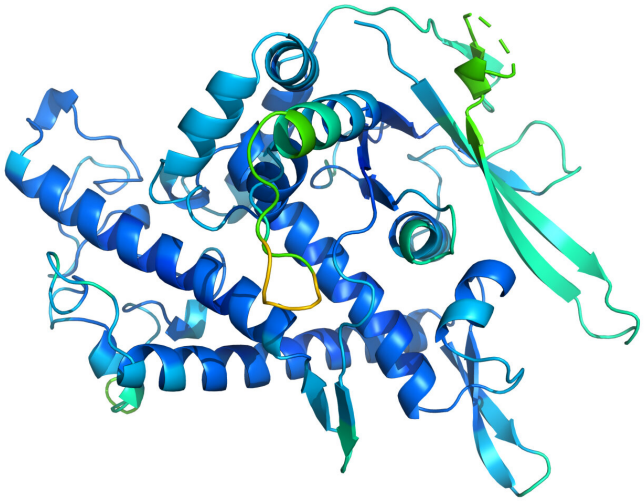
CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG



Detecting Metastable States in Proteins using E(3) Equivariant VAMPnets

A study using VAMPnets trained by E(3) equivariant neural networks to detect metastable states in proteins

Master's thesis in Computer Science and Engineering

SARA ARNESEN AND DAVID NORDSTRÖM

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2023

Master's thesis 2023

Detecting Metastable States in Proteins using E(3) Equivariant VAMPnets

A study using VAMPnets trained by E(3) equivariant neural networks to detect metastable states in proteins

SARA ARNESEN AND DAVID NORDSTRÖM

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2023

Detecting Metastable States in Proteins using E(3) Equivariant VAMPnets
A study using VAMPnets trained by E(3) equivariant neural networks to detect metastable states in proteins
SARA ARNESEN AND DAVID NORDSTRÖM

Supervisor: Simon Olsson, Computer Science and Engineering.
Examiner: Devdatt Dubhashi, Computer Science and Engineering

Master's Thesis 2023
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Illustrative Image of Protein Structures

Typeset in LaTeX
Gothenburg, Sweden 2023

Detecting Metastable States in Proteins using E(3) Equivariant VAMPnets
A study using VAMPnets trained by E(3) equivariant neural networks to detect metastable states in proteins

SARA ARNESEN
DAVID NORDSTRÖM
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

As proteins fold, they encounter intermediary conformations, often denoted metastable states, that are vital to deciphering diseases related to malfunctions in conformational changes. To detect these metastable states, a deep learning framework using the variational approach for Markov processes (VAMP) has been proposed, dubbed VAMPnets. In this master's thesis, we improve the training of VAMPnets through the use of E(3) equivariant neural networks. These networks incorporate the symmetries of Euclidean space, facilitating faster and more data-efficient learning. To study the effectiveness of these networks, we benchmark two different equivariant Transformer architectures and an equivariant convolutional network against both a simple and an invariant multilayered perceptron. The models are evaluated on molecular dynamics trajectories of alanine dipeptide and protein folding datasets.

The use of E(3) equivariant neural networks in training VAMPnets is shown to significantly improve the prediction accuracy on random downsampled data. Using only 1% of the dataset, the equivariant Transformer achieves almost twice the VAMP-2 score as the benchmarks. Furthermore, the model exhibits improved robustness. With only 20% data remaining, the model scores on par with the complete dataset. On average, the model requires significantly fewer backward passes, converging more than twice as fast as the benchmark models, showing enhanced data efficiency. Furthermore, the results highlight the significant computational burden that equivariant neural networks pose, especially for larger molecules, proving almost 1,000 times slower on the protein folding dataset. Finally, we propose a novel algorithm for detecting the number of metastable states of a molecule using the VAMP-2 score and provide estimates for the 12 proteins in the protein folding dataset.

Keywords: Computer Science, Engineering, Project, Thesis, Deep Learning, Protein Structures, Equivariant Neural Networks, Molecular Dynamics, Computational Biology, VAMPnets, Transformers

Acknowledgements

This thesis was completed during the spring of 2023 as a final step towards a master's degree at Chalmers University of Technology. The thesis was conducted at the Department of Computer Science & Engineering under the Division of Data Science & AI.

First and foremost, we would like to express our utmost gratitude to Simon Olsson, Assistant Professor within the Data Science & AI division at Chalmers University of Technology. We are immensely grateful for his contribution as a supervisor, having helped us outline the research project and providing guidance throughout the process.

Furthermore, we would like to convey our utmost appreciation to all researchers in the field of VAMPnets and E(3) equivariant neural networks. Without your invaluable knowledge and research, we would not have been able to complete this thesis. In particular, we want to thank DE Shaw Research for providing the simulation data for the fast folding protein data [33], Mario Geiger and Tess Smidt for their work on e3nn [14], Yi-Lun Liao and Tess Smidt for their development of the Equiformer [32] and Mardt et al., for providing us with the VAMPnets framework [35]. In addition, we would like to extend our utmost gratitude to the National Academic Infrastructure for Supercomputing in Sweden (NAISS) for approving our project 2023/22-392 with the title "*Detecting meta-stable states in molecular simulations with geometric deep learning*" and granting us the compute needed to fulfill the research purpose.

Lastly, we would like to convey our immense appreciation to professors and fellow students at Chalmers University of Technology, from which we have had the opportunity to learn from during the past five years.

Sara Arnesen
David Nordström
Gothenburg, 2023-06-10

Contents

List of Figures

List of Tables

Acronyms

C Carbon

DNN Deep Neural Network

DP Dot Product

DTP Depth-wise Tensor Product

E(3) Euclidean group in three dimensions

ELU Exponential Linear Unit

CK Chapman-Kolmogorov

GA Graph Attention

GCNN Graph Convolutional Neural Network

GNN Graph Neural Network

H Hydrogen

HDF5 Hierarchical Data Format 5 (file type)

LN Layer Normalization

MD Molecular Dynamics

MLP Multilayer Perceptron

MM Molecular Mechanics

MSM Markov State Models

N Nitrogen

O Oxygen

QM Quantum Mechanics

PDB Protein Data Bank (file type)

RELU Rectified Linear Unit

SiLU Sigmoid-Weighted Linear Units

VAMP Variational Approach for Markov Processes

1

Introduction

As proteins fold, they encounter intermediary conformations. These intermediary states denoted metastable states, are vital for understanding the complexity of biological processes and deciphering diseases related to malfunctions in conformational changes [17, 25]. As such, it is of great interest to be able to detect these metastable states for a given protein structure. Using Molecular Dynamics (MD) data, which simulates conformational trajectories, state-of-the-art models have aimed to estimate Markov State Models (MSM) leveraging dimensionality reduction and clustering to detect the metastable states [25]. However, this approach requires substantial modeling expertise and is very sensitive to extensive modeling errors [35].

With the rapid development of Deep Neural Networks (DNNs) in recent years, due to their successful application to image and natural language processing, researchers have searched for other application areas, including the prediction of a protein's native structure and exploration of its energy landscape [25]. For instance, Wu et al. [69] and Mardt et al. [35] proposed a deep generative MSM based on a DNN to learn MD and sample conformations from conformation space [25]. VAMPnets, the DNN architecture proposed by Mardt et al. [35], proved to perform equally or better than state-of-the-art Markov modeling methods and comes as a single end-to-end framework, reducing the model complexity and corresponding errors. However, the number of metastable states for the specific protein or peptide must be known in order to train the DNN. To automatically select the number of metastable states from MD data is still an open research problem [25].

Translational equivariance has become the state-of-the-art of image recognition through Convolutional Neural Networks (CNNs) such as the ImageNet [54] and ResNet [20] architectures. Generalizing these results to irregular graph the field of Geometric Deep Learning emerged through the success of Graph Convolutional Neural Networks (GCNNs) such as ChebNet [7], EdgeConv [64], Graph U-Net [13] and GraphSAGE [18]. Recent research has shown the that state-of-the-art results on atomistic systems can be achieved by incorporating E(3) equivariance into these networks in architectures such as the Equiformer [32], NequIP [2] and Tensor Field Networks [60]. Equivariance is a mathematical property detailed in group and representation theory. If a function or transformation is equivariant with respect to a given symmetry group it means that applying the transformation to both the input and the output of the function yields the same result. Invariance is a special case of equivariance. While equivariance describes the behavior of a function under

transformations, invariance refers to a property of a function or an object that remains unchanged or "invariant" under transformations of the symmetry group. The symmetry group that we refer to when labeling our models as equivariant is the E(3) symmetry group, also known as the Euclidean group in three dimensions. This group consists of the rigid transformations that preserve the Euclidean geometry of three-dimensional space. It consists of all possible translations, rotations, and reflections in three dimensions [72].

In this thesis, we unite the two aforementioned subjects by implementing VAMPnets, a neural network architecture framework that can be trained by maximizing the VAMP-2 score, using equivariant neural networks to improve prediction accuracy, robustness, and data efficiency in comparison with earlier implementations. Previous research has found that switching from invariant to equivariant models, even within the same framework, improves accuracy in training tasks [2, 40].

