom type integer between 1-5. This ensured that the toy model data had exactly the same shape, features, and variety that would be expected from the MD graphs.

To lend realism to the behaviour of the model, each of the integers 1-5 that represented an

Algorithm 4 The process used to clean and sift MD data to identify each defect state present. Over the preset duration of the MD simulation, as long as the time t was less than t_{limit}, data was saved at intervals of 100 fs, producing n saved files, where $n = t_{limit}/100$ fs. After minimization and vacancy identification was completed, any extracted defects that did not match the bond topology of the ideal vacancy, $template$, were subject to a looped re-processing in which incrementally larger bond lengths, $bond\_len$ were used in OVITO up to a user defined threshold value, $bond\_limit$. This reduced the number of malformed graphs from roughly ∼7% of the data investigated to << 1%

while $t < t_{limit}$ **do**
  Evolve system in time, t
  if t % 100 fs == 0 **then**
    Export MD data for system as step_t.dump
for i in range(n) **do**
  convert "step_[i].dump" → "step_[i].data"
for i in range(n) **do**
  Run LAMMPS minimize "step_[i].data" → "step_[i]_min.dump"
for i in range(n) **do**
  Pull final frame "step_[i]_min[-1].dump" → "step_[i]_min_final.dump"
for i in range(n) **do**
  Process "step_[i]_min_final.dump" in OVITO → *state, time, frame*
for i in range(n) if state[i == "Vacancy"] **do**
  if OVITO topology[i] == $template$ **then**
    Process "step_[i]_min_final.dump" in OVITO → *topology, elements, IDs*
  else
    $noncompliant$.append(i)
while $bond\_len$ < $bond\_limit$ **do**
  $bond\_len$ += 0.1
  for j in $noncompliant$ **do**
    if OVITO topology[j] == $template$ **then**
      Process "step_[j]_min_final.dump" in OVITO → *topology, elements, IDs*
      delete j from $noncompliant$
Export $noncompliant$ → "$noncompliant$.txt"

atom type in the HEA were randomly assigned an atomic mass reasonable for a transition metal (40-60 amu) and a base migration energy in the range 0.6-0.8 eV at the same order of magnitude as experimental HEA vacancy migration energy barriers [95]. A temperature variable was also used, preset to 1000 K. The lattice parameter 'a' used was the same as in the MD simulations, 3.552 Å. Rates for the system that produced the results presented for the proof-of-concept model were (in ns^{-1}) *[1: 0.71, 2: 2.89, 3: 0.37, 4: 0.62, 5: 0.58]*, rounded to two decimal places.

Equation 8.2 and Equation 8.3 were used to govern the attempt frequency and the escape

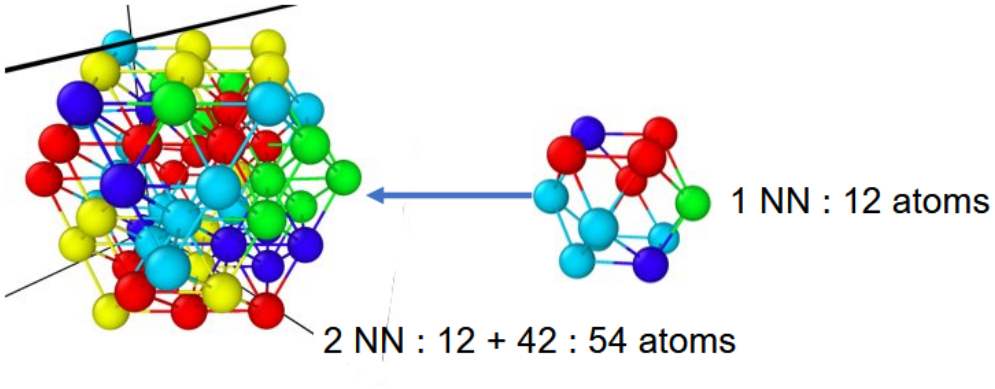


Figure 8.4: The two configurations of FCC regions-of-interest used in this work, 1-NNI (right) and 2-NNI (left). For the 1-NNI environment, only 12 atoms are included, the ones immediately surrounding the vacancy. For the 2-NNI environment, 54 atoms are included, including all the atoms from the 0-NNI/1-NNI as well as their own respective graph nearest neighbours. The reason that these environments were chosen is that the transition of a vacancy defect is influenced both by the immediate neighbours and their neighbours (See Figure 8.1) [25, 148].

rates for movement of the vacancy to each allowed atom. The equation for the attempt frequency of a vacancy defect transition A is Equation 8.2, where for the sake of simplicity of the toy model the mass M and migration energies E_V^m were the arithmetic mean of the 5 values present in the system contents [61].

$$A = \frac{1}{a}\sqrt{\frac{E_V^m}{M}} \tag{8.2}$$

The Arrhenius equation, Equation 8.3, determined the rate at which an atom in the 3D toy model made a move to position itself at the vacancy site. Thus in this case the factor A is the same attempt frequency from Equation 8.2.

$$k = Ae^{\frac{-E_a}{k_B T}} \tag{8.3}$$

Next the evolution of the system was set up using Kinetic Monte Carlo (KMC) rules. Rate constants for different pathways/atoms were scaled to sum to unity by dividing by k_{tot}. The selection of the next state was performed by taking a uniformly-distributed random number, residence/escape time from the state was obtained by drawing an exponentially-distributed

random variable from a generator parameterized by k_{tot}.

For the toy proof-of-concept system, interatomic interactions influencing the escape rates through each atom in the vacancy environment were constructed according to three interaction modes: 0-NNI, 1-NNI, and 2-NNI. Each of these interaction modes were named in a graph, rather than a crystalline sense (i.e. a graph neighbourhood), abstracting the vacant space inside the vacancy cell as a central node and working outwards. For all interaction modes, atoms were assigned rate constants as described previously. In the 0-NNI case, these base rates were simply summed to generate the total escape rate k_{tot} without modification. For the 1-NNI and 2-NNI cases, a symmetric $5x5$ interaction matrix was randomly generated to influence the base rate with respect to atomic neighbourhoods. The interaction of the first atom type with itself was set to 1. Then the neighbourhoods of each atom/node were obtained from their graph representations and converted to vectors encoding the counts for each atom type (e.g. An atom node might have a neighbourhood set of 4 atoms belonging to one of 5 types converted as [1,3,4,1] $\rightarrow$ [2 0 1 1 0]). The host node/atom was not included in this vector, instead a vector from the interaction matrix corresponding to interactions between the host atom type was taken and the product of the neighbourhood count vector with this was used to produce a factor that modified the base rate for the host atom/node. The 0-NNI and 1-NNI setups both used graphs of identical construction, containing the 12 atoms immediately surrounding the vacancy, and the 2-NNI setup used larger graphs containing a total of 54 atoms surrounding the vacancy (See Figure 8.4). For the 2-NNI case, only the 12 core atoms around the vacancy ultimately contributed to k_{tot}, (rate predictions for the rest were zeroed to prohibit them as transition targets) as per the assumptions made earlier in this work. For the proof-of-concept model data shown in Section 8.4, all datasets were generated from the same interaction rules via a fixed seed. Likewise, the compositions, chosen atoms, and times were fixed, with the exception of the 2-NNI case, which generated different states and times due to the increased number of random draws needed for additional atoms/nodes.

8.3.2 Data Representation

There are a number of benefits to representing spatial systems such as cubic crystals using graphs instead of matrices/grids or flattened input vectors. Firstly, the visualization is an intuitive one. In addition, periodic boundaries are easily represented by a graph structure, as nodes representing boundary sections of the atomic system can be connected by graph edges. The edges and nodes of a graph are also versatile and can store multiple features simply by adding new keywords to the NX package graph constructors.

In order to test the viability of designing a graph/grid based system to represent a crystal structure, the Ising model case was used, with 'atoms' having randomly-assigned values of [-1,1] on a square 2D lattice. Once the construction techniques were successfully implemented in this environment, they were adapted to the MD data.

Extraction of vacancy defect data from MD was accomplished by extracting atoms and topology using OVITO (algorithm 5 and Figure 8.5) and then creating GNN-compatible graphs (algorithm 6). Extracting defect data from the toy proof-of-concept system for representations was trivial in comparison, as the defect states were constructed in-situ without any extra components.

In order to put graphs containing n nodes each with k node features and e edges with f edge features into a GCN, they are converted into three objects: A $n\ x\ n$ adjacency matrix (as described in Figure 5.3), a $n\ x\ k$ *node feature matrix*, and an $e\ x\ f$ *edge feature matrix*.

Algorithm 5: Extraction of topological details of the i_{th} of n vacancy defect environments in OVITO, as performed in this work via the OVITO Python package. Steps 2-5 are intended to be applied as an OVITO pipeline modifier stack. Images of the process from the standalone OVITO application are shown for reference in Figure 8.5.

1: Open defect file [i].dump in OVITO
2: CNA $\xrightarrow{\text{Select type (Other)}}$
3: Expand selection $\xrightarrow{\text{range = 3 Å}}$
4: Invert selection
5: Delete selected
6: Apply bonds $\xrightarrow{\text{universal cutoff = 3 Å}}$
7: Save **Topology** from **Bond table**

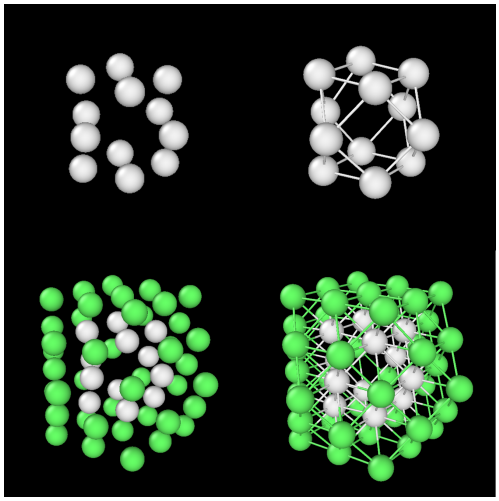
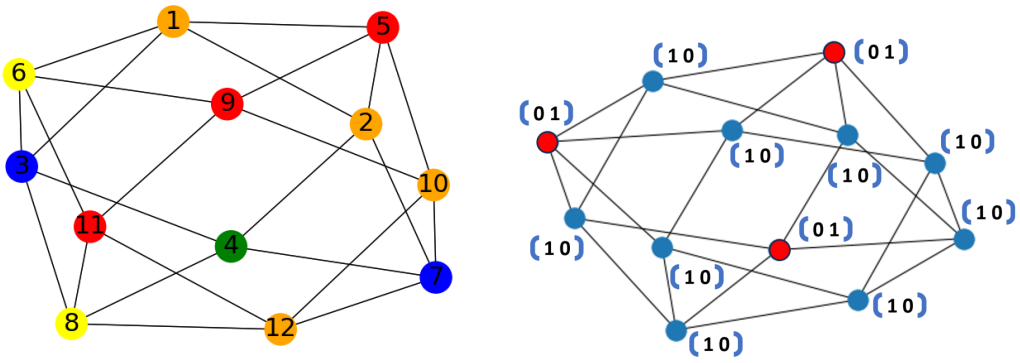


Figure 8.5: OVITO views of atoms involved in algorithm 5. The 0-NNI/1-NNI environment is shown on the left, obtained by skipping step 3. The 2-NNI environment is shown on the right, obtained from algorithm 5 in full. Top and bottom views display before and after step 5 of algorithm 5, with the core 12 atoms allowed for transitions in white.

Algorithm 6 Construction of a list G_list of n graphs G from n MD vacancy states (data extracted with algorithm 5) for use in GCN models, using both the Python OVITO package and NX packages.

```
Extract Particle_identifier, Particle_type, Topology from OVITO via algorithm 5
G_list = []
for p in range(n) do
    G = nx.graph()
    graph_data = []
    edge_data = []
    for i, j in zip(Particle_identifier[p], Particle_type[p]) do
        graph_data.append((i, {"Type": j}))
    G.add_nodes_from(graph_data)
    for i, j in Topology[p] do
        edge_data.append((Particle_identifier[p][i], Particle_identifier[p][j]))
    G.add_edges_from(edge_data)
    G_list.append(G)
return G_list
```

An arbitrary vacancy graph assumes the form seen in Figure 8.6 (a) and is converted to a 1-hot representation of itself as follows in Figure 8.6b.



(a) (b)

Figure 8.6: Graph representations from NX of the vacancy defect 1-NNI environment. (a) A layout captured and labelled in NX for the proof-of-concept data, showing nodes coloured based on their atom feature (type) and numbered by arbitrary ID number tags. (b) An arbitrary and unlabelled graph layout modified to demonstrate a one-hot encoding scheme for a binary alloy

8.3.3 Dynamics Learner Function

Algorithm 7 The unsupervised dynamics learner loss function used in this work, adapted from [1]. The log-likeihood U of a trajectory $omega$ composed of K states C with transition rates W is given by $U_\omega = \sum_{k=0}^{K-1}(lnW_{C_k \to k+1} - \Delta t_{C_k} R_{C_k})$, where R_{C_k} is the total escape rate, given by $R_{C_k} = \sum_{C'} W_{C_k \to C'}$ [1]. In this work, the original version was modified to exclude the special case of the final step of a trajectory, and the final steps were accordingly deleted from the training data before use.