The equivariant architectures used in this thesis are a GCNN inspired by the Simple Network as proposed by Geiger and Smidt [14], a Dot Product (DP) Equiformer and a Graph Attention (GA) Equiformer as suggested by Liao and Smidt [32], which build on the transformer proposed by Vaswani et al. [61]. We compare two benchmarks to investigate the effectiveness of equivariant models. Firstly, an invariant multilayered perceptron (MLP) that utilizes the pairwise distances of the atoms to guarantee E(3) invariance. Secondly, a regular MLP, inspired by Mardt et al. [35], trained on the positional coordinates of the atoms.

Additionally, we propose a novel algorithm to determine the number of metastable states from the MD data leveraging the VAMP-2 score and reducing the need for expert judgment. An estimation of the number of metastable states for the 12 fast-folding proteins provided by DE Shaw Research [33] is provided leveraging this algorithm.

The resulting research questions we aim to answer are phrased as follows:

1. *Do the molecular physical symmetries encoded by E(3) Equivariant Networks improve prediction accuracy, robustness, or data efficiency in VAMPnets training compared to an MLP encoding?*
2. *Do VAMPnets built on E(3) Equivariant Networks demonstrate improved accuracy in detecting metastable states in protein folding datasets compared to state-of- the-art implementations?*

2

Theory

This chapter presents the theoretical frameworks relevant to this thesis, beginning with the protein folding process and the biological foundations. Next, the mathematical foundations are derived through the VAMP-score and the VAMPnets framework. Finally, the foundational aspects of the neural networks are examined by introducing equivariance and group theory as well as the Transformer and GCNN.

2.1 Protein Folding

Proteins are essential building blocks for all life, and understanding their structure can provide a mechanistic comprehension of their functionality [25, 26]. The functional properties of proteins are contingent upon their three-dimensional biomolecular structure, which are known to undergo reversible transitions between different conformations [25]. Prior to the release of AlphaFold2, the extent of the structural coverage of proteins in computational science was bottlenecked by processes lasting months to years to determine a single protein structure. For over a half-century, the prediction of the three-dimensional conformation that a protein will assume based on its amino acid sequence, also known as the structure prediction component of the "protein folding problem"[9], remained a significant and unresolved research challenge [1].

By leveraging multi-sequence alignments into the design of a deep learning algorithm, AlphaFold2 exemplified that a high prediction accuracy could be achieved in the CASP14 challenge [26]. The effectiveness of AlphaFold2 is due to its use of attention mechanisms, which enable it to incorporate a variety of data sources, including evolutionary and biophysical data. The algorithm demonstrates a unique framework incorporating an equivariant attention architecture using a key component denoted Evoformer, viewing protein structure prediction as a 3D graph inference problem. Residues positioned close to one another decide the edges of the graph. The components of the pair representation encode pertinent information about how the residues are related to one another [26]. Bearing some resemblance to the network architectures used in this thesis to detect the metastable states of proteins.

The development of AlphaFold2 made it possible to predict a protein's native structure only from its amino acid sequence [26]. This method focuses mainly on the protein-folding problem [9], which aims at the conformation of the lowest free energy. Nevertheless, detecting the intermediary conformations in protein folding, also

called metastable states, are still in need of further investigation [25].

Levinthal's paradox asserts that the folding process could not be random. Even for a short peptide, protein folding would take longer than the estimated age of the universe if all intermediate folding conformations were equally probable. However, experimentally, protein folding happens in a matter of seconds or less. Instead, as a result of local interactions during the protein folding process, the protein enters various stable conformations on its evolution to the folded state [73].

Depiction of the structural and functional properties of metastable proteins is not only required to understand the complexity of biological processes, including protein folding patterns but also to comprehend mechanisms of anomalous aggregation of different proteins [17] as well as deciphering diseases related to malfunctions in conformational changes [25]. Such diseases include Alzheimer's disease, Mad cow disease, Huntington's disease, and Parkinson's disease. Consequently, detecting and enumerating these metastable protein structures allow us to target proteins more specifically and directly [17].

2.1.1 Metastable States

The protein folding process comprises discrete steps that stabilize the protein in different conformations [17]. The free energy landscape of the conformational space is characterized by a rugged terrain containing several high-energy barriers partitioning the conformational space into a set of low-energy wells, denoted metastable states [25]. A metastable state of a protein refers to a kinetically trapped structure with a local free energy minimum. This local minimum is separated from the global free energy minimum, corresponding to the final, stable conformation of the protein [17].

Energy barriers give rise to the short-lived metastable states, separating its energy from the energy minima of alternative conformations [25]. In the folding process, the protein molecule must overcome energy barriers to reach its final conformation [17]. Energy is provided in the form of heat, usually supplied by the collision with water molecules in the protein's surrounding environment, often referred to as the canonical ensemble, and enables the transition from one folded protein conformation to another through energy jumps [17].

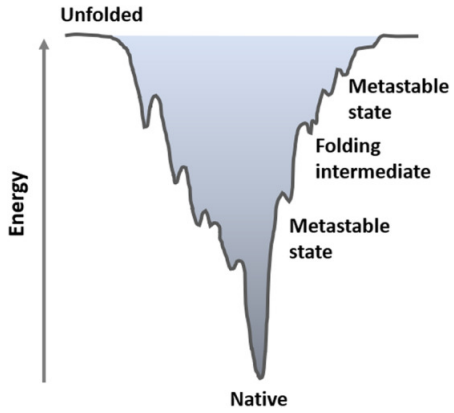


Figure 2.1: Illustration of the energy landscape of protein folding, comprising various folded structures that are formed through a series of stages, including the formation of intermediate conformations such as metastable states. Note: x-axis illustrates the conformation space.
Image Source: [17]

To illustrate the possible conformations, or residues, of the amino acids of alanine dipeptide, a molecule used later in this study, torsion angles ϕ and ψ are commonly used in what is referred to as a Ramachandran plot [49]. There are global geometrical constraints on the shape, not all shapes are obtained as a folded chain of amino acids [49]. A graphical illustration of the possible angles is obtained by depicting the torsion angles on a two-dimensional grid.

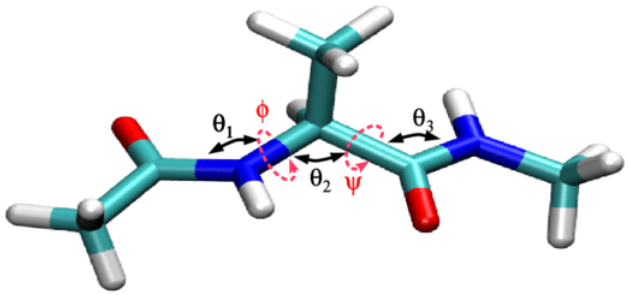


Figure 2.2: Illustrating the torsion angles ϕ and ψ by the example of alanine dipeptide.
Image Source: [27]

By situating each planar peptide bond in relation to the two adjacent planar peptide bonds, each residue in a peptide's torsional angles defines the geometry of its attachment to its two adjacent residues. Hence the torsional angles define the conformation of the residues and the peptide. Because of steric hindrance, certain angle combinations, and thus residue conformations, are impossible.

The dense regions in a Ramachandran plot detail the metastable states of the peptide [55]. In the case of alanine dipeptide, there are six such states, not equally densely populated. The metastable states of alanine dipeptide are illustrated in Figure 2.3.

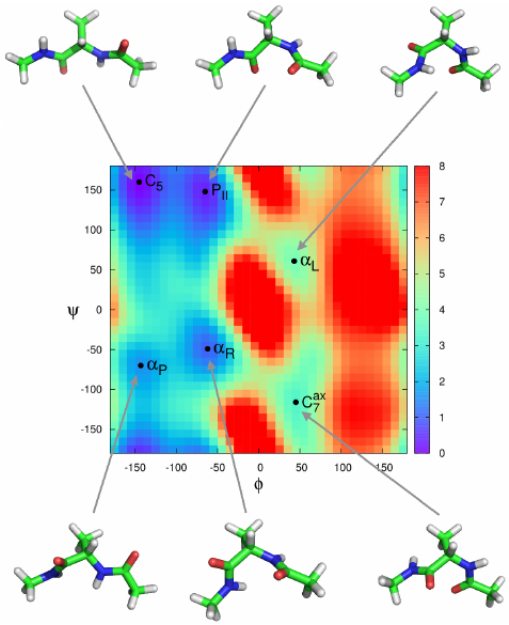


Figure 2.3: Illustrating the torsion angles ϕ and ψ by the example of alanine dipeptide colored by the associated free energy.
Image Source: [8]

2.1.2 Computational Methods for Protein Folding

Classical MD simulations can be used to accurately estimate protein folding. Yet, proteins are too large for these methods to be computationally viable. It is computationally inefficient to compute short timesteps for any molecules, motivating the need for state-of-the-art deep learning methods [52].