Require: ML model $model$, state transition times $times$, chosen escape indices $idxs$, graph node features x, graph edges $edges$, and number of nodes per graph n.

1: $lograte s = model(x, edges)$.reshape$(-1, n)$
2: $rates =$ torch.exp$(lograte s)$
3: $R =$ torch.sum$(rates,$ dim = 1$)$
4: $loglike =$ torch.gather$(lograte s, 1, idxs)$.sum$() - (times \times R)$.sum$()$
5: $loss = -1 \times loglike$
6: $trainloss$ += $loss$.detach$()$
7: $loss$.backward$()$, optimizer.step$()$, optimizer.zero_grad$()$

By using the negative of the log-likelihood function for the dynamics of a Markovian system, Casert et al used the log-likelihood of a trajectory as a source of feedback to a ML model in the place of a conventional loss function [1]. This approach was used with a few modifications. Firstly, the last entries of the training trajectories were simply deleted instead of accommodated as special cases. Training was also done on batches of the trajectory and did not include a secondary training stage over the full trajectory for fine-tuning.

8.3.4 Graph Network Architecture

To examine the impact of the mode of interaction present in the proof-of-concept data, two Pytorch models were implemented: a Graph-based network without convolutional layers ("BaseNet") and a GCN with 1 GraphConv convolutional layer. Both networks had a depth of 3 layers in total and used the same dynamics learner loss function (See algorithm 7) based on log-likelihood estimation from Section 6.5 [1]. For both networks the torch seeds were fixed and torch was set for deterministic behaviour. In order to obtain a reliable sampling of results from model training and testing, a set of 5 different seed values were used to produce the data shown on tables and plots in Section 8.4. The data shown in Section 8.4 was produced after training for 100 epochs with a learning rate of 2.5E-4 and a batch size of 250. The size of the training set varied, while the testing set was comprised of 4000 graphs in each case, with the same batch size as the training set.

In addition to the use of the log-likelihood dynamics learner loss function, the L1 (MAE) loss was evaluated during training and testing of both models, but not used in backpropagation or optimization. This was done in order to compare results between different models and datasets more effectively. In a practical application with unknown true dynamics such as the MD dataset, an alternative performance metric is necessary, as discussed later in this work.

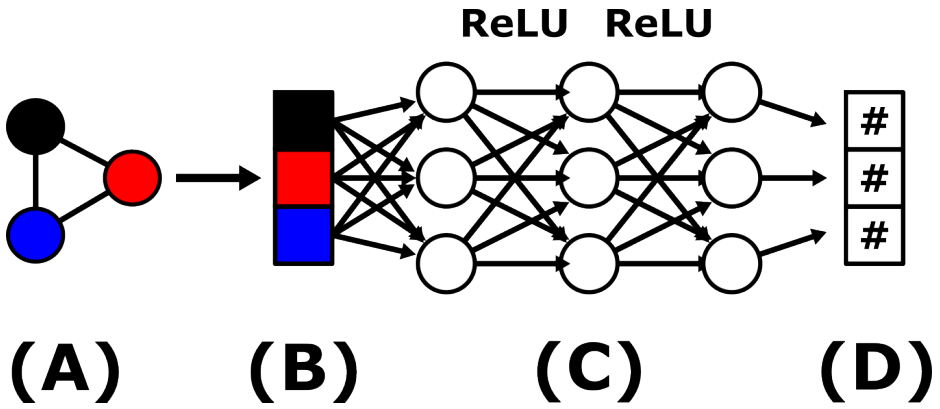


Figure 8.7: A summary of the non-convolutional GNN structure "BaseNet" used in this work. Initially, (A) a graph input is converted into (B) node features which are then each passed through (C) a pair of Pytorch Geometric Linear layers with ReLU activation before (D) a final layer returns a single float value prediction for the rate constant per input atom/node. This process (C)-(D) is performed for each node in the graph, for each graph in a batch of the dataset during training. Note that in this diagram the tricoloured graph nodes are stand-ins for one-hot encoded atom features such as those shown in Figure 8.6b.

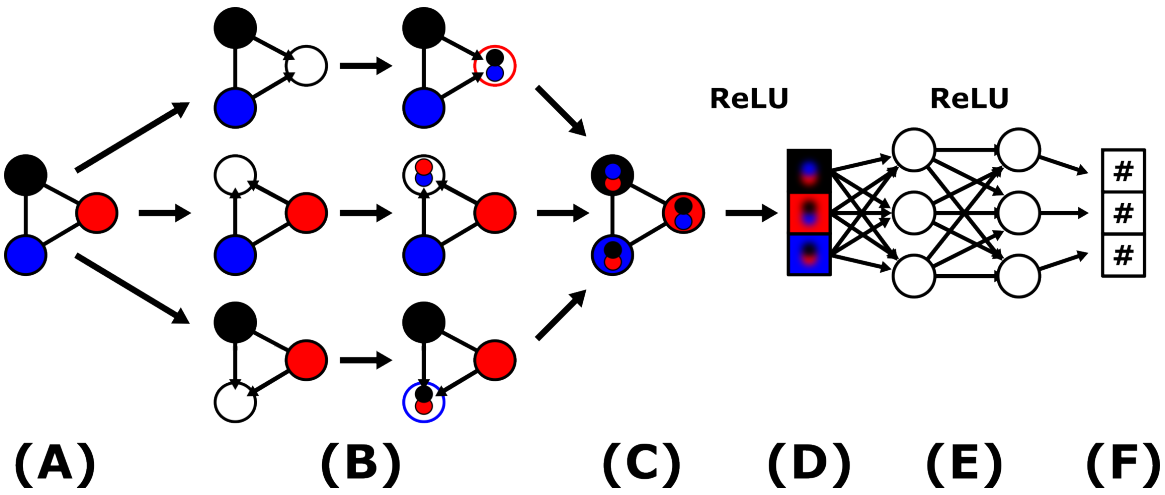


Figure 8.8: A summary of the GraphConv GCN structure used in this work. Initially, (A) a graph input is converted passed through (B) a single Pytorch Geometric GraphConv convolutional layer which collects information on the host and neighbour nodes (white and black/blue/red in (B) respectively) before (C) applying a ReLU activation and passing the convolved graph (D) node features into (E) a Pytorch Geometric Linear layer with ReLU activation and an (F) output layer identical in form to the one used in the BaseNet (Figure 8.7). Note that in this diagram the tricoloured graph nodes are stand-ins for one-hot encoded atom features such as those shown in Figure 8.6b.

The GCN formulation used by Kipf and Welling (Equation 6.26) is a common starting point for GCNs, however for this purpose it is insufficient due to how this technique handles neighbour data. In the convolution, the multiplication of the adjacency matrix and the feature matrix results in a collection of node features which does not preserve the relative significance of the host node (in the case of self-loops). Without self-loops, the information associated with the host node that is necessary to determine the correct value of the desired property is also lost, when equivalent neighbourhoods are taken into account. Thus a convolution that allowed for emphasis to be placed on the relationship between the host node and the peripheral nodes was essential instead.

This was implemented with GraphConv, a method that places emphasis on the central node [38]. In this convolution, the central node $\mathbf{x_i}$ and the neighbourhood nodes $\mathbf{x_j}$ with edge weights $e_{j,i}$ have their own weight matrices W_1 and W_2 respectively (Equation 8.4) [38]. The alternative convolutional layer Crystal Graph Convolution (CGConv) was briefly tested due to also distinguishing the central node, but was more difficult to set up and did not appear to provide a crucial advantage in early stages of this work.

$$[ht]\mathbf{x_i'} = \mathbf{W_1}\mathbf{x_i} + \mathbf{W_2}\sum_{j\in\mathcal{N}(i)} e_{j,i}\cdot\mathbf{x_j} \tag{8.4}$$

The model interpretability demonstrated by the designers of the CGConv technique served as a guideline for obtaining network outputs that map output properties of a crystalline input graph to the local environment(s) of each node. A modification of this technique is illustrated in Figure 8.8.

8.3.5 Evolving Systems via Model

Using an arbitrary trained model configured as shown in Figure 8.7 or Figure 8.8, an algorithm was built to take predictions from the model and use them to produce an ML-accelerated trajectory using the model to provide transition and escape rates. This algorithm was dubbed "EvoSys" and made to export ML-driven trajectories in the form of *.dump* files. Note that this "EvoSys" code currently uses the centroid rather than the centre of mass of a vacancy defect when making a transition, due to the way that OVITO processes labelled atom data.

Algorithm 8 Given a model *model*, a trained model state *trained*, a duration T, a starting defect state *initial_state*, generate synthetic trajectories in MD file format.

Require: $model$, $trained$, T, $initial\_state$

Load $model$ with saved state $trained$
$init\_pipe$ = OVITO.import_file($initial\_state$)
self.$Ovstate$ = $init\_pipe$.compute(0)
self.G = nx.graph()
Run self.init_graph_system()
    CreateBondsModifier → extract bond and atom data from OVITO
    Use nx.add_nodes_from() and nx.add_edges_from to add to self.G
Run self.init_find_vac()
    Apply pipeline modifiers from algorithm 5 steps 2-5
    ClusterAnalysisModifier $\xrightarrow{\text{cutoff = 3 Å, compute CoM}}$
Run self.init_graph_vac()
    G_defect = deepcopy(self.G)
    Delete nodes in G_defect for atom IDs not returned by cluster analysis
    Split G_defect into separate graphs for each vacancy detected → self.$defect\_graphs$
for i in $defect\_graphs$ **do**
    Run self.vac_encoding(i) → self.Predictor(i, m) → save prediction values for index, log-likelihood, residence time t, etc.
    if self.collision_check(i, $defect\_graphs$) == False **then**
        Run self.updates(i) to modify G and self.$Ovstate$, with accepted atom move
        OVITO export self.$Ovstate$
    while $t < T$ **do**
        argmin(t) for defect(s) in system → repeat steps 16-19 for corresponding defect i

8.4 Results

8.4.1 MD Dataset

7.5E4 state transitions were isolated from 10 MD trajectories containing roughly 9E6 frames in total, with about 3.3E4 unique core (12 atom) states. Approximately 4E4 of the states captured were "high-entropy" states, in that they contained at least one example of every type of element present in the system. Using the high-entropy states, the chosen atom type for each associated transition was recorded and weighted according to the relative proportion of the HEA vacancy environment that it occupied (out of the 12 core atoms considered for transitions), in order to estimate the relative average probabilities across the dataset associated with a transition to each of the elements: Fe, Ni, Cr, Co, Cu (Figure 8.9). These values are expected to vary locally due to composition/configuration factors in HEAs

(i.e. mixtures of atomic radii, bonding interactions, etc. [28, 97]), but serve as a useful benchmark and when combined with residence time data can provide details on the order of magnitude for unknown true rates in the MD simulations. The trajectory data shows a strong preference for migrations escaping through Cu atoms (~0.7), with the closest following, Cr, being chosen in less than a fifth of transitions in high-entropy states. Of the remaining three least likely atoms in Figure 8.9, Ni and Co were chosen with almost equal proportions and Fe was last, with very few vacancy transitions in high-entropy states occurring by Fe paths. Encouragingly, the marked preference for one particular atom type was shared by the proof-of-concept system, where synthetic dynamics were implemented to mimic a simplified HEA system.

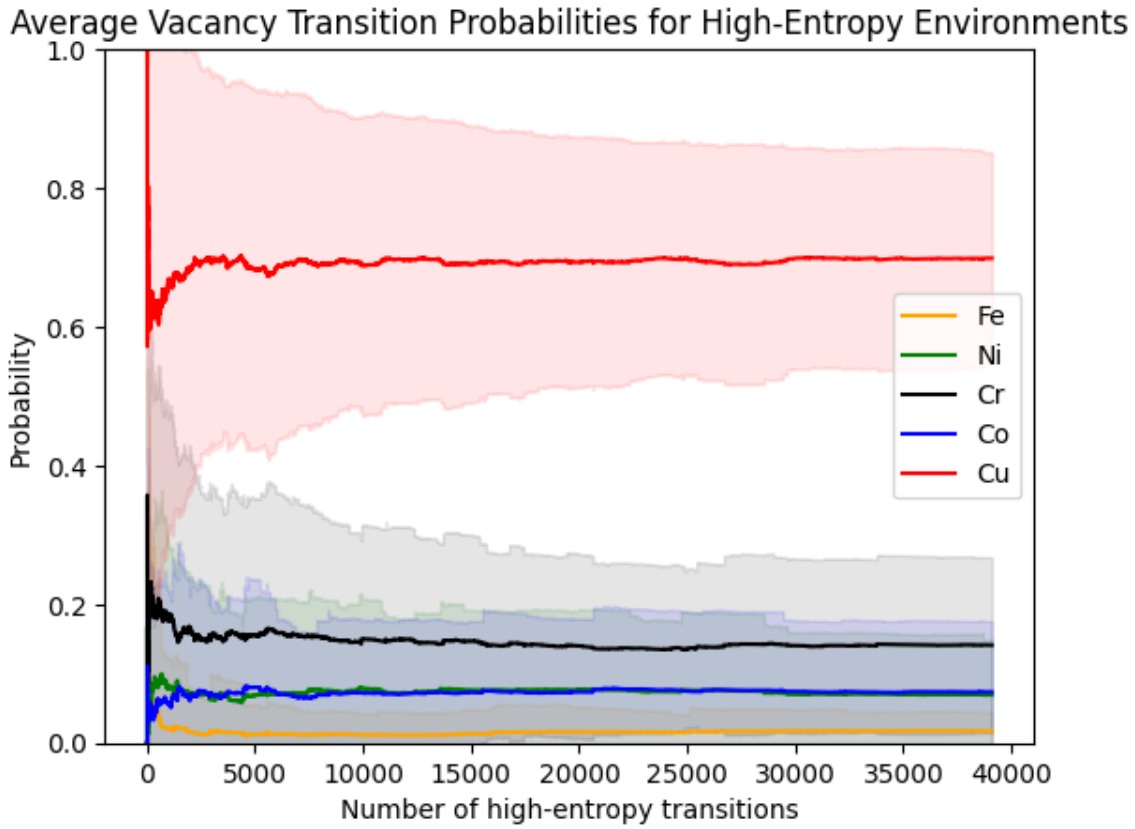


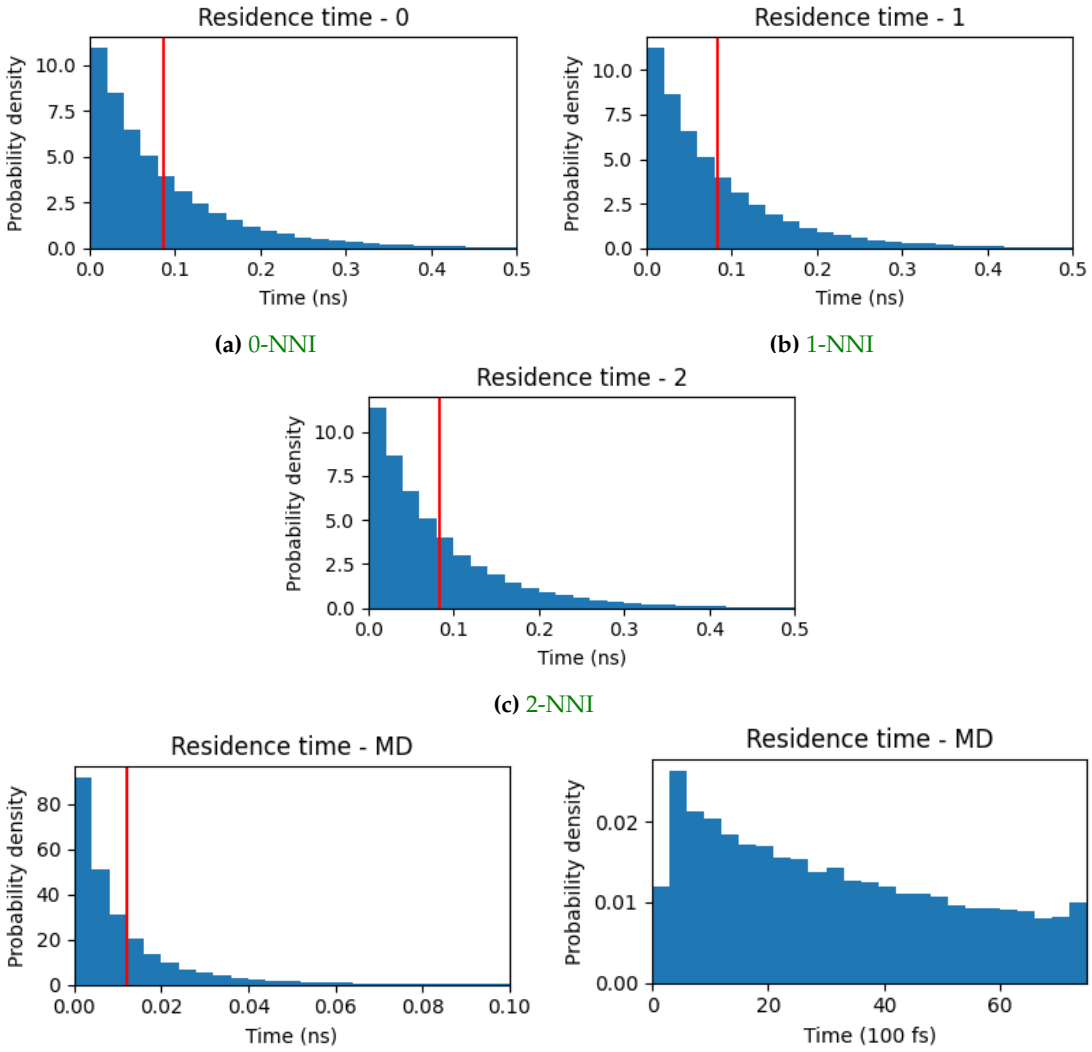
Figure 8.9: A plot of the average atom types selected in vacancy transitions over time for MD simulations. These values were obtained from 3.9E4 states containing at least one instance of each of the 5 atom types present in the HEA (Fe, Ni, Cr, Co, Cu). Shaded regions indicate the standard deviation for each element type.

In order to observe how frequently the vacancy in the MD trajectories was filled by the exact same atom, all instances where the chosen atom was identical to the next chosen atom were

catalogued. Instances of this back-and-forth came to a total of 839 across all trajectories. This represents ∼1.1 % of the total dataset. These are expected to represent trapping states or kinetic traps present in the MD simulations.

For a Poisson process (such as a single defect escape/transition) the times between events are exponentially-distributed. As each state along a trajectory obeys its own exponential distribution dictated by its configuration, the plot of residence times in both the toy proof-of-concept and the MD data is expected to take the form of a *mixed distribution* composed of a weighted average of exponential distributions belonging to each of the configurations sampled in the course of the trajectory. The MD data contained 1180 unique states by composition (i.e. ignoring configuration), which is more than half of the composition space. Consider the set of k_{tot} rates present in the simplest proof-of-concept interaction setup, 0-NNI. With 5 float values representing the rate constants k_{ij} which lead from state *i* to *j*, for 12 atoms, with replacement, the combinations are given by $C_r^n = \frac{n+r-1}{r!(n-1)!}$. This yields $C_{12}^5 = \frac{(5+12-1)!}{12!(5-1)!} = \frac{16!}{12!4!} = \frac{43680}{24} = 1820$ values for k_{tot}. Note that these are not necessarily unique values, but with float valued and randomly selected rates, they are likely to be distinct nonetheless. As each of the atoms in the proof-of-concept system was selected randomly and with equiatomic fractions, the distribution of k_{tot} values sampled in 0-NNI under these conditions was expected to follow a multinomial distribution, as if it was the sum of 12 dice rolls where each die had 5 sides (one representing each atom type).

As seen in Figure 8.10, the distributions of residence times for the proof-of-concept states as well as the MD states for the most part appeared as expected to be similar to an exponential distribution. Closer analysis would be necessary to determine how much the times deviated from that of a pure exponential distribution. Each of the proof-of-concept datasets had a similar rounded mean value of 0.09 or 0.08 ns, while the MD histogram had a much smaller mean value of only about 0.01 ns, which is explained due to coming from a different dynamics setup as well as a higher temperature. An additional observation is that at a closer view, the histogram of residence times has a lower first bin value than what would be expected in an exponential or mixed exponential distribution. The cause of this anomaly was likely that the timesteps produced by the MD simulation were discrete, and so any transitions with a lower residence time than 100 fs were not detectable from the dataset.



(a) 0-NNI **(b)** 1-NNI

(c) 2-NNI

(d) MD

Figure 8.10: Histograms of residence times for states from the three proof-of-concept datasets as well as the MD dataset. The right side figure of d) is an enlarged view of the MD data for the range 0-75, showing the reduced value of the first bin of the histogram compared to its neighbours.

8.4.2 Network Training & Testing

For the proof-of-concept toy model system, the dynamics were completely predetermined, and as such they were evaluated simply by taking the L1 (MAE) loss of the predicted rates from the trained models. All tables values are rounded to 2 decimal places. Training curves are visualized with rolling window smoothing set to a value of 5 steps. The trajectories generated as training and testing sets for the proof-of-concept model had true log-likelihood

values between roughly -8.2 to -8.5.

As can be seen from Figure 8.14b BaseNet results in particular, the log-likelihood Δ loss values from the test datasets were often smaller than the log-likelihood values evaluated from the training datasets. This behaviour is explained by the fact that the log-likelihood values, even those with the same generating dynamics, are not identical, as the exact transitions and times vary between trajectories. With this in mind, comparison of final differences between the log-likelihood values for different trajectories may lead to erroneous conclusions. The general trend of the final log-likelihood Δ loss for each trajectory (train and test) shrinking in magnitude as the total number of samples increases may be due to both models improving with larger datasets, but again it is cautioned that log-likelihood values from the same model over different datasets are not directly comparable. One useful observation from the log-likelihood results in Figure 8.13 is that each plot shows the expected smooth decrease in the difference between the training and exact log-likelihoods for the proof-of-concept model [1]. The MAE serving as a performance metric in the proof-of-concept case provides more stimulus for discussion in Figure 8.12.

The 0-NNI mode showed both in testing and training that the BaseNet GNN outperformed the GraphConv GCN for the L1 loss, with even the worst-performing set of training runs ([5,1000] case) being slightly better than the best case of the GCN ([5,4000] case) at learning the true rate values (Figure 8.12(a). The BaseNet did similarly well between both of its best sets, the 20000 sample cases ([1,20000] and [5,4000]), and slightly worse on the 5-repeat 5000 graph sample case than the no-repeat one. This last pair have some overlap in their standard deviation, and may prove more similar with additional trials. Of the two best performing baseline cases, the 5-repeat set appeared to have a similar value to the no-repeat set in both training and testing, but with roughly half the standard deviation on the test data, suggesting that the dataset was ample to allow for the dynamics to be learned and that the additional examples proved useful to the BaseNet. Like the BaseNet, the GCN benefited in the 0-NNI case from additional training examples, however the improvement in the L1 loss diminished with the use of the larger 20000 sample cases ([1,20000], [5,4000]) and the standard deviations did not change noticeably for any of the test sets with additional repeats. In the 0-NNI L1 plot, some signs of overfitting were observed from the [1,5000]

case during training, as the loss began to increase towards the end of the training run. This was also suggested by the corresponding log-likelihood loss plot in Figure 8.14(a), which showed that the [1,5000] set of runs dipped well into the negative.

The subpar performance of the GCN compared to the BaseNet was attributed to the fact that the 0-NNI system had no interactions between atoms whatsoever, and as such, the GraphConv convolutional layer in the GCN would be required to learn to ignore any information obtained from convolution (the W_2 weights in Equation 8.4), which the model structure is biased against by the nature of convolution. For 0-NNI both networks were still able to learn the true values (See Subsection 8.3.1) to within < 0.05 L1 loss.

In contrast to the results in the ideal 0-NNI environment, the BaseNet consistently struggled to fit the dynamics of the proof-of-concept 1-NNI system, with the L1 loss showing little sign of improvement. The BaseNet L1 loss was nearly on-par with the GCN for the two sets of 5000 graph runs ([1,5000] and [5, 1000]), but was definitely surpassed by the longer 20000 graph trajectories for L1 (Figure 8.12(b)), even though the total number of configurations in the [5,4000] state was 1000 states smaller. The superior performance of the GCN was expected with the use of a convolutional layer in an interacting system. The best L1 loss reported for this network for 1-NNI was almost half of the best loss achieved by the BaseNet on the same setup, suggesting that the convolutional layer was able to help the log-likelihood loss function learn the dynamics of the proof-of-concept system. For future work it is recommended that the GCN be given additional training epochs to converge, as the 1-NNI and some of the 2-NNI plots appear to be able to benefit from further training. From the Δ log-likelihood plots it was noted that both of the worst performing runs appeared to have been overfitting, reaching noticeably negative log-likelihood Δ losses. In contrast to the results in the ideal 0-NNI environment, the BaseNet consistently struggled to fit the dynamics of the proof-of-concept 1-NNI system, with the L1 loss showing little sign of improvement.