There are several computational methods for studying protein folding and predicting the three-dimensional structures of proteins. These methods can be separated into two groups, those that leverage fundamental physical laws to brute-force compute the evolution of the protein, and those who model its behavior through MSMs [47]. In this thesis, we take the latter approach. However, for smaller molecules and smaller timescales, the former provides a way to compute the exact evolution of the molecule.

Based on the laws of thermodynamics, these former techniques predict the most stable shape of a protein using a variety of energy functions. Van der Waals, hydrogen bonds, and electrostatic interactions between protein atoms constitute the most typical energy function employed in protein folding. Quantum mechanical techniques, like quantum mechanics/molecular mechanics (QM/MM) calculations, or empirical force fields, such as the CHARMM or AMBER force fields, are used to calculate these interactions [47, 52]. These mechanical techniques were used to compute the simulation data used in this thesis but prove very computationally expensive.

2.2 Variational Approach for Markov Processes

It is of great importance to be able to, through analyzing time series data, make predictions and exert control over intricate dynamic systems [68], particularly in molecular dynamics. Examination of highly nonlinear dynamical systems is rendered feasible by identifying a nonlinear transformation of the system coordinates, which leads to a representation of the dynamics in terms of features that can be modeled as a linear Markovian system [35, 68]. This approach, Variational Approach for Markov Processes (VAMP), allows us to find optimal feature mappings and optimal Markovian models based on the dynamics from time series data [68]. In addition to its capacity for projecting data into lower dimensional representations, the VAMP model also possesses a set of scoring functions, commonly referred to as VAMP scores, which facilitate the ranking of features.

2.2.1 Markovian Models

The Markov property, which (in a discrete setting) states that the state at time t is only dependant on the value at the previous step $t-1$, forms the foundation of Markovian models [10].

Definition 2.2.1. Markov Property

Let $X_0, X_1, ...$ be a Markov chain. Then, for all $m < n$ and state indices i and j,

$$\begin{aligned} P(X_{n+1} = j | X_0 = i_0, ..., X_{n-m-1} = i_{n-m-1}, X_{n-m} = i) \\ = P(X_{n+1} = j | X_{n-m} = i) = \\ P(X_{m+1} = j | X_0 = i) = P_{ij}^{m+1} \end{aligned} \tag{2.1}$$

for all i,j, i_0, ... , i_{n-m-1} and $n \geq 0$.

The dynamics of a Markovian system can be modeled by the transition density, i.e. the probability density to transition to a state space point $\mathbf{y}$ at time $t+\tau$, given that the system was at state $\mathbf{x}$ at time t, where t is the time step and τ is the delay or lag time [36, 68].

$$p_\tau(\mathbf{x}, \mathbf{y}) = \mathbb{P}\left(\mathbf{x}_{t+\tau} = \mathbf{y} \mid \mathbf{x}_t = \mathbf{x}\right) \tag{2.2}$$

Molecular dynamics can be represented as a Markov process $\{x_t\}$ in the full state space Ω [35]. For a given potential energy function, simulation setup, and time-step integrator, the dynamics are fully described by a transition density $p_\tau(\mathbf{x},\mathbf{y})$. According to Markovian theory, the $\mathbf{y}$ can be sampled by knowing only the $\mathbf{x}$, without knowledge of earlier time steps [10, 35]. Although the dynamics in the variables x_t may be highly nonlinear, Koopman theory explains that the initial variables are changed into some features or latent variables that, on average, evolve in accordance with linear transformation [29].

Using mathematical terms, there exist transformations to features or latent variables, $f(x) = (f_1(x), f_2(x), ...)^T$ and $g(x) = (g_1(x), g_2(x), ...)^T$, such that the dynamics in

these variables are approximately governed by a matrix $\mathbf{K}$ [35]. Thus we can write the following expression for a linear model approximating the Markov dynamics at a lag time τ [35, 68]:

$$\mathbb{E}[g(x_{t+\tau})] \approx \mathbf{K}^T \mathbb{E}[f(x_t)] \tag{2.3}$$

In the limit of an infinite number of features (i.e. as $m \to \infty$), the approximation becomes exact. However, even with low-dimensional feature transformations, the approximation can still yield excellent results for sufficiently large lag times τ [35].

Eq.(2.3) is an algebraic representation of the projection of the Koopman operator $\mathcal{K}$ onto the subspace spanned by functions f and g. $\mathbf{K}$ is therefore referred to as the Koopman matrix [68]. $f(x) = (f_1(x), f_2(x), ...)^T$ and $g(x) = (g_1(x), g_2(x), ...)^T$ are feature transformations that transform the state variables x into the feature space in which the dynamics are approximately linear. $\mathbb{E}$ denotes an expectation value over time that accounts for stochasticity in the dynamics [39, 66].

VAMP builds on this approach of Markovian dynamical systems, in which the future state of the system e.g., $x_{t+\tau}$ only depends on the current state x_t. For sufficiently long observation lag times τ, physics, and engineering processes, such as molecular dynamics, have been shown to be accurately modeled by Markovian models [68].

2.2.2 Koopman Operator

Based on the transition density from Eq.(2.2), the time evolution of the ensemble of system states can be denoted as below [36]:

$$p_{t+\tau}(\mathbf{y}) = \left(\mathcal{P}_\tau p_t\right)(\mathbf{y}) \triangleq \int p_\tau(\mathbf{x}, \mathbf{y}) p(\mathbf{x}) \mathrm{d}\mathbf{x} \tag{2.4}$$

p_t is the probability density of the system being in any state at time t. The resolution of the model is given by the lag time τ. It is possible to model the propagation of general observable functions f as [36]:

$$\mathbb{E}\left[f\left(\mathbf{x}_{t+\tau}\right) \mid \mathbf{x}_t = \mathbf{x}\right] = \left(\mathcal{K}_\tau f\right)(\mathbf{x}) \triangleq \int p_\tau(\mathbf{x}, \mathbf{y}) f(\mathbf{y}) \mathrm{d}\mathbf{y} \tag{2.5}$$

The integral operators $\mathcal{P}_\tau$ and $\mathcal{K}_\tau$ are referred to as the propagator and the Koopman operator, respectively. Both operators are capable of fully capturing the dynamics of a Markovian system, the first propagates the probability density of the system while the second deals with the observable functions of the system.

A central result of the VAMP theory is that the best approximation of Eq.(2.3) is found when the subspaces spanned by f and g are identical to those spanned by the top m left and right singular functions of $\mathcal{K}_\tau$ [35, 68].

By letting ρ_0 represent the empirical distribution of $\mathbf{x}_t$ and ρ_1 represent the empirical distribution of $\mathbf{x}_{t+\tau}$ in all transition pairs $(\mathbf{x}_t, \mathbf{x}_{t+\tau})$. It has been shown that

an optimal finite-rank approximation of the transition density can be expressed as follows [36]:

$$\hat{p}_\tau(\mathbf{x}, \mathbf{y}) = \mathbf{f}(\mathbf{x})^\top \mathbf{S} \mathbf{g}(\mathbf{y}) \rho_1(\mathbf{y}), \tag{2.6}$$

where $\mathbf{S} = \mathbf{K} \left(\mathbb{E} \left[\mathbf{g}\left(\mathbf{x}_{t+\tau}\right) \mathbf{g}\left(\mathbf{x}_{t+\tau}\right)^T \right] \right)^{-1}$ [68]. A finite-dimensional model can effectively capture the fundamental or long-term aspects of the dynamics by choosing $\mathbf{f}$ and $\mathbf{g}$ to be the dominant singular functions or eigenfunctions of the Koopman operator. Recall that the Hilbert-Schmidt norm is given by:

$$\|A\|_{\mathrm{HS}}^2 \stackrel{\mathrm{def}}{=} \sum_{i \in I} \|Ae_i\|_H^2, \tag{2.7}$$

where H is a Hilbert space and where $\{e_i : i \in I\}$ is an orthonormal basis. More specifically, if the modeling error of the Koopman operator is evaluated in terms of the Hilbert-Schmidt norm, the optimal model can be obtained through a truncated singular value decomposition of the transition density.

$$p_\tau(\mathbf{x}, \mathbf{y}) \approx \sum_{i=1}^{k} \sigma_i \psi_i(\mathbf{x}) \phi_i(\mathbf{y}) \rho_1(\mathbf{y}) \tag{2.8}$$

where $(\sigma_1, \ldots, \sigma_k)$ are the largest singular values of the Koopman operator $\mathcal{K}_\tau$, $(\psi_1, \ldots, \psi_k)$ and $(\phi_1, \ldots, \phi_k)$ are the corresponding dominant left and right singular functions [36].

As such, the Koopman operator is a complete description of the dynamical properties of a Markovian system [68]. For deeper insights into the Koopman operator we refer the reader to [68] and [29].