Trial setups with repeats performed slightly better than their no-repeat counterparts, with similar relative differences between their losses. In general however, the improvement to the L1 losses for each run and network appeared small. Across training and testing runs

Model-System	[1, 5000]	[5, 1000]	[1, 20000]	[5, 4000]
B-0	1.97E-2 (2.6E-3)	2.20E-2 (1.4E-3)	1.05E-2 (1.7E-3)	1.20E-2 (2.4E-3)
G-0	4.69E-2 (3.0E-3)	3.70E-2 (2.3E-3)	2.38E-2 (1.9E-3)	2.44E-2 (1.7E-3)
B-1	6.80E-2 (6E-4)	6.80E-2 (2E-4)	6.64E-2 (8E-4)	6.53E-2 (3E-4)
G-1	5.96E-2 (5.0E-3)	5.73E-2 (5.8E-3)	3.74E-2 (2.6E-3)	3.46E-2 (3.5E-3)
B-2	9.6E-3 (1E-4)	9.3E-3 (2E-4)	9.0E-3 (2E-4)	8.9E-3 (1E-4)
G-2	9.5E-3 (8E-4)	1.07E-2 (7E-4)	6.9E-3 (7E-4)	7.3E-3 (9E-4)

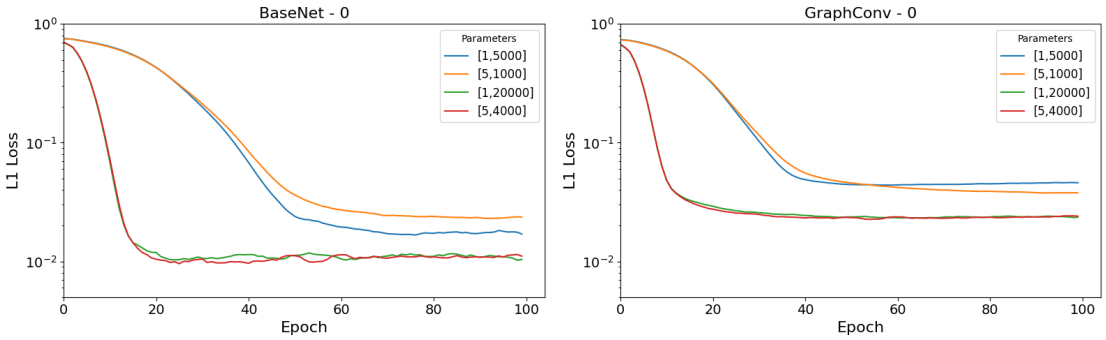
Table 8.1: L1 training loss in the toy proof-of-concept model for the BaseNet (B) and GraphConv GCN (G) ML models and each of the three tested interaction modes, reported as μ (σ) obtained from 5 runs with fixed seeds in each of the 6 test cases. Column titles in the format [x, y] indicate the number of repeated samples of a state that were used in the input dataset as well as the number of randomly generated states created for the input dataset respectively. See Figure 8.12 for corresponding plots.

Model-System	[1, 5000]	[5, 1000]	[1, 20000]	[5, 4000]
B-0	1.83E-2 (3.3E-3)	2.15E-2 (1.8E-3)	1.15E-2 (6.6E-3)	1.12E-2 (3.2E-3)
G-0	4.88E-2 (2.6E-3)	3.69E-2 (2.6E-3)	2.67E-2 (4.0E-3)	2.52E-2 (3.6E-3)
B-1	6.87E-2 (5E-4)	6.78E-2 (2E-4)	6.76E-2 (1.4E-3)	6.62E-2 (1.6E-3)
G-1	6.36E-2 (4.6E-3)	5.35E-2 (4.3E-3)	3.82E-2 (2.2E-3)	3.24E-2 (3.0E-3)
B-2	9.3E-3 (2E-4)	9.2E-3 (1E-4)	8.9E-3 (2E-4)	8.9E-3 (4E-4)
G-2	9.1E-3 (1.0E-3)	1.05E-2 (5E-4)	6.8E-3 (4E-4)	7.2E-3 (5E-4)

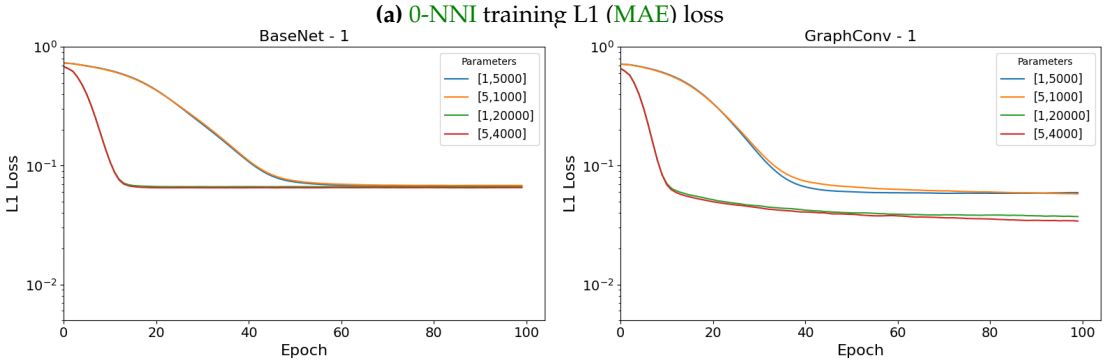
Table 8.2: L1 testing loss in the toy proof-of-concept model for the BaseNet (B) and GraphConv GCN (G) ML models and each of the three tested interaction modes, reported as μ (σ) obtained from 5 runs with fixed seeds in each of the 6 test cases. Column titles in the format [x, y] indicate the number of repeated samples of a state that were used in the input dataset as well as the number of randomly generated states created for the input dataset respectively. See Figure 8.12 for corresponding plots.

with the 1-NNI and 2-NNI interaction setups, the BaseNet L1 losses showed the smallest standard deviations, often several times smaller than their GraphConv GCN counterparts (Figure 8.12). This is tentatively attributed to the BaseNet reaching its lowest L1 loss in the more complex interaction setups by producing an average value for each node/atom type, rather than learning a complex set of rules that the BaseNet architecture is incapable of fully representing. In this case, each BaseNet trial run, reaching the same impasse, would produce very similar results. This supposition can be examined closer using the values produced by the BaseNet model.

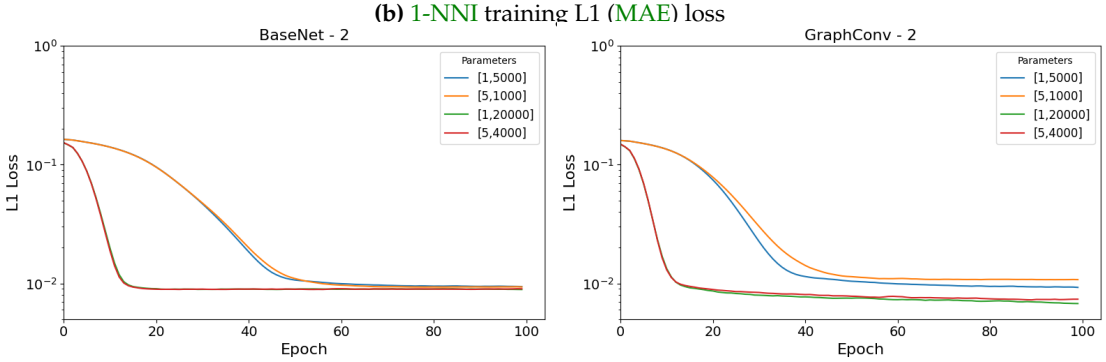
In summary, the BaseNet performed very similarly to the GCN on the non-interacting 0-NNI and 2-NNI interaction setups. The first of these observations is unsurprising, as there are no interaction dynamics at play in the 0-NNI setup. The performance on the



(a) 0-NNI training L1 (MAE) loss



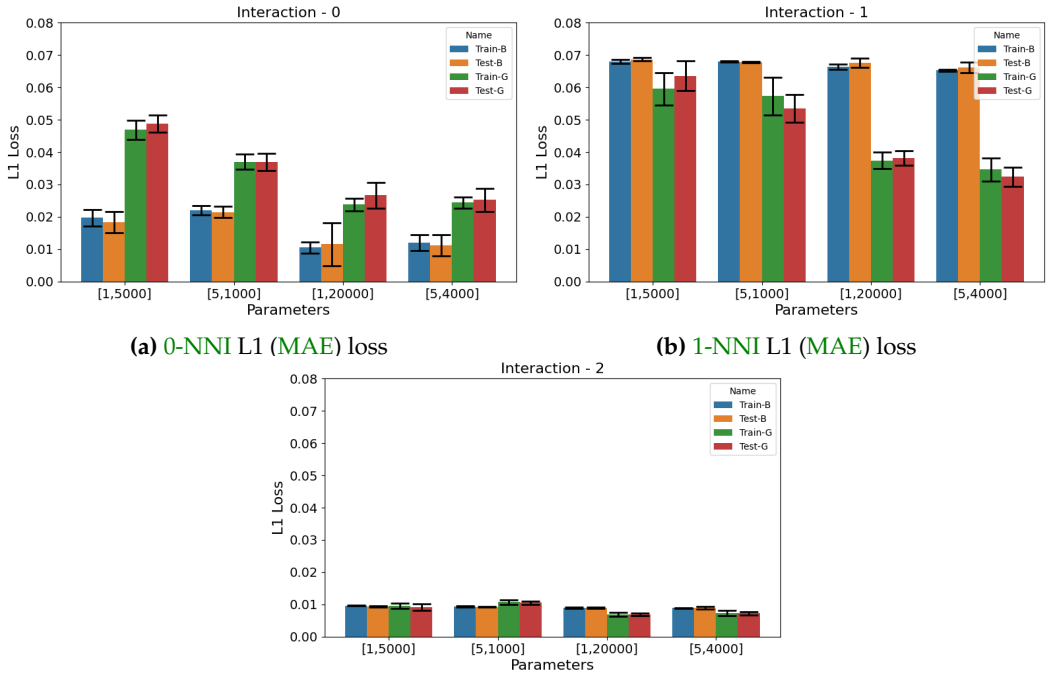
(b) 1-NNI training L1 (MAE) loss



(c) 2-NNI training L1 (MAE) loss

Figure 8.11: Plots of L1 (MAE) training loss in the toy proof-of-concept model for the BaseNet (left) and GraphConv GCN (right) ML models for each of the three tested interaction modes. Each plotted line was obtained from the average of 5 runs with fixed seeds. The legend indicates the number of repeated samples of a state that were used in the input dataset as well as the number of randomly generated states created for the input dataset in the format [x, y] respectively. See Table 8.1 for more details.

2-NNI setup was initially unexpected, as it was anticipated that due to a more complicated interaction scheme involving a larger group of atoms that the simple BaseNet GNN would perform more poorly than in the previous schemes. One explanation that was investigated was that the 2-NNI interaction setup introduced a weaker change in the base rates of the system due to averaging the contribution of more atoms than the 1-NNI setup, making



(a) 0-NNI L1 (MAE) loss

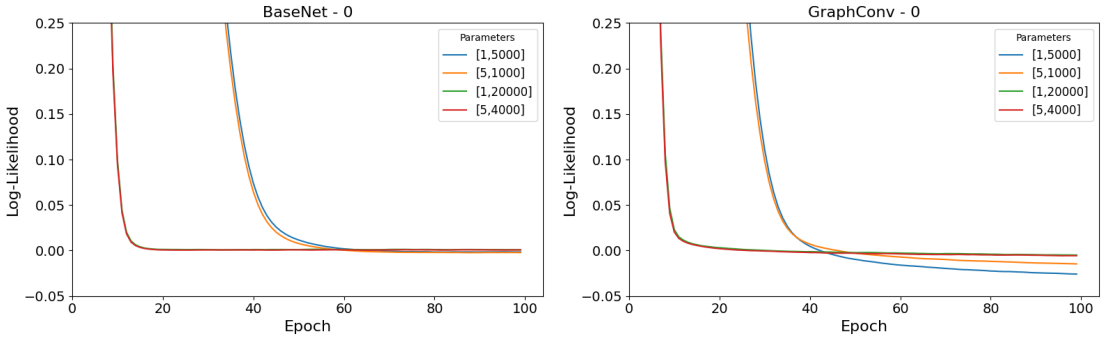
(b) 1-NNI L1 (MAE) loss

(c) 2-NNI L1 (MAE) loss

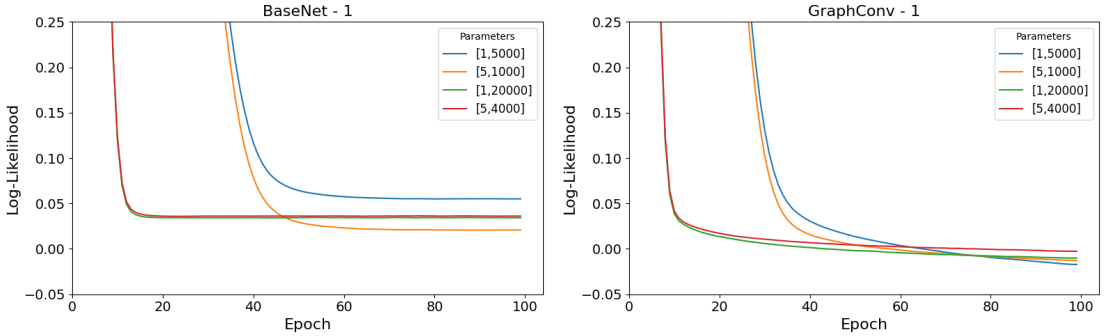
Figure 8.12: Charts of L1 (MAE) losses in the toy proof-of-concept model for the BaseNet (B) and GraphConv GCN (G) ML models for each of the three tested interaction modes. Each bar was obtained from the average of 5 runs with fixed seeds. The x-axis labels indicate the number of repeated samples of a state that were used in the input dataset as well as the number of randomly generated states created for the input dataset in the format [x, y] respectively. Losses were averaged over batches in the dataloader for each dataset, where each batch contained 250 shuffled items. See Table 8.2 and Table 8.1 for specific values.

models that picked the average value for diffusion of a given atom type more accurate. This was supported by the distribution of rate shifts for each of the interaction setups for each element (See Figure 8.15 and Table 8.3).

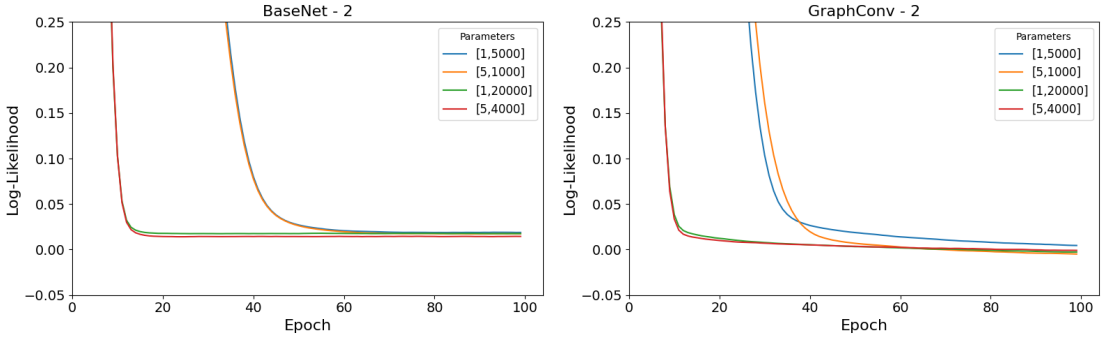
The mean for each of the five randomly-generated elements were nearly identical between setups, as the interactions between pairs in both used the same interaction matrix values. However, the width of the distributions was very different, with standard deviation of 1-NNI being nearly double that found in 2-NNI for all element types. This is taken as confirmation that the 2-NNI setup unintentionally made the dynamics learner problem presented to the graph networks easier instead of harder, leading to an apparent increase in the effectiveness of the BaseNet GNN as well as the GraphConv GCN.



(a) 0-NNI training log-likelihood loss



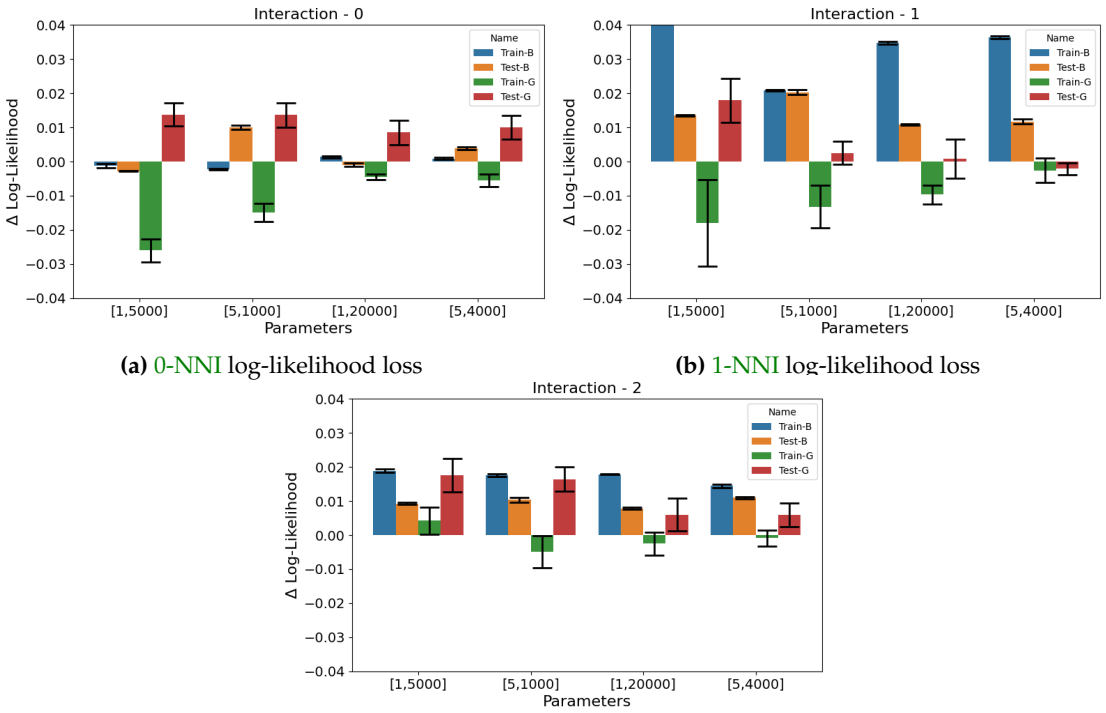
(b) 1-NNI training log-likelihood loss



(c) 2-NNI training log-likelihood loss

Figure 8.13: Plots of log-likelihood Δ training loss in the toy proof-of-concept model for the BaseNet (left) and GraphConv GCN (right) ML models for each of the three tested interaction modes. Each plotted line was obtained from the average of 5 runs with fixed seeds. The legend indicates the number of repeated samples of a state that were used in the input dataset as well as the number of randomly generated states created for the input dataset in the format [x, y] respectively. Values are shown as the difference, $\Delta U_\omega = U_{exact}/T - U_{learned}/T$ between the exact and learned log-likelihoods of the trajectory divided by the total time T of the trajectory ω

.



(a) 0-NNI log-likelihood loss

(b) 1-NNI log-likelihood loss

(c) 2-NNI log-likelihood loss

Figure 8.14: Charts of log-likelihood Δ losses in the toy proof-of-concept model for the BaseNet (B) and GraphConv GCN (G) ML models for each of the three tested interaction modes. Each bar was obtained from the average of 5 runs with fixed seeds. The x-axis labels indicate the number of repeated samples of a state that were used in the input dataset as well as the number of randomly generated states created for the input dataset in the format [x, y] respectively. Values are shown as the difference, $\Delta U_\omega = U_{exact}/T - U_{learned}/T$ between the exact and learned log-likelihoods of the trajectory divided by the total time T of the trajectory ω.

Element	1-NNI shift (%)	2-NNI shift (%)
1	10.88 (8.52)	10.86 (5.18)
2	1.03 (5.92)	1.01 (3.57)
3	8.19 (10.62)	8.39 (6.44)
4	6.39 (10.54)	6.54 (6.41)
5	-0.88 (9.57)	-0.79 (5.77)

Table 8.3: The shifts in rate constants due to different interaction schemes for each randomly generated atom type in this work, compared between the 1-NNI and 2-NNI setups (the 0-NNI case is excluded as the shift is simply zero). Shifts (μ (σ)) are reported in terms of their proportion of the base value for their elements and rounded to 2 decimal places. See Figure 8.15 for corresponding plots.

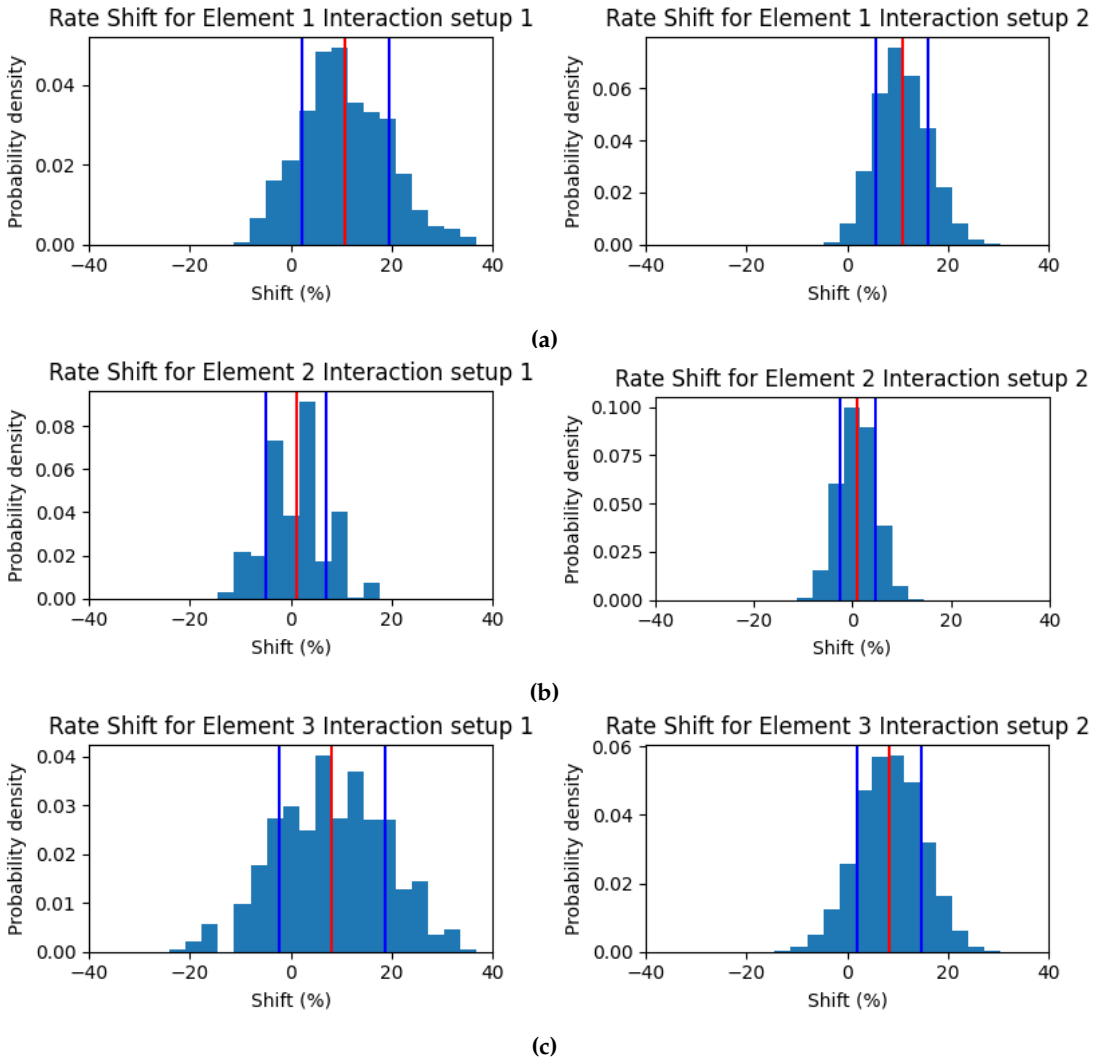
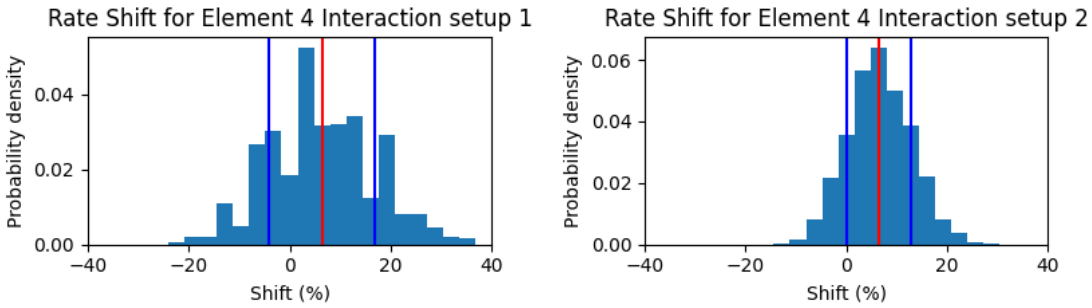
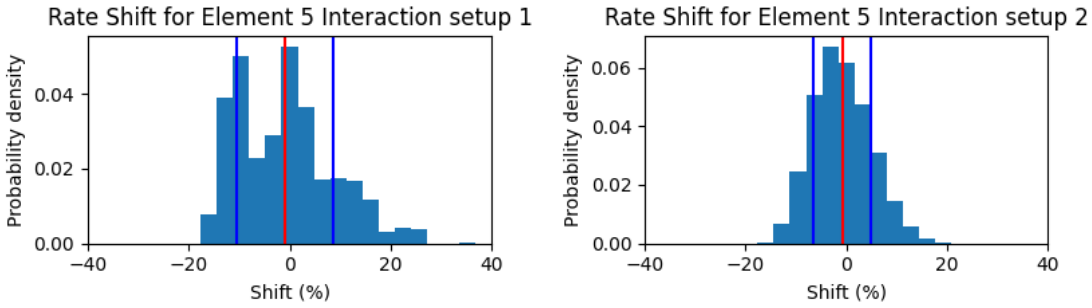


Figure 8.15: A comparison of changes in rate constant values in the two interacting systems used in the proof-of-concept toy model, the 1-NNI and 2-NNI setups. The distributions are presented for arbitrarily generated elements 1-5 in plots a-e respectively, with the 1-NNI figures on the left and the 2-NNI figures on the right. All plots share the same x-range and contain more than 2E4 examples each, from a total of 1E4 example states. Mean and standard deviation are indicated by vertical lines for each case. See Table 8.3 for specific values for each distribution.