2.2.2.1 Approximating the Koopman Matrix

Given the transformations $\mathbf{f}$ and $\mathbf{g}$, the covariance matrices are defined as:

$$\mathbf{C}_{00} = E_t[\mathbf{f}(x_t)\mathbf{f}(x_t)^T] \tag{2.9}$$

$$\mathbf{C}_{01} = E_t[\mathbf{f}(x_t)\mathbf{g}(x_{t+\tau})^T] \tag{2.10}$$

$$\mathbf{C}_{11} = E_{t+\tau}[\mathbf{g}(x_{t+\tau})\mathbf{g}(x_{t+\tau})^T] \tag{2.11}$$

where $\mathbb{E}_t[\cdot]$ and $\mathbb{E}_{t+\tau}[\cdot]$ denote the expectations that encompass time points and lagged time points within individual trajectories as well as across a set of trajectories [35].

The optimal matrix $\mathbf{K}$ that minimizes the least square error $E_t\left[||\mathbf{g}(x_{t+\tau}) - \mathbf{K}^T\mathbf{f}(x_t)||^2\right]$ is given by:

$$\mathbf{K} = \mathbf{C}_{00}^{-1}\mathbf{C}_{01} \tag{2.12}$$

To find suitable transformations $\mathbf{f}$ and $\mathbf{g}$, one cannot simply minimize the least square error in Eq.(2.12), as for instance $f(x) = g(x) = (1(x))$ i.e., the state space is mapped to the constant 1. In this case the least square error is 0 for $\mathbf{K} = [1]$, but the model is completely uninformative as all dynamical information is lost [35]. Instead, the VAMP variational principle presented by [68] is used, see Section 2.2.3.

2.2.2.2 The Chapman-Kolmogorov Equation

The estimate of the Koopman matrix $\mathbf{K}$ must fulfill the Chapman-Kolmogorov (CK) equation for any $n \geq 1$, namely [35]:

$$\mathbf{K}(n\tau) = \mathbf{K}^n(\tau) \tag{2.13}$$

Where $\mathbf{K}(\tau)$ and $\mathbf{K}(n\tau)$ represent the model at a lag of time τ and $n\tau$, respectively. For model estimation, the equation will only be fulfilled approximately. Thus, we can construct a statistical significance test to validate our model by imposing that the CK equation must hold for our model to be valid. One test that can be employed in order to select a fitting dynamical model is conducting an eigenvalue decomposition for each estimated Koopman matrix, $\mathbf{K}(\tau)\mathbf{r_i} = \mathbf{r_i}\lambda_i(\tau)$, and then computing the implied timescales as a function of the lag time τ [35].

$$t_i(\tau) = -\frac{\tau}{\ln|\lambda_i(\tau)|} \tag{2.14}$$

The timescale refers to the characteristic times at which events or transitions occur within the system, in the case of this thesis this refers to the transition between metastable states. Following this equation, we choose a value of τ where $t_i(\tau)$ is roughly constant in the lag time τ. Using this τ, we can then test if Eq.(2.14) is fulfilled with statistical significance. A test to ensure that the model is sufficiently dynamic goes as follows: Train the neural network and obtain the network transformation χ using a fixed lag time τ^*. Now, an estimation of the ith eigenfunction can be obtained through the following [35].

$$\hat{\psi}^e(x) = \sum_j r_{ij}\chi_j(x) \tag{2.15}$$

If the model satisfies reversibility, the eigenvalue decomposition and singular value decomposition (SVD) are identical. In other words, $\sigma_i = \lambda_i$ and $\psi_i = \psi_i^e$ [35].

2.2.3 VAMP-score

A class of variational scores, designated as $VAMP-r$, for $r = 1, 2, ...$, is proposed to quantify the resemblance between the estimated singular functions and the actual ones through the utilization of the variational representation of singular components. The maximization of any of these variational scores results in the optimal determination of model parameters and is computationally equivalent to the application of Canonical Correlation Analysis (CCA) to the transformed time-lagged pair of variables x_t and $x_{t+\tau}$ [68]. This approach can be employed to learn feature transformation by nonlinear function approximates, such as deep neural networks [68].

Theorem 1 ***Principle for Variational approach for Markov processes (VAMP)***

The k dominant singular components of a Koopman operator are the solution of the following maximization problem:

$$\begin{aligned} \sum_{i=1}^{k} \sigma_i^r &= \max_{\boldsymbol{f},\boldsymbol{g}} \mathcal{R}_r[\boldsymbol{f}, \boldsymbol{g}] \\ s.t\ \langle f_i, f_j \rangle_{\rho 0} &= 1_{i=j} \\ \langle g_i, g_j \rangle_{\rho 1} &= 1_{i=j} \end{aligned} \tag{2.16}$$

where $r \geq 1$ can be any positive integer. The maximal value is achieved by the singular functions $f_i = \psi_i$ and $g_i = \phi_i$ and

$$\mathcal{R}_r[\boldsymbol{f}, \boldsymbol{g}] = \sum_{i=1}^{k} \langle f_i, \mathcal{K}_\tau g_i \rangle_{\rho 0}^r \tag{2.17}$$

is called the VAMP-r score of f and g.

Specifically, the VAMP-1 score maximizes the Rayleigh trace, which represents the sum of the estimated eigenvalues, as established in the works of Noé and Nüske [42] and McGibbon and Pande [37]. Meanwhile, the VAMP-2 score maximizes the kinetic variance, a concept introduced by Noé and Clementi [41]. Hao Wu and Frank Noé [68] suggest r = 2 in applications for the direct relationship between the VAMP-2 score and approximation error of Koopman operators and the convenience of cross-validation, which is pivotal in our thesis.

In this thesis, we implement the VAMP-2 score as the negative loss function. The VAMP-2 score evaluates the sum of the singular value $s = \sum_i \sigma_i^2$ which maximizes the kinetic variance captured by the model [41]. It is also directly related to the approximation error to the true Koopman operator $\mathcal{K}_r$.

We denote the VAMP-2 score of two sets of linearly independent functions $f(x)$ and $g(x)$ by:

$$\mathcal{R}_2[\mathbf{f}, \mathbf{g}] = ||C_{00}^{-\frac{1}{2}} C_{01} C_{11}^{-\frac{1}{2}}||_F^2 \tag{2.18}$$

Where the correlation matrices are defined as Section 2.2.2.1 above and $||M||_F^2$ is the Frobenius norm of the $n \times n$ matrix M. This VAMP-2 score is maximized in the sought-after scenario, i.e. when the top m left and right of the Koopman singular functions belong to the span of $(f_1, ..., f_m)$ and $(g_1, ..., g_m)$, respectively. The VAMP-2 score has been demonstrated to possess more amenable gradients for optimization compared to VAMP-1, which equips it with desirable numerical stability. This is due to its direct relationship with the Koopman approximation error [35]. In the specific scenario where the dynamics exhibit reversibility with regards to the equilibrium distribution, the theorem above specializes to variational principle for reversible Markov processes [42, 44].

2.2.4 VAMPnets

VAMPnets represent a deep learning approach that utilizes neural networks to parameterize transformations $\mathbf{f}$, and $\mathbf{g}$, to optimize a VAMP score [35]. The input data is transformed into a feature representation that optimizes the approximation of the Koopman operator using these transformations. The network is commonly implemented as a multilayer perceptron (MLP), but other architectures have been proposed [16, 35].

, in contrast to prior attempts to incorporate "depth" or "hierarchy" into the analysis approach [45, 48].

VAMPnets combine the tasks of featurization, dimension reduction, discretization, and coarse-grained kinetic modeling into a single end-to-end learning framework [35]. The VAMPnets architecture comprises two parallel components, referred to as lobes, designed to receive the coordinates of time-delayed molecular dynamics configurations as inputs, represented as x_t and $x_{t+\tau}$ [35]. Each lobe in the VAMPnets architecture features $\mathbf{m}$ output nodes and is trained to learn specific transformations, denoted as $f(x_t)$ and $g(x_{t+\tau})$. To evaluate the efficacy of a particular set of transformations, $\mathbf{f}$, and $\mathbf{g}$, a batch of training data is passed through the network and the training VAMP score is calculated.