(d)



(e)

Figure 8.15: Rate constant shifts between interaction setups (continued).

8.4.3 Generating & Evaluating Trajectories Using Learned Dynamics

The "EvoSys" algorithm was given a GCN model trained on the MD dataset for a few epochs, producing a short sample trajectory of LAMMPS-compatible files evolved state-to-state by learned dynamics, from which a transition was captured in OVITO and edited to create an annotated example, Figure 8.16. Additional work should focus on evaluation of the capability of the algorithm via comparison of its ML-accelerated trajectories to those of the original MD system used to produce the training data. The RMSF technique has been used for determining differences between dynamics in MD systems and is a candidate for these evaluations [144]. The first and second halves of each trajectory are intended for use as controls to examine the self-consistency of diffusion data, as has been done in similar comparisons [144].

Regarding the reliability of using elevated temperature data for practical applications of this work, it is suggested that techniques adapted from Accelerated Molecular Dynamics (AMD) such as Temperature-Accelerated Dynamics (TAD) be implemented to screen transitions for low-temperature regimes and make this technique more generalizable [149].

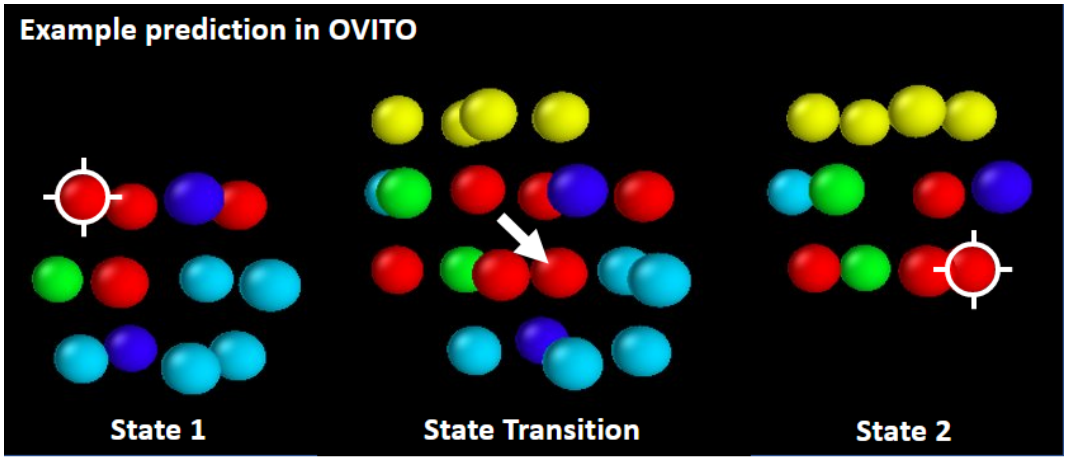


Figure 8.16: An annotated arbitrary example of a single vacancy defect migration made with EvoSys with a trained GCN input.

8.5 Conclusion

A proof-of-concept GCN model for generating KMC trajectories of MD file types using learned vacancy diffusion dynamics was shown, as well as data extraction algorithms for

excising regions of various sizes around vacancy defects from MD data and converting them into a training dataset. Work is ongoing to verify the performance of this technique on true MD data using the previously-discussed HEA potential.

Some more minor next steps to improve the efficiency and/or accuracy of this technique will be briefly discussed. The brute force search for vacancy transitions in the MD dataset could be relatively simply speed up by binary search based algorithm instead by i) Picking the beginning and ending vacancy states in the dataset and ii) iteratively sampling the frames between them for differences in the local environment until iii) continuity between environments has either been achieved or ruled out as an "illegal" transition according to the assumptions previously detailed. The binary search would still be vulnerable to erring if, for instance, a vacancy had migrated and then migrated back to exactly the same environment at a later timestep, which the brute force approach is not fooled by. An additional binary search through the residence times of long-standing states would be the naive solution to apply to avoid this case.

The next update that could be considered would be changing the topological conformity conditions used on the MD dataset to be more adaptive. As it is presently, the code that enforces the ideal bond topology of the HEA vacancy environment rejects and returns vacancies that do not meet user-specified templates. However, the rejected states could still prove salvageable, as they typically exist over several frames in the course of the MD simulation, any of which could prove to be a more relaxed configuration that would pass muster. Therefore, especially in highly distorted systems, it could be a boon to be able to i) list topological nonconforming states and ii) check how long their residence in the system was, in order to iii) pick another frame of the same state and ideally iv) get a more representative sample of it that is further away from the transition event(s). Lastly, the centre of mass calculation used in OVITO is actually the centroid, as the OVITO LAMMPS dump file exporter only reports numeric atom ID types, which unfortunately removes masses and element names. A change to a different exported file type such as *.XYZ* or LAMMPS *.data* would retain those traits.

Testing other graph convolutional layers such as the CGCNN, and CGConv layers would

also be desirable to observe any differences or improvements in speed and performance. The use of different seeds allowing for different base system dynamics in 0-NNI, 1-NNI, and 2-NNI would also be worthwhile to observe which loss trends were specific to the system used.

Chapter 9

Concluding Remarks

Exploration of the space of possible material compositions has been greatly affected by the development of deep learning techniques, allowing for simulations of larger systems, the generation of accurate distributions of atoms in alloys, and the training of surrogate models for DFT, MD, or KMC calculations to ease computational burdens. In particular, the use of graph and graph-convolutional techniques in ML has been a large contributor to this, aided by the fact that graphs are a natural representation for atomistic systems such as molecules and crystals.

At the same time, the class of entropic materials represented by HEAs has emerged as one of great interest to materials science, due to promises of extraordinary properties intrinsic to their highly entropic makeup. The potential applications of HEAs in nuclear technologies in particular have led to many studies explaining and challenging their irradiation resistance and "self-healing" properties.

The damage initially caused by irradiation exists on the nanoscale in the form of point defects in pairs of self-interstitials and vacancies. Each of the point defect types exhibits distinct behaviours for formation and migration that make them the subject of intense and ongoing research. The development of new materials such as HEAs that could significantly alter irradiation defect dynamics to avert disastrous mechanical failure and increase service lifetimes of mechanical components is therefore one of great importance to the future of the energy sector.

In the projects described here, graph-based techniques were used to extract easily visualized and intuitive representations of defect details in HEAs from MD simulations. The first

analyzed and classified the states present in self-interstitial defect trajectories in a simulated FCC HEA using a graph-based fingerprinting technique. The second trained a GCN model to predict transition escape rates for each atom surrounding a vacancy defect and demonstrated the application for the model to form hybrid systems and use the learned dynamics to produce accelerated MD-compatible vacancy trajectories.

Bibliography

[1] C. Casert, I. Tamblyn, and S. Whitelam, "Learning stochastic dynamics and predicting emergent behavior using transformers," *arXiv preprint arXiv:2202.08708*, 2022.

[2] Y. Chang, I. Benlolo, C. Reimer, H. Zhang, D. Zhou, H. Choubisa, X. Li, W. Chen, P. Ou, I. Tamblyn, and E. H. Sargent, "Discovery of high-entropy alloy electrocatalysts using machine learning informed by quantum-inspired similarity analysis," in preparation.

[3] T. A. A. B. et al., "High-Entropy Alloys as a Discovery Platform for Electrocatalysis," *Joule*, vol. 3, no. 3, pp. 834–845, 2019.

[4] P. S. et al., "Deep Learning and Crystal Plasticity: A Preconditioning Approach for Accurate Orientation Evolution Prediction," *Computer Methods in Applied Mechanics and Engineering*, vol. 389, 2022.

[5] T. Xie and J. C. Grossman, "Crystal graph convolutional neural networks for an accurate and interpretable prediction of material properties," *Phys. Rev. Lett.*, vol. 120, p. 145301, Apr 2018. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.120.145301

[6] J. Damewood, J. Karaguesian, J. R. Lunger, A. R. Tan, M. Xie, J. Peng, and R. Gómez-Bombarelli, "Representations of materials for machine learning," 2023.

[7] C. C. et al., "A Critical Review of Machine Learning of Energy Materials," *Chem. Mater.*, vol. 31, no. 9, 2019.

[8] L. Cian, G. Lancioni, L. Zhang, M. Ianese, N. Novelli, G. Serra, and F. Maresca, "Atomistic graph neural networks for metals: Application to bcc iron," 2021.

[9] Y. Xia, L. Wu, and G. Wang, "Rapid evaluation method for anisotropic growth of ws2 monolayers by combining machine learning algorithms and kinetic monte carlo

simulation data," *Computational Materials Science*, vol. 184, p. 109922, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0927025620304134

[10] A. M. L. Scotti, "Interstitial diffusion of O, N, and C in α-Ti from first-principles: Analytical model and kinetic Monte Carlo simulations," *J. Chem. Phys.*, vol. 144, 2016.

[11] M. M. M. Bracconi, "Training set design for machine learning techniques applied to the approximation of computationally intensive first-principles kinetic models," *Chemical Engineering Journal*, vol. 400, 2020.

[12] T. Z. et al., "Tailoring nanoprecipitates for ultra-strong high-entropy alloys via machine learning and prestrain aging," *Journal of Materials Science I& Technology*, vol. 69, p. 156–167, 2021.

[13] W. H. et al., "Machine-learning phase prediction of high-entropy alloys," *Acta Materialia*, vol. 169, pp. 225–236, 2019.

[14] C. G. T. F. et al., "Neural evolution structure generation: High Entropy Alloys," *The Journal of Chemical Physics*, vol. 155, 2021.

[15] W.-M. Choi, Y. H. Jo, S. S. Sohn, S. Lee, and B.-J. Lee, "Understanding the physical metallurgy of the cocrfemnni high-entropy alloy: an atomistic simulation study," *Npj Computational Materials*, vol. 4, no. 1, p. 1, 2018.

[16] E. J. P. et al., "High-Entropy Alloys for Advanced Nuclear Applications," *Entropy*, vol. 23, 2021.

[17] Z. H. Aitken, V. Sorkin, and Y.-W. Zhang, "Atomistic modeling of nanoscale plasticity in high-entropy alloys," *Journal of Materials Research*, vol. 34, no. 9, p. 1509–1532, 2019.

[18] L. Nibbelink, "Simulating Vacancy Formation and Diffusion in NbMoTaW," *ProQuest Dissertations Publishing*, 2020.

[19] W. Yang and Y. Wu, "Effects of annealing on the micro-internal stress induced by interstitial defects in aluminum crystal by molecular dynamics simulations," *AIP Advances*, vol. 12, no. 2, p. 025226, 2022.

[20] S. A. Starikov, A. R. Kuznetsov, and V. V. Sagaradze, "Crowdion in deformed fcc metal. atomistic modeling," *Physics of metals and metallography*, vol. 122, no. 12, pp. 1207–1212, 2021.

[21] S. Bukkuru, U. Bhardwaj, K. S. Rao, A. D. P. Rao, M. Warrier, and M. C. Valsakumar, "Kinetics of self-interstitial migration in bcc and fcc transition metals," *Materials Research Express*, vol. 5, no. 3, pp. 35 513–, 2018.

[22] S. P. Fitzgerald, "Structure and dynamics of crowdion defects in bcc metals," *Journal of micromechanics and molecular physics*, vol. 3, no. 3n04, pp. 1 840 003–, 2018.

[23] D. R. Mason, A. E. Sand, and S. L. Dudarev, "Atomistic-object kinetic monte carlo simulations of irradiation damage in tungsten," *Modelling and simulation in materials science and engineering*, vol. 27, no. 5, pp. 55 003–, 2019.

[24] M. Posselt, F. Gao, and H. Bracht, "Correlation between self-diffusion in si and the migration mechanisms of vacancies and self-interstitials: An atomistic study," *Physical review. B*, vol. 78, no. 3, 2008.

[25] A. F. Voter, "Introduction to the kinetic monte carlo method," in *Radiation Effects in Solids*, ser. NATO Science Series. Dordrecht: Springer Netherlands, 2007, pp. 1–23.