The initial left and right singular functions of the Koopman operator are always congruent to the constant function $1(\mathbf{x}) \equiv 1$ [68]. Hence, the basis functions can be augmented by adding 1, and the network can be optimized by maximizing [35]:

$$\hat{R}_2 \left[\begin{pmatrix} 1 \\ f \end{pmatrix}, \begin{pmatrix} 1 \\ g \end{pmatrix} \right] = \left\| \overline{\mathbf{C}}_{00}^{-\frac{1}{2}} \overline{\mathbf{C}}_{01} \overline{\mathbf{C}}_{11}^{-\frac{1}{2}} \right\|_F^2 + 1 \tag{2.19}$$

where $\overline{\mathbf{C}}_{00}, \overline{\mathbf{C}}_{01}, \overline{\mathbf{C}}_{11}$ are mean-free covariances of the feature transformed coordinates:

$$\begin{aligned} \overline{\mathbf{C}}_{00} &= (T-1)^{-1} \overline{\mathbf{X}} \overline{\mathbf{X}}^\top \\ \overline{\mathbf{C}}_{01} &= (T-1)^{-1} \overline{\mathbf{X}} \overline{\mathbf{Y}}^\top \\ \overline{\mathbf{C}}_{11} &= (T-1)^{-1} \overline{\mathbf{Y}} \overline{\mathbf{Y}}^\top \end{aligned} \tag{2.20}$$

The matrices are defined as $\mathbf{X} = [X_{ij}] = g_i(\mathbf{x}_j) \in \mathbb{R}^{m\times T}$ and $\mathbf{Y} = [Y_{ij}] = g_i(\mathbf{x}_{j+\tau}) \in \mathbb{R}^{m\times T}$ with $\{(\mathbf{x}_j, \mathbf{x}_{j+\tau})\}_{j=1}^{T}$ representing all available transition pairs, and their mean-free versions $\overline{\mathbf{X}} = \mathbf{X} - T^{-1}\mathbf{X}\mathbf{1}, \overline{\mathbf{Y}} = \mathbf{Y} - T^{-1}\mathbf{Y}$ [35]. The gradients of $\hat{R}_2$ are given by:

$$\nabla_{\mathbf{X}}\hat{R}_2 = \frac{2}{T-1}\overline{\mathbf{C}}_{00}^{-1}\overline{\mathbf{C}}_{01}\overline{\mathbf{C}}_{11}^{-1}\left(\overline{\mathbf{Y}} - \overline{\mathbf{C}}_{01}^{\mathrm{T}}\overline{\mathbf{C}}_{00}^{-1}\overline{\mathbf{X}}\right) \tag{2.21}$$

$$\nabla_{\mathbf{Y}}\hat{R}_2 = \frac{2}{T-1}\overline{\mathbf{C}}_{11}^{-1}\overline{\mathbf{C}}_{01}^{\mathrm{T}}\overline{\mathbf{C}}_{00}^{-1}\left(\overline{\mathbf{X}} - \overline{\mathbf{C}}_{01}\overline{\mathbf{C}}_{11}^{-1}\overline{\mathbf{Y}}\right) \tag{2.22}$$

and are back-propagated to train the two network lobes.

After training, the quality of the learned features and selected hyperparameters (e.g., network size, learning rate) are assessed whilst avoiding overfitting using VAMP-2 score as validation.

$$\hat{R}_2^{\mathrm{val}} = \left\| \left(\overline{\mathbf{C}}_{00}^{\mathrm{val}}\right)^{-\frac{1}{2}} \overline{\mathbf{C}}_{01}^{\mathrm{val}} \left(\overline{\mathbf{C}}_{11}^{\mathrm{val}}\right)^{-\frac{1}{2}} \right\|_F^2 + 1 \tag{2.23}$$

where $\overline{\mathbf{C}}_{00}^{\mathrm{val}}, \overline{\mathbf{C}}_{01}^{\mathrm{val}}, \overline{\mathbf{C}}_{11}^{\mathrm{val}}$ are mean-free covariance matrices computed from a validation data set not used during the training.

2.3 E(3) Equivariant Neural Networks

Traditionally, machine learning has struggled with data embedded in 3D space because the coordinate systems used to describe the data are susceptible to the symmetries of 3D space [14]. Data augmentation is needed to learn 3D patterns in arbitrary orientations, requiring approximately a 500-fold increase in brute training. By building a simple invariant model, the need for data augmentation is removed. However, the expressability of an invariant model is likely insufficient for capturing the higher order symmetries present in molecular data, thus, motivating the need for incorporating higher order symmetries in the machine learning model when using 3D data [14]. In the past five years, this area of research has seen multiple significant developments, which the neural network architecture of this thesis builds upon.

In 2017, Cohen and Weiling introduced Steerable CNNs, which generalize the idea of convolutional neural networks to irregular graphs [5]. The refinement of these networks gave rise to the paper *Tensor field networks: Rotation- and translation-equivariant neural networks for 3D point clouds* [60], which incorporate a systematic approach using spherical harmonics and tensor products to build these networks. This approach builds on previous works on Harmonic Networks [67], and has been widely successful.

Approaches similar to what was presented in [60] have been employed on numerous problems, and multiple novel ideas for network architectures have been proven effective. For example, see [2, 3, 12, 32]

Even though equivariance is straightforward, implementing it needs substantial technical expertise. In an effort to unify the procedures and simplify the use of equivariant neural networks in applied research, the authors of `e3nn` developed a common framework [15]. Subsequently, this thesis leverages prior work by Geiger and Smidt [14] and uses the `e3nn` open-source Python module [15]. The module contains the most crucial and technically demanding concepts, including tensor products, a class for working with irreducible representations, and tools for efficiently computing spherical harmonics.

2.3.1 Group Theory

In our thesis, we deal with 3-dimensional molecular data in Euclidean space. As such, to construct neural networks that respect the symmetries of this space, understanding the group actions to which it is subject and how to represent them for different model inputs is needed. The goal of this section is thus to provide, for the curious reader, a necessary background to the topic.

A group is an algebraic structure that is composed of a set G and a composition law $G \times G \to G$ that obeys the following rules [72]:

- **Closure**: $gh \in G$ for all $g, h \in G$
- **Existence of identity element**: The group G contains an identity element $e \in G$ such that $ae = ea = a$, $\forall a \in G$
- **Associativity**: For elements $a, b, c \in G$, $abc = (ab)c = a(bc)$ for all $a, b, c \in G$
- **Existence of inverse:** Each element $a \in G$ has an inverse denoted $a^{-1} \in G$ such that $aa^{-1} = a^{-1}a = e$

In our thesis we will focus on 3D rotation, translation and inversion, similar to [14, 32]. Thus, the relevant groups for our thesis include [32]:

1. The Euclidean group in three dimensions E(3): 3D rotation, translation, and inversion
2. The special Euclidean group in three dimensions SE(3): 3D rotation and translation
3. The orthogonal group in three dimensions O(3): 3D rotation and inversion
4. The special orthogonal group in three dimensions SO(3): 3D rotation

Combining the group consisting of the set of all 3D rotations with the group of all 3D translations and the group comprising the identity and inversion form the Euclidean Group, E(3) [72]. The group comprising only the 3D rotations and inversions is throughout this paper referred to as O(3).

2.3.2 Representations of Groups

Transformations are defined by the actions of groups. More precisely, a transformation that acts on a vector space X, which is parameterized by a group element

$g \in G$ [32], can be expressed as an injective function $T_g : X \to X$. A representation of a group is a way of expressing the transformations of the group as matrices, such that the group's operations (e.g., rotations, translations) are represented by matrix multiplications [72]. These matrices are called the group representations.

A representation of a group G is a group homomorphism from G to a set of $N \times N$ invertible matrices formally denoted: $D : G \to GL(N)$. Thus, if we have a group element g and a group element h, and we represent them as matrices A and B, respectively, then the matrix representation of their product (gh) will be the product of the matrices A and B [72].

Definition 2.2.3. Group Representation

For a given group G a function $D : G \to \mathbb{R}^{d \times d}$ that maps each element of the group to a $d \times d$ matrix is a representation if and only if it follows the structure of the group [14, 72]:

1. $D(e) = 1$

2. $D(ab) = D(a)D(b)$

Irreducible representations are a particular class of representations that cannot be written as the direct sum of two or more non-trivial representations. These representations are important in group theory because any representation of a group can be decomposed into a direct sum of irreducible representations [72].

2.3.3 Irreducible Representations

The irreducible representations (irreps) of $O(3)$ are the data types for `e3nn`. Another way to think about this is that we must explain to our network how the data transform in the first place if we want our network to maintain the transformation properties of our data. We employ irreducible representations because they are the smallest representation and a complete representation [72], i.e., any finite-dimensional representation of O(3) can be broken down into the irreducible representations of O(3). Inversion, sometimes known as parity, and SO(3) irreps can be formed from the irreps of O(3) [14].

It is possible to create irreducible representations of a group in a variety of ways. One frequent technique is to "project" a representation of the group into a subspace that is unaffected by the group's actions. Utilizing the group characters, which are functions that characterize the traces of the group elements in a specific representation, is another technique. For the purpose of this thesis, the irreducible representations of SO(3) are widely known, referred to as the Wigner D-matrices, and thus do not require us to compute them explicitly. Any group representation of SO(3) can be expressed as a concatenation of irreps [14, 32]:

$$D(g) = P^{-1}\left(\bigoplus_i D_{l_i}(g)\right)P = P^{-1}\begin{pmatrix} D_{l_0}(g) & & \\ & D_{l_1}(g) & \\ & & \ldots \end{pmatrix}P \tag{2.24}$$

where $D_{l_i}(g)$ are Wigner-D matrices with degree l_i.

The general case of this decomposition is such that any representation of SO(3) can be decomposed into irreps of strictly odd dimensions, denoted by $2l+1$ for $l \in \mathbb{N} \cup \{0\}$. The irreps acting on the $2l+1$-dimensional space is referred to as the Wigner-D matrix of order l, denoted in Eq.(2.3.3) as $D_{l_i}(g)$ [65]. Degree l is a non-negative integer and can be interpreted as an angular frequency and determines how quickly vectors change when rotating coordinate systems. $D_L(g)$ of different L can act on independent vector spaces. Vectors transformed by $D_L(g)$ are type-L vectors, with scalars and Euclidean vectors being type-0 and type-1 vectors [32].

This concept will prove important in the construction of our equivariant network, we will use irreps to encode how the features in our network transform under the actions of O(3) and later we will introduce tensor operations that preserve equivariance and can be parameterized by learnable features (weights) such that we can construct the network.

2.3.4 Equivariance

For a group G, a function $f : X \to Y$ is equivariant under G if the following equation holds for $\forall g \in G, x \in X$ where $D_X(g)$ and $D_Y(g)$ is the respective representations of the group element g acting on the vector space X and Y respectively.

$$f(D_X(g)x) = D_Y(g)f(x) \tag{2.25}$$

This means that acting with the group on the inputs or the outputs is equivalent. E.g. applying a rotation to the input is the same as first applying the function and then applying the rotation to the output, we can then say that the function is equivariant with respect to the group.

An important aspect of equivariant functions, which is crucial for the creation of equivariant neural networks, is the following three theorems.

1. If both $f : X_1 \to X_2$ and $g : Y_1 \to Y_2$ are equivariant, their composition $f \circ g$ is also equivariant.
2. If both $f : X_1 \to X_2$ and $g : Y_1 \to Y_2$ are equivariant, their sum $f + g$ is also equivariant.
3. Multiplication: the tensor product of the two vectors can be shown to be equivariant. This property is further detailed in Section 2.3.6.

In particular, this means that Euclidean neural networks (where $G = E(3)$), incorporating learned scalar weights, guarantee equivariance simply by requiring individual components to be equivariant [14].

The last property, where a certain definition of the tensor product preserves equivariance, enables the construction of neural networks through incorporating learnable weights in the coefficients of the tensor product.

2.3.5 Spherical Harmonics

While applications of spherical harmonics can be found throughout mathematics, physics, and chemistry, we will restrict our discussion to what relates to creating E(3) equivariant neural networks. In this context, drawing inspiration from the work proposed by [14, 32], spherical harmonics are basis functions for irreducible representations of SO(3).

Euclidean vectors $\vec{r} \in \mathbb{R}^3$ can be projected into type-L vectors denoted $f^{(L)}$ using spherical harmonics $Y^{(L)}$: $f^{(L)} = Y^{(L)}\left(\frac{\vec{r}}{||\vec{r}||}\right)$. Spherical harmonics are a family of functions Y^l from the unit sphere to the irreducible representation D^l, where l represents the angular velocity, detailing how objects are transformed under rotation. They are equivariant with:

$$D_L(g)f^{(L)} = Y^{(L)}\left(\frac{D_1(g)\vec{r}}{||D_1(g)\vec{r}||}\right) \tag{2.26}$$

Spherical harmonics of relative position $\vec{r_{ij}}$ generates the first set of irreps features. These are later used to form the internal representations in our equivariant neural networks.

The spherical harmonics can be regarded as a $2l+1$ dimensional vector of functions $Y^l(\vec{x}) = (Y^l_{-l}(\vec{x}), Y^l_{-l+1}(\vec{x}), ..., Y^l_l(\vec{x}))$, $\forall l \in \mathbb{N} \cup \{0\}$:

$$Y^l_m(R\vec{x}) = \sum_{n=-l}^{l} D^l(R)_{mn} Y^l_n(\vec{x}) \tag{2.27}$$

R can be any rotation matrix and D^l are the irreducible representations of SO(3). The equation above implies that each Y^l is equivariant to the actions of SO(3) with respect to the irreducible representation of the same order.

The spherical harmonics are heavily dependent on the choice of basis for the irreducible representations. Once the basis of the irreducible representations is fixed, the spherical harmonics functions are unique for $\vec{x}$ on the unit sphere up to the choice of their sign $\pm Y^l$ [14].

2.3.6 Tensor Product

To interact with different type-(L,p) vectors, where L is the type and p is parity, tensor products are used [32]. In the tensor product, Clebsch-Gordon coefficients are used to combine type-L_1 vector $f^{(L_1)}$ and type-L_2 vector $g^{(L_2)}$ and produce type-L_3 vector $h^{(L_3)}$. The equation below displays the tensor product for the aforementioned vectors for the SO(3) group:

$$h_{m_3}^{(L_3)} = \left(f^{(L_1)} \otimes g^{(L_2)}\right)_{m_3} = \sum_{m_1=-L_1}^{L_1} \sum_{m_2=-L_2}^{L_2} C_{(L_1,m_1)(L_2,m_2)}^{(L_3,m_3)} f_{m_1}^{(L_1)} g_{m_2}^{(L_2)} \tag{2.28}$$

In Eq.(2.28) above, m_1 corresponds to the m_1-th element of $f^{(L_1)}$ and equates to the order. For $|L_1 - L_2| \leq L_3 \leq |L_1 + L_2|$ the Clebsch-Gordan coefficients above, $C_{(L_1,m_1)(L_2,m_2)}^{(L_3,m_3)}$ are non-zero. For the Clebsch-Gordon coefficients to be zero implies that the output vectors are restricted to certain types. While in theory, vectors of increasingly higher dimensions could prove useful to capture higher-order symmetries, the operations scale poorly and it is thus practical to limit the degree of internal representations in the networks. For example, while the tensor product of a type-L_1 vector and a type-L_2 vector will produce a type-L_3 vector as its highest dimensional output, we could constrain the model output to L_2 and thus set the coefficient for the type-L_3 vector to zero.

We denote our hyperparameter for limiting vectors of increasingly higher dimensions as L_{max} and will omit vectors with type $L > L_{\text{max}}$. This will prove important later in order to constrain our network's internal representations.

Each distinct non-trivial combination of $L_1 \bigotimes L_2 \rightarrow L_3$ is referred to as a path. Each path is independently equivariant, and it is possible to designate one learnable weight to each path.

There is one difference between tensor products for O(3) and SO(3) [32]. For O(3) we must keep track of the output parity p_3, where $p_3 = p_1 \times p_2$, and adhere to the following multiplication rules:

1. $e \times e = e$
2. $o \times o = e$
3. $e \times o = o \times e = o$

To clarify, a tensor product of a type-(1,e) and vector and a type-(1,e) vector might lead to one type-(0,o) vector. one type-(1,o) vector and one type-(2,o) vector.

The Clebsch-Gordan coefficients mentioned above are the change of basis decomposing the tensor product into irreps [14, 32]. For SO(3) they satisfy the equation below:

$$C_{lmn}^{l_1,l_2,l_3} = C_{ijk}^{l_1,l_2,l_3} D_{il}^{l_1}(R) D_{jm}^{l_2}(R) D_{kn}^{l_3}(R) \quad \forall R \in SO(3) \tag{2.29}$$

The Clebsch-Gordan coefficients for SO(3) are calculated by integrating over the basis functions of a specified irreducible representation, in other words, the real spherical harmonics. The equation can be viewed below:

$$C^{(L_3,m_3)}_{(L_1,m_1)(L_2,m_2)} = |L_1 m_1; L_2 m_2\rangle \langle L_3 m_3| = \int d\Omega Y^{(L_1)*}_{m_1}(\Omega) Y^{(L_2)*}_{m_2}(\Omega) Y^{(L_3)}_{m_3}(\Omega) \quad (2.30)$$

For the majority of combinations of L_1, L_2 and L_3 the coefficients are zero, leading to the subsequent selection rule for coefficients that are not trivial: $-|L_1 + L_2| \leq L_3 \leq |L_1 + L_2|$.

2.4 Graph Convolutional Neural Networks

Graph Convolutional Neural Networks (GCNNs) are a class of neural network models designed to operate on graph-structured data. GCNNs leverage the concept of convolution, commonly used in image processing in state-of-the-art architectures such as ImageNet [54] and ResNet [20], and adapt it to graph domains.

In a GCNN, the key idea is to generalize the notion of convolution from regular grids (like images) to irregular graphs. The goal is to learn node representations that capture both the node's local neighborhood information and the global graph structure.

By repeatedly applying the graph convolution operation, information is exchanged and updated across nodes, enabling the network to capture both local and global patterns in the graph structure. The final learned representations can be used for various downstream tasks, such as node classification, link prediction, or graph-level prediction.

State-of-the-art architectures include ChebNet [7], EdgeConv [64], Graph U-Net [13] and GraphSAGE [18].

2.4.1 Message Passing

Message passing is a core mechanism in any Graph Neural Network (GNN), this includes the Transformer architecture, outlined in Section 2.5, when adapted to irregular graphs and using the attention mechanism as the means of message propagation. However, we choose to outline the message passing schema here and dive deeper into the attention mechanism later.