[26] J.-W. Jang, B.-J. Lee, and J.-H. Hong, "Influence of cu, cr and c on the irradiation defect in fe: A molecular dynamics simulation study," *Journal of nuclear materials*, vol. 373, no. 1, pp. 28–38, 2008.

[27] A. Ervin and H. Xu, "Mesoscale simulations of radiation damage effects in materials: A seakmc perspective," *Computational Materials Science*, vol. 150, pp. 180–189, 2018.

[28] B. W. C. S. Becquart, "1.14 – kinetic monte carlo simulations of irradiation effects", comprehensive nuclear materials," in *Comprehensive Nuclear Materials, Five Volume Set*, R. J. Konings, Ed. Elsevier Science, 2012.

[29] L. Qian, H. Bao, R. Li, and Q. Peng, "Atomistic insights of a chemical complexity effect on the irradiation resistance of high entropy alloys," *Materials advances*, vol. 3, no. 3, pp. 168–1686, 2022.

[30] O. K. Orhan, M. Hendy, and M. Ponga, "Electronic effects on the radiation damage in high-entropy alloys," *Acta materialia*, vol. 244, pp. 118 511–, 2023.

[31] K. Nordlund, S. Zinkle, A. Sand, F. Granberg, R. Averback, R. Stoller, T. Suzudo, L. Malerba, F. Banhart, W. Weber, F. Willaime, S. Dudarev, and D. Simeone, "Primary radiation damage: A review of current understanding and models," *Journal of Nuclear Materials*, vol. 512, 10 2018.

[32] F. J. Domínguez-Gutiérrez, J. Byggmästar, K. Nordlund, F. Djurabekova, and U. von Toussaint, "Computational study of crystal defect formation in mo by a machine learning molecular dynamics potential," *Modelling and Simulation in Materials Science and Engineering*, vol. 29, no. 5, p. 055001, may 2021.

[33] A. C. D. Farkas, "Model interatomic potentials and lattice strain in a high-entropy alloy," *Journal of materials research*, vol. 33, pp. 3218–3225, 2018.

[34] W. Cai, J. Li, and S. Yip, "1.09-molecular dynamics," *Comprehensive nuclear materials*, pp. 249–265, 2012.

[35] K. Ryczko, J. T. Krogel, and I. Tamblyn, "Machine learning diffusion monte carlo energies," *Journal of Chemical Theory and Computation*, vol. 18, no. 12, pp. 7695–7701, nov 2022. [Online]. Available: https://doi.org/10.1021%2Facs.jctc.2c00483

[36] K. Ryczko, D. A. Strubbe, and I. Tamblyn, "Deep learning and density-functional theory," *Physical Review A*, vol. 100, no. 2, aug 2019. [Online]. Available: https://doi.org/10.1103%2Fphysreva.100.022512

[37] B. X. et al, "Revealing the crucial role of rough energy landscape on self-diffusion in high-entropy alloys based on machine learning and kinetic Monte Carlo," *Acta Materialia*, vol. 234, 2022.

[38] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe, "Weisfeiler and leman go neural: Higher-order graph neural networks," 2021.

[39] D. A. Scott, *Metallography and microstructure of ancient and historic metals*. Marina del Rey, CA: Getty Conservation Institute, 1991.

[40] V. Bennett, K. Bowman, and S. Wright, "Doe fundamentals handbook: Material science. volume 1," USDOE, Washington, DC, Tech. Rep. DOE-HDBK–1017/1-93, 1993. [Online]. Available: https://inis.iaea.org/search/searchsinglerecord.aspx?recordsFor=SingleRecord&RN=24058518

[41] G. S. Rohrer, "Crystal structures," in *Structure and Bonding in Crystalline Materials*. United Kingdom: Cambridge University Press, 2001, pp. 135–204.

[42] F. Lamelas, "Periodicity and lattices," in *Encyclopedia of Condensed Matter Physics*, F. Bassani, G. L. Liedl, and P. Wyder, Eds. Oxford: Elsevier, 2005, pp. 238–246. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B0123694019004198

[43] G. Iadonisi, G. Cantele, and M. L. Chiofalo, *Introduction to Solid State Physics and Crystalline Nanostructures*, ser. UNITEXT for Physics. Milano: Springer Milan, 2014.

[44] A. Silva and J. van Wezel, "The simple-cubic structure of elemental polonium and its relation to combined charge and orbital order in other elemental chalcogens," *SciPost physics*, vol. 4, no. 6, pp. 028–, 2018.

[45] P. Flowers, K. Theopold, R. Langley, and W. R. Robinson, "Lattice Structures in Crystalline Solids," in *Chemistry 2e*. OpenStax, 02 2019.

[46] Brandon, "Face-Centered Cubic (FCC) Unit Cell," Materials Science engineering Student, Blog, 2022. [Online]. Available: https://msestudent.com/face-centered-cubic-fcc-unit-cell/

[47] M. J. Mehl, D. Hicks, C. Toher, O. Levy, R. M. Hanson, G. Hart, and S. Curtarolo, "The aflow library of crystallographic prototypes," 2016. [Online]. Available: https://arxiv.org/abs/1607.02532

[48] D. Lu, Q. Jiang, X. Ma, Q. Zhang, X. Fu, and L. Fan, "Defect-related etch pits on crystals and their utilization," *Crystals (Basel)*, vol. 12, no. 11, pp. 1549–, 2022.

[49] G. Gottstein, M. Goerdeler, and G. Prasad, "Mechanical properties: Plastic behavior," in *Encyclopedia of Condensed Matter Physics*, F. Bassani, G. L. Liedl,

and P. Wyder, Eds. Oxford: Elsevier, 2005, pp. 298–305. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B0123694019005696

[50] S. Thomesen, O. S. Hopperstad, and T. Børvik, "Anisotropic plasticity and fracture of three 6000-series aluminum alloys," *Metals (Basel)*, vol. 11, no. 4, pp. 557–, 2021.

[51] W. O. Soboyejo, *Mechanical properties of engineered materials*, ser. Mechanical engineering ; 152. New York: Marcel Dekker, 2003.

[52] M. Baricco, "Alloys: Overview," in *Encyclopedia of Condensed Matter Physics*, F. Bassani, G. L. Liedl, and P. Wyder, Eds. Oxford: Elsevier, 2005, pp. 57–64. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B0123694019005313

[53] W. Hosford, "Alloys: Copper," in *Encyclopedia of Condensed Matter Physics*, F. Bassani, G. L. Liedl, and P. Wyder, Eds. Oxford: Elsevier, 2005, pp. 24–45. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B0123694019005337

[54] M. Epler, "Structures by Precipitation from Solid Solution," in *Metallography and Microstructures*. ASM International, 12 2004. [Online]. Available: https://doi.org/10.31399/asm.hb.v09.a0003731

[55] D. Han, J.-X. He, X.-J. Guan, Y.-J. Zhang, and X.-W. Li, "Impact of short-range clustering on the multistage work-hardening behavior in cu–ni alloys," *Metals*, vol. 9, no. 2, 2019. [Online]. Available: https://www.mdpi.com/2075-4701/9/2/151

[56] Y. Sun and S. Dai, "High-entropy materials for catalysis: A new frontier," *Science advances*, vol. 7, no. 20, 2021.

[57] D. Connétable, Éric Andrieu, and D. Monceau, "First-principles nickel database: Energetics of impurities and defects," *Computational Materials Science*, vol. 101, pp. 77–87, 2015.

[58] T. S. Tetsuo Mohri, "Solid solution hardening by impurities," in *Impurities in Engineering Materials*, F. Bassani, G. L. Liedl, and P. Wyder, Eds. Routledge, 1999, pp. 259–299.

[59] J. K. Pedersen, T. A. A. Batchelor, A. Bagger, and J. Rossmeisl, "High-entropy alloys as catalysts for the co2 and co reduction reactions," *ACS catalysis*, vol. 10, no. 3, pp. 2169–2176, 2020.

[60] G. S. Was, *Chapter 4 Point Defect Formation and Diffusion*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 155–190. [Online]. Available: https://doi.org/10.1007/978-3-540-49472-0_4

[61] W. G. Wolfer, "1.01 – fundamental properties of defects in metals," in *Comprehensive Nuclear Materials, Five Volume Set*, R. J. Konings, Ed. Elsevier Science, 2012.

[62] R. Smallman and A. Ngan, "Chapter 6 - point defect behaviour," in *Modern Physical Metallurgy*, 8th ed., R. Smallman and A. Ngan, Eds. Oxford: Butterworth-Heinemann, 2014, pp. 251–285. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780080982045000067

[63] J. Derby, "Crystal growth, bulk: Theory and models," in *Encyclopedia of Condensed Matter Physics*, F. Bassani, G. L. Liedl, and P. Wyder, Eds. Oxford: Elsevier, 2005, pp. 274–282. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B0123694019004150

[64] G. Müller and J. Friedrich, "Crystal growth, bulk: Methods," in *Encyclopedia of Condensed Matter Physics*, F. Bassani, G. L. Liedl, and P. Wyder, Eds. Oxford: Elsevier, 2005, pp. 262–274. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B0123694019004162

[65] W. Fowler, "Point defects," in *Encyclopedia of Condensed Matter Physics*, F. Bassani, G. L. Liedl, and P. Wyder, Eds. Oxford: Elsevier, 2005, pp. 318–323. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B0123694019004125

[66] D. Sun, Y. Gao, J. Xue, and J. Zhao, "Defect stability and electronic structure of doped β-ga2o3: A comprehensive ab initio study," *Journal of alloys and compounds*, vol. 794, pp. 374–384, 2019.

[67] B. Uberuaga, L. Vernon, E. Martinez, and A. Voter, "The relationship between grain boundary structure, defect mobility, and grain boundary sink efficiency," *Scientific*

reports, vol. 5, 03 2015.

[68] R. A. Konchakov, A. S. Makarov, G. V. Afonin, M. A. Kretova, N. P. Kobelev, and V. A. Khonik, "Relation between the shear and dilatational elastic energies of interstitial defects in metallic crystals," *JETP letters*, vol. 109, no. 7, pp. 460–464, 2019.

[69] W. Zhou, Y. Li, L. Huang, Z. Zeng, and X. Ju, "Dynamical behaviors of self-interstitial atoms in tungsten," *Journal of Nuclear Materials*, vol. 437, no. 1, pp. 438–444, 2013.

[70] E. Korznikova, V. Shunaev, I. Shepelev, O. Glukhova, and S. Dmitriev, "Ab initio study of the propagation of a supersonic 2-crowdion in fcc al," *Computational Materials Science*, vol. 204, p. 111125, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0927025621007904

[71] A. P. Thompson, H. M. Aktulga, R. Berger, D. S. Bolintineanu, W. M. Brown, P. S. Crozier, P. J. in 't Veld, A. Kohlmeyer, S. G. Moore, T. D. Nguyen, R. Shan, M. J. Stevens, J. Tranchida, C. Trott, and S. J. Plimpton, "Lammps - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales," *Computer Physics Communications*, vol. 271, p. 108171, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0010465521002836

[72] E. Vincent, C. Becquart, and C. Domain, "Ab initio calculations of self-interstitial interaction and migration with solute atoms in bcc fe," *Journal of nuclear materials*, vol. 359, no. 3, pp. 227–237, 2006.

[73] X. Zhang, S. V. Divinski, and B. Grabowski, "Ab initio prediction of vacancy energetics in hcp al-hf-sc-ti-zr high entropy alloys and the subsystems," *Acta Materialia*, vol. 227, p. 117677, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1359645422000647

[74] L. Mansur, "Theory and experimental background on dimensional changes in irradiated alloys," *Journal of Nuclear Materials*, vol. 216, pp. 97–123, 1994.

[75] Y. C. et al., "Void Swelling and Microstructure of Austenitic Stainless Steels Irradiated in the BOR-60 Reactor," *United States Nuclear Regulatory Commission*, 2012.

[76] A. V. et al., "Non-random walk diffusion enhances the sink strength of semicoherent interfaces," *Nature Communications*, vol. 7, 2016.

[77] M. Tschopp, M. Horstemeyer, F. Gao, X. Sun, and M. Khaleel, "Energetic driving force for preferential binding of self-interstitial atoms to fe grain boundaries over vacancies," *Scripta materialia*, vol. 64, no. 9, pp. 908–911, 2011.

[78] A. E. Stuchbery and E. Bezakova, "Thermal-spike lifetime from picosecond-duration preequilibrium effects in hyperfine magnetic fields following ion implantation," *Physical review letters*, vol. 82, no. 18, pp. 3637–3640, 1999.

[79] G. S. Was, "The damage cascade," in *Fundamentals of Radiation Materials Science*. Springer, 2016.