A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is formally made up of edges $(i, j) \in \mathcal{E}$ that reflect connections between nodes i and $j \in \mathcal{V}$. For this thesis, a suitable example of a graph is an atomistic system, such as a molecule, where each node is an atom and edges can be bonds or atom pairs close together. Via layers of message passing, GNNs derive meaningful representations from graphs. A message passing scheme takes a given feature f_i^l on node i and edge feature e_{ij}^l at the l-th layer and first aggregates

messages m_{ij}^l from neighbors and then updates features on each node as according to the following equation:

$$m_{ij}^l = \phi_m(f_i^l, f_j^l, e_{ij}^l) \text{ and } f_i^{l+1} = \phi_u(f_i^l, \sum_{j \in \mathcal{N}(i)} m_{ij}^l) \tag{2.31}$$

where ϕ_m and ϕ_u represent learnable functions and $\mathcal{N}(i)$ denotes the set of neighbors of node i. Each node i in atomistic systems in 3D spaces is also connected with its location, or $\bar{r_i}$, and edge properties often become functions of relative position, or $\bar{r_{ij}} = \bar{r_j} - \bar{r_i}$ [32]. As we are dealing with atomistic systems, our approach will be part of the general GNN approach.

2.4.2 Equivariant Graph Convolutions

As in previous papers on this topic [2, 3, 60, 65], the GCNN used in this thesis uses the message passing schema on irregular graphs presented with a convolutional kernel as the learnable functions ϕ_m and ϕ_u above. Following the work of Batzner et al. [2], in this thesis, we use equivariant convolutions acting on geometric objects that comprise a direct sum of irreps of E(3).

Since convolution filters operate on relative interatomic distance vectors, these operations are by nature translation invariant. A very important property is that the sum of contributions from several atoms is invariant under permutations of those atoms. Permutations in this context refers to permuting the atom indices. The overall potential energy of the system is invariant to permutation, whereas atomic properties are equivariant to permutation of atom indices. To ensure rotational equivariance, the convolutional filters $S_m^{(l)}(\overrightarrow{r_{ij}})$ are limited to be the resulting products of learnable radial functions and spherical harmonics, which as pointed out earlier, are equivariant under the actions of SO(3). The convolutional filter can be written as:

$$S_m^{(l)}(\overrightarrow{r_{ij}}) = R(r_{ij}) Y_m^{(l)}(\hat{r_{ij}}) \tag{2.32}$$

where $\overrightarrow{r_{ij}}$ is used to denote the relative position from the atom i central to the convolution to its neighboring atom j. $\hat{r_{ij}}$ and r_{ij} denote the associated unit vector and interatomic distance, respectively, and lastly, as above, $S_m^{(l)}(\overrightarrow{r_{ij}})$ denotes the convolutional filter [2].

2.5 Transformers

Transformer architectures, leveraging the mechanics of attention, have seen a remarkable increase in popularity [61]. Its widespread adoption can be attributed to a design that is both well-suited for backpropagation and parallelization. Moreover, it has shown excellent efficacy on a variety of tasks, including language modeling

[61], image recognition [51], and graph-structured data [62]. Recently, work applying the Transformer architecture on point-cloud data has been published displaying promising results [31, 70, 71].

However, in its general form, the Transformer architecture does not encode any constraints imposed by 3D space. In particular, models have emerged for 3D point clouds and graph data offering attention mechanisms that preserve equivariance to the actions of the Euclidean group. Such models have been proposed by [2, 12, 32, 59]. In this paper, we leverage the work on this topic, in particular, we use the Equiformer model, developed by Liao and Smidt [32], as one of our equivariant neural network architectures.

Proposed by Vaswani et al. [61] the Transformer architecture relies on attention mechanisms to understand global dependencies between input and output.

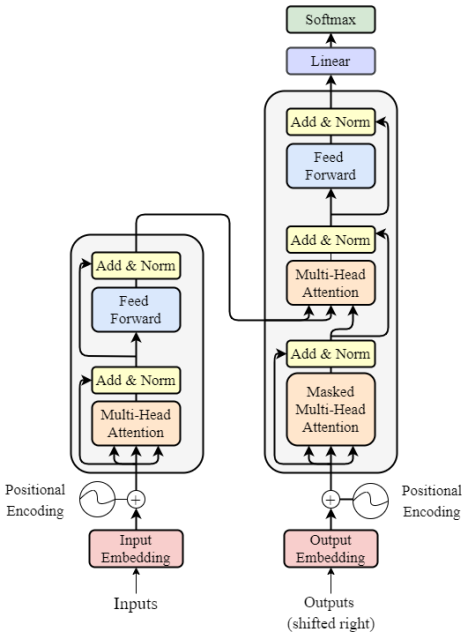


Figure 2.4: The original Transformer architecture proposed by Vaswani et al. [61] for machine translation.
Image Source: [61]

Transformers use stacked self-attention and point-wise, fully connected layers for both the encoder and decoder. This is shown in Figure 2.4.

2.5.1 Self-Attention

Self-attention is an attention mechanism that establishes connections between various positions within a single sequence to generate a sequence representation. Formally it can be described as the mapping of one-variable length sequence of symbol representations $(x_1, ..., x_n)$ to another sequence of equal length $(z_1, ..., z_n)$ with $x_i, z_i \in \mathbb{R}$. The attention mechanism for Transformers can be described as mapping a query and a set of key-value pairs to an output, all being represented in form of

vectors [61]. We denote queries q and keys k, where both vectors are of dimension d_k. Let v denote a vector of dimension d_v containing the values. The attention function is computed over a set of queries, keys, and values packed together, denoted Q, K, and V.

A weighted sum is used for computing the output of the values, where the weight is determined by a compatibility function of the query with its corresponding key.

2.5.1.1 Scaled Dot-Product Attention

The typical attention mechanism in Transformers is the scaled dot-product [32]. To compute an attention matrix we used the packed queries Q, keys K, and values V. The dot product is computed between Q and K, and the results are scaled using $\sqrt{d_k}$, which is the dimension of q and k. A softmax function is then applied to obtain the weights of the values [61]. The final step is to multiply the weights with the values V. The function can be viewed below:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}V}) \tag{2.33}$$

2.5.1.2 Multi-headed Attention

However, it was found that it is beneficial to linearly project the queries, keys, and values h times with different learned projections each time. For each of these h projections, the single attention mechanism is applied in parallel, producing h outputs. These h outputs are then concatenated and projected again yielding the final result. This way of parallelizing was proposed by Vaswani et al. [61] and is called a multi-head attention mechanism. The function for multi-head attention can be viewed below:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O \tag{2.34}$$

Where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$ $\forall i = 1, ...h$ and the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_k}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$

Thus one first computes the linearly projected versions of the queries, keys, and values by multiplying with the respective weight matrices (W_i^Q, W_i^K, W_i^V), for each head_i. Then the single attention function, from Eq(2.33), is applied for each head_i. The outputs of the heads are concatenated, and lastly, a linear projection is applied to the concatenated output by multiplying it with W^O.

2.5.2 Equivariant Transformers

Transformers have seen widespread success but have yet to perform well across datasets in the domain of 3D atomistic graphs, such as the protein folding data set we are using for this thesis. However, recently Liao and Smidt [32] proposed an

Equiformer, a graph neural network leveraging the strength of the Transformer architectures as well as incorporating SE(3)/E(3) equivariant features based on irreps. Two types of Equiformers are presented, a simple and effective architecture, which replaces the original operations in Transformers [61] with equivariant operations like tensor products, and a novel attention mechanism which is called equivariant graph attention [32]. The latter replaces dot product attention with MLP attention and includes non-linear message passing. In the thesis, we employ both models to test which performs better on the alanine dipeptide data. The Equiformer architecture can be viewed below.

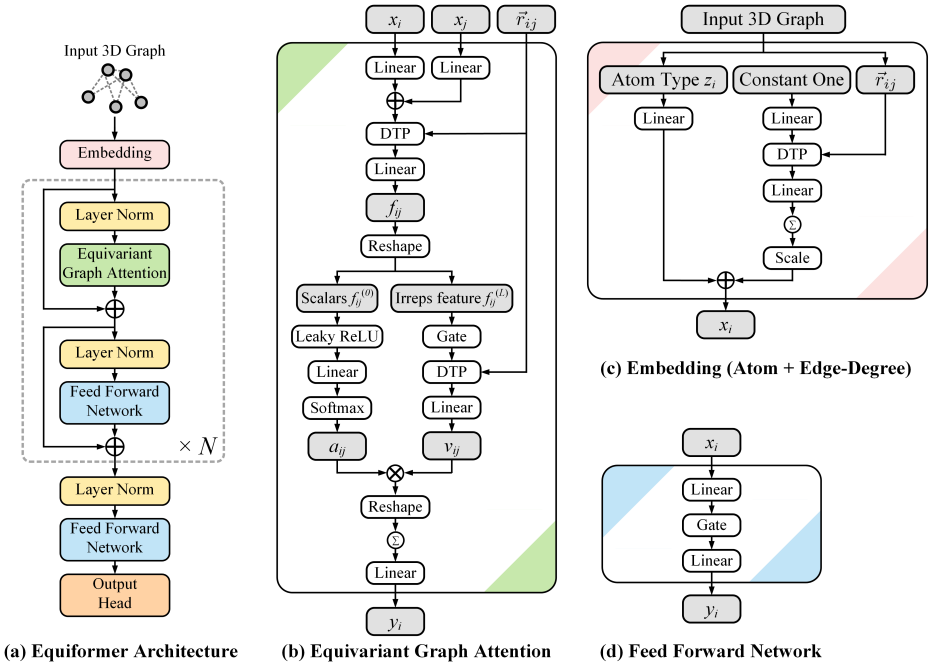


Figure 2.5: The Equiformer architecture proposed by [32] for 3D atomistic point clouds. The 3D graphs used as input are embedded with atom and edge-degree embeddings and processed through Transformer blocks consisting of equivariant graph attention and feed-forward networks. In this figure, $\otimes$ denotes multiplication, $\oplus$ denotes addition, and DTP stands for Depth-wise Tensor Product. P within a circle denotes summation over all neighbors. Gray cells indicate intermediate irreps features.
Image Source: [32]

2.5.2.1 Simple Equiformer Architecture

In order to implement something similar to the Transformer architecture by Vaswani et al. [61] but preserving equivariance, some technical details are needed to assemble the building blocks of the Transformer. Luckily, previous research on the topic [2, 12, 32, 59] have developed efficient tools to achieve this.