[80] R. Smallman and A. Ngan, "Chapter 6 - point defect behaviour," in *Modern Physical Metallurgy*, 8th ed., R. Smallman and A. Ngan, Eds. Oxford: Butterworth-Heinemann, 2014, pp. 251–285. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780080982045000067

[81] S. qin XIA, Z. WANG, T. fei YANG, and Y. ZHANG, "Irradiation behavior in high entropy alloys," *Journal of Iron and Steel Research, International*, vol. 22, no. 10, pp. 879–884, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1006706X15300844

[82] V. Bennett, K. Bowman, and S. Wright, "Doe fundamentals handbook: Material science. volume 2," USDOE, Washington, DC, Tech. Rep. DOE-HDBK–1017/2-93, 1993. [Online]. Available: https://inis.iaea.org/search/searchsinglerecord.aspx?recordsFor=SingleRecord&RN=24058517

[83] X. Xiao, "Fundamental mechanisms for irradiation-hardening and embrittlement: A review," *Metals*, vol. 9, no. 10, p. 1132, Oct 2019. [Online]. Available: http://dx.doi.org/10.3390/met9101132

[84] L. K. Mansur, "Void swelling in metals and alloys under irradiation: An assessment of the theory," *Nuclear technology*, vol. 40, no. 1, pp. 5–34, 1978.

[85] A. Hojná, "Overview of intergranular fracture of neutron irradiated austenitic stainless steels," *Metals*, vol. 7, no. 10, p. 392, 2017.

[86] W. Hoffelner, "Damage assessment in structural metallic materials for advanced nuclear plants," *Journal of materials science*, vol. 45, no. 9, pp. 2247–2257, 2010.

[87] M. V. et al., "High-entropy alloys by mechanical alloying: A review," *Journal of Materials Research*, vol. 34, p. 664–686, 2019.

[88] D. Miracle and O. Senkov, "A critical review of high entropy alloys and related concepts," *Acta Materialia*, vol. 122, pp. 448–511, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1359645416306759

[89] A. K. et al., "Short communication: 'Low activation, refractory, high entropy alloys for nuclear applications'," *Journal of Nuclear Materials*, vol. 526, p. 156–167, 2019.

[90] Y. J. Zhou and Y. Zhang, "Solid solution formation criteria for high entropy alloys," *Materials science forum*, vol. 561-565, pp. 1337–1339, 2007.

[91] P. Barron, A. Carruthers, J. Fellowes, N. Jones, H. Dawson, and E. Pickering, "Towards v-based high-entropy alloys for nuclear fusion applications," *Scripta Materialia*, vol. 176, pp. 12–16, 2020.

[92] Z. F. et al., "A high-entropy alloy with hierarchical nanoprecipitates and ultrahigh strength," *Science Advances*, vol. 4, no. 10, 2018.

[93] M. C. Gao, J.-W. Yeh, P. K. Liaw, and Y. Zhang, "Potential applications and prospects," in *High-Entropy Alloys*. Switzerland: Springer International Publishing AG, 2016, pp. 493–512.

[94] Q. Xu, H. Guan, Z. Zhong, S. Huang, and J. Zhao, "Irradiation resistance mechanism of the cocrfemnni equiatomic high-entropy alloy," *Scientific reports*, vol. 11, no. 1, pp. 608–608, 2021.

[95] E. Lu, J. Zhao, I. Makkonen, K. Mizohata, Z. Li, M. Hua, F. Djurabekova, and F. Tuomisto, "Enhancement of vacancy diffusion by c and n interstitials in the equiatomic femnnicocr high entropy alloy," *Acta Materialia*, vol. 215, p.

117093, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1359645421004730

[96] C. Lu, L. Niu, N. Chen, K. Jin, T. Yang, P. Xiu, Y. Zhang, F. Gao, H. Bei, S. Shi, M.-R. He, I. M. Robertson, W. J. Weber, and L. Wang, "Enhancing radiation tolerance by controlling defect mobility and migration pathways in multicomponent single-phase alloys," *Nature communications*, vol. 7, no. 1, pp. 13 564–13 564, 2016.

[97] M. C. Gao, J.-W. Yeh, P. K. Liaw, and Y. Zhang, "Physical metallurgy," in *High-Entropy Alloys*. Switzerland: Springer International Publishing AG, 2016, pp. 51–113.

[98] T. Schlick, *Molecular Dynamics: Basics*. New York, NY: Springer New York, 2010, pp. 425–461. [Online]. Available: https://doi.org/10.1007/978-1-4419-6351-2_13

[99] A. C. Rencher and G. B. Schaalje, *Linear Models in Statistics*. Newy York: John Wiley & Sons, Incorporated, 2008.

[100] LibreText, "The Monte Carlo Simulation Method," LibreTexts Statistics, 2020. [Online]. Available: https://stats.libretexts.org/Bookshelves/Computing_and_Modeling/RTG%3A_Simulating_High_Dimensional_Data/The_Monte_Carlo_Simulation_Method

[101] ——, "The Monte Carlo Simulation V2," LibreTexts Statistics, 2020. [Online]. Available: https://stats.libretexts.org/Bookshelves/Computing_and_Modeling/RTG%3A_Simulating_High_Dimensional_Data/The_Monte_Carlo_Simulation_V2

[102] M. Andersen, C. Panosetti, and K. Reuter, "A practical guide to surface kinetic monte carlo simulations," *Frontiers in Chemistry*, vol. 7, 2019.

[103] D. Galvin, "Discrete Mathematics, Spring 2009 Graph theory notation," University of Notre Dame. [Online]. Available: https://www3.nd.edu/~dgalvin1/60610/60610_S09/60610graphnotation.pdf

[104] M. Hernandez, A. Zaribafiyan, M. Aramon, and M. Naghibi, "A novel graph-based approach for determining molecular similarity," 2016. [Online]. Available: https://arxiv.org/abs/1601.06693

[105] L. A. Zager and G. C. Verghese, "Graph similarity scoring and matching," *Applied mathematics letters*, vol. 21, no. 1, pp. 86–94, 2008.

[106] P.-A. Champin and C. Solnon, "Measuring the similarity of labeled graphs," in *Case-Based Reasoning Research and Development*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 80–95.

[107] Z. Li, X. Jian, X. Lian, and L. Chen, "An efficient probabilistic approach for graph similarity search," in *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 2018, pp. 533–544.

[108] R. Pan, Z. Wang, Y. Wei, H. Gao, G. Ou, C. C. Cao, J. Xu, T. Xu, and W. Chen, "Towards efficient visual simplification of computational graphs in deep neural networks," *IEEE Transactions on Visualization and Computer Graphics*, pp. 1–14, 2022.

[109] C. Olah, "Calculus on computational graphs: Backpropagation," Colah's blog, August 31 2015. [Online]. Available: http://colah.github.io/posts/2015-08-Backprop/

[110] M. Taboga, "Conditional models," StatLect. [Online]. Available: https://www.statlect.com/fundamentals-of-statistics/conditional-models

[111] ——, "Predictive models," StatLect. [Online]. Available: https://www.statlect.com/machine-learning/predictive-model

[112] ——, "Statistical model," StatLect. [Online]. Available: https://www.statlect.com/glossary/statistical-model

[113] ——, "Linear regression model," StatLect. [Online]. Available: https://www.statlect.com/fundamentals-of-statistics/linear-regression

[114] ——, "Logistic Classification Model," StatLect. [Online]. Available: https://www.statlect.com/fundamentals-of-statistics/logistic-classification-model

[115] ——, "Classification Models," StatLect. [Online]. Available: https://www.statlect.com/fundamentals-of-statistics/classification-models

[116] ——, "Multinoulli distribution," StatLect. [Online]. Available: https://www.statlect.com/probability-distributions/multinoulli-distribution

[117] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, ser. Springer series in statistics. Springer, 2009. [Online]. Available: https://books.google.ca/books?id=eBSgoAEACAAJ

[118] M. Taboga, "Maximum likelihood estimation," StatLect, 2021. [Online]. Available: https://www.statlect.com/fundamentals-of-statistics/maximum-likelihood

[119] ——, "Log-likelihood," StatLect, 2021. [Online]. Available: https://www.statlect.com/glossary/log-likelihood

[120] D. Montgomery, G. Runger, and N. Hubele, *Engineering Statistics*, 5th ed. John Wiley & Sons, Incorporated, 2010. [Online]. Available: https://books.google.ca/books?id=BcwbAAAAQBAJ

[121] R. T. McGibbon and V. S. Pande, "Efficient maximum likelihood parameterization of continuous-time markov processes," *The Journal of chemical physics*, vol. 143, no. 3, pp. 034 109–034 109, 2015.

[122] M. Taboga, "Poisson distribution - Maximum Likelihood Estimation," StatLect, 2021. [Online]. Available: https://www.statlect.com/fundamentals-of-statistics/Poisson-distribution-maximum-likelihood

[123] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org

[124] M. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2015. [Online]. Available: https://books.google.ca/books?id=STDBswEACAAJ

[125] B. Fortuner, "Neural networks: Concepts," ML Glossary, 2017. [Online]. Available: http://ml-cheatsheet.readthedocs.io/en/latest/nn_concepts.html

[126] Y. Goldberg, "A primer on neural network models for natural language processing," *The Journal of artificial intelligence research*, vol. 57, pp. 345–420, 2016.

[127] C. Beeler, "A3md ml bootcamp: Neural networks," Presentation, 2020, accessed: 2023-09-28.

[128] M. Zhou, "PyTorch-Tutorial," 2020. [Online]. Available: https://github.com/MorvanZhou/PyTorch-Tutorial/blob/master/tutorial-contents/203_activation.py

[129] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014. [Online]. Available: https://arxiv.org/abs/1412.6980

[130] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," 2016. [Online]. Available: https://arxiv.org/abs/1603.07285

[131] D. Foster, *Generative deep learning : teaching machines to paint, write, compose, and play*, 1st ed. Sebastopol, CA: O'Reilly Media, Inc., 2019.

[132] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei, "Deformable convolutional networks," in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 764–773.

[133] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666651021000012

[134] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2016. [Online]. Available: https://arxiv.org/abs/1609.02907

[135] D. Grattarola, "A practical introduction to gnns - part 1," Daniele Grattarola, blog, March 3 2021. [Online]. Available: https://danielegrattarola.github.io/posts/2021-03-03/gnn-lecture-part-1.html

[136] ——, "A practical introduction to gnns - part 2," Daniele Grattarola, blog, March 12 2021. [Online]. Available: https://danielegrattarola.github.io/posts/2021-03-12/gnn-lecture-part-2.html

[137] P. Rodríguez, M. A. Bautista, J. Gonzàlez, and S. Escalera, "Beyond one-hot encoding: Lower dimensional target embedding," *Image and vision computing*, vol. 75, pp. 21–31, 2018.

[138] L. Messina, T. Schuler, M. Nastar, M.-C. Marinica, and P. Olsson, "Solute diffusion by self-interstitial defects and radiation-induced segregation in ferritic fe–x (x=cr, cu, mn, ni, p, si) dilute alloys," *Acta Materialia*, vol. 191, pp. 166–185, 2020.

[139] S. Zinkle, "1.03-radiation-induced effects on microstructure," *Comprehensive nuclear materials*, vol. 1, pp. 65–98, 2012.

[140] S. Han, L. A. Zepeda-Ruiz, G. J. Ackland, R. Car, and D. J. Srolovitz, "Self-interstitials in v and mo," *Phys. Rev. B*, vol. 66, p. 220101, Dec 2002. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevB.66.220101

[141] U. von Toussaint, F. Domínguez-Gutiérrez, M. Compostella, and M. Rampp, "Favad: A software workflow for characterization and visualizing of defects in crystalline structures," *Computer Physics Communications*, vol. 262, p. 107816, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0010465520304094

[142] A. Zunger, "Inverse design in search of materials with target functionalities," *Nature Reviews Chemistry*, vol. 2, no. 4, pp. 1–16, 2018.

[143] M. Widom, W. P. Huhn, S. Maiti, and W. Steurer, "Hybrid monte carlo/molecular dynamics simulation of a refractory metal high entropy alloy," *Metallurgical and Materials Transactions A*, vol. 45, pp. 196–200, 2014.

[144] J. Farmer, F. Kanwal, N. Nikulsin, M. C. B. Tsilimigras, and D. J. Jacobs, "Statistical measures to quantify similarity between molecular dynamics simulation trajectories," *Entropy*, vol. 19, no. 12, 2017. [Online]. Available: https://www.mdpi.com/1099-4300/19/12/646

[145] LAMMPS, "minimize command," LAMMPS Documentation, 2022. [Online]. Available: https://docs.lammps.org/minimize.html

[146] OVITO, "About," OVITO. [Online]. Available: https://www.ovito.org/about/

[147] A. Stukowski, "Visualization and analysis of atomistic simulation data with OVITO – the Open Visualization Tool," *Modelling Simul. Mater. Sci. Eng.*, vol. 18, 2010.

[148] M. Nastar and F. Soisson, "1.18-radiation-induced segregation," *Comprehensive nuclear materials*, pp. 471–496, 2012.

[149] R. J. Zamora, D. Perez, and A. F. Voter, "Speculation and replication in temperature accelerated dynamics," *Journal of materials research*, vol. 33, no. 7, pp. 823–834, 2018.