Below follows an account of the most important building blocks for the Transformers and a brief description of how these are implemented in an equivariant fashion, as proposed by Liao and Smidt [32].

The operations the simple Equiformer architecture replaces from the original Transformer are the linear layers, layer normalization (LN), gate activation, and Depth-wise Tensor Product (DTP) [32, 61]. The different operations are visualized in the graph below.

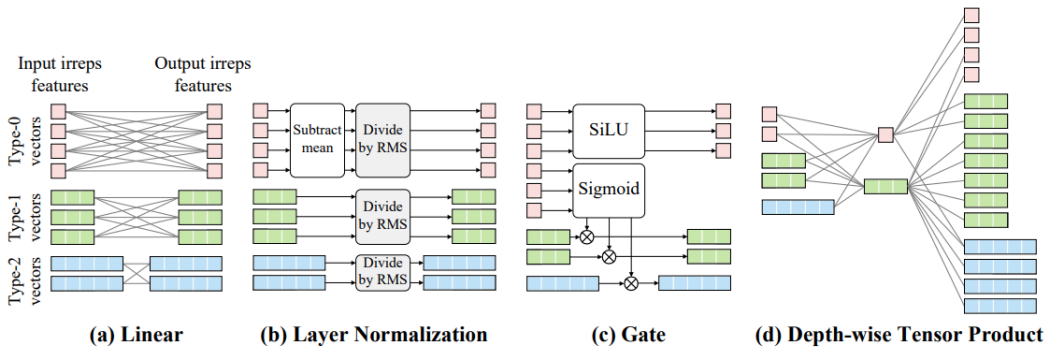


Figure 2.6: Equivariant operations used in Equiformer
Image Source: [32]

In their work, Liao and Smidt [32] transform different type-L vectors using separate linear operations for each group of type L-vectors in order to generalize linear layers in the feed-forward neural network to irreps features. The bias term is removed for the non-scalar feature when L > 0 as biases are independent of inputs, preventing potential breaks of equivariance.

Layer normalization (LN) is employed in the Transformers architecture to stabilize training. LN uses an input of $x \in \mathbb{R}^{N \times C}$, where C is the number of channels and N is the number of nodes. It calculates the linear transformation of the normalized input as $\mathrm{LN}(x) = \left(\frac{x-\mu_C}{\sigma_C}\right) \circ \gamma + \beta$, where μ_C and $\sigma_C \in \mathbb{R}^{N \times 1}$ denote the mean and standard deviation of the input x along the channel dimension, and γ and $\beta \in \mathbb{R}^{1 \times C}$ are learnable parameters. The element-wise product is denoted by the symbol $\circ$. By treating the standard deviation as the root mean square value (RMS) of the L2-norm of type-L vectors, LN can be extended to irreps features [32].

For an input $x \in \mathbb{R}^{N \times C \times (2L+1)}$ of type-L vectors, the output is $\mathrm{LN}(x) = \left(\frac{x}{\mathrm{RMS}_C(\mathrm{norm}(x))}\right) \circ \gamma$, where $\mathrm{norm}(x) \in \mathbb{R}^{N \times C \times 1}$ calculates the L2-norm of each type-L vector in x, and $\mathrm{RMS}_C(\mathrm{norm}(x)) \in \mathbb{R}^{N \times 1 \times 1}$ calculates the RMS of the L2-norm with mean taken along the channel dimension. Means and biases for type-L vectors with $L \neq 0$ are removed [32].

To maintain the equivariance when employing the activation function, Liao and Smidt [32] use the gate activation function proposed by Weiler et al. [65]. Activation functions are typically applied to L-type vectors where $l = 0$. When $l > 0$ the vectors are multiplied with non-linearly transformed $l = 0$ type L-vectors for equivariance.

If the input x contains non-scalar C_L type-L vectors where $0 < l \leq L_{\max}$ and $(C_0 + \sum_{l=1}^{L_{\max}} C_L)$ type-0 vectors, then Sigmoid-Weighted Linear Units (SiLU) [11, 50] is applied to the first C_0 type-0 vectors, and sigmoid function to the other $\sum_{l=1}^{L_{\max}} C_L$

type 0-vectors. With this non-linear weights are obtained, and each type-L vector can be multiplied with its corresponding non-linear weights.

Tensor products are a mathematical operation that describe interactions between vectors of different L (irreps). Its efficiency can be improved by using the DTP, where one type-L vector in output irreps features depends only on one type-L' vector in input irreps features [32], which can be viewed in Figure 2.5.2.1, where L is equal or different to L'. In other words, one output channel depends on only one input channel. The DPT between x and y, two tensors, is denoted $x \otimes_w^{DTP} y$, where w denotes the weight. The weight can be input-independent or conditioned on relative distances [32].

Equivariant Dot-Product Attention

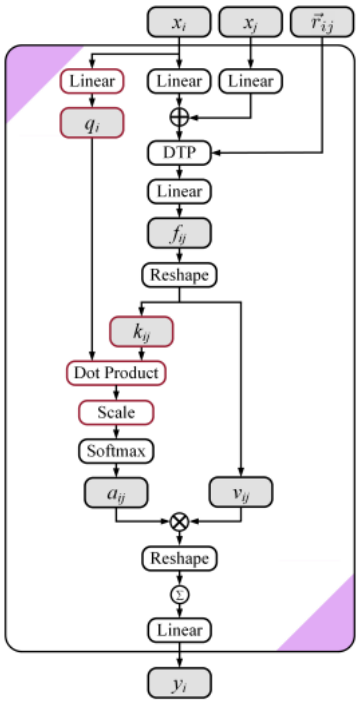


Figure 2.7: Architecture of equivariant DP attention without non-linear message passing proposed by [32]. Illustration: ⊗ denotes multiplication, ⊕ denotes addition, and DTP stands for Depth-wise Tensor Product. P within a circle denotes summation over all neighbors and gray cells indicate intermediate irreps features. The red color is used to highlight the difference between dot product attention and multi-layer perceptron attention.
Image Source: [32]

The equivariant dot-product attention mechanism is presented in Figure 2.7 above. The multi-layer perceptron attention differs in how attention weights, a_{ij}, are derived from f_{ij}. Liao and Smidt split f_{ij} into key k_{ij} and value v_{ij}, two irreps features, and obtain the query q_i with a linear layer [32]. Thereafter, the scaled dot product [61] is used between q_i and k_{ij} to obtain the attention weights.

2.5.2.2 Equivariant Graph Attention

The Equivariant Graph Attention architecture proposed by Liao and Smidt [32] does not only replace original operations in the Transformer with their equivariant counterparts but proposes a novel attention mechanism, improving on the typical attention in Transformers, by replacing the dot product attention with multi-layer perception attention and including non-linear message passing. The architecture is illustrated in Figure 2.6. There are two main components to this Equiformer architecture, multi-layered perceptron attention and non-linear message passing.

In the multi-layered perceptron attention [32], attention weights a_{ij} describe how each node interacts with its neighboring nodes. The attention weights are invariant to geometric transformation, hence only type-0 vectors of message f_{ij} are denoted as f_{ij}^0 for attention. The messages incorporate directional information as they are generated by tensor products of type-L vectors with $L \geq 0$. In comparison with the dot product, MLPs are universal approximators and can thus capture any attention patterns, in theory, [6, 22, 23]. For f_{ij}^0, one leaky ReLu layer and one linear layer for a_{ij} are used:

$$z_{ij} = a^T \text{LeakyReLU}(f_{ij}^{(0)}) \tag{2.35}$$

$$a_{ij} = \text{softmax}_j(x_{ij}) = \frac{\exp x_{ij}}{\sum_k \in N(i) \exp z_{ik}} \tag{2.36}$$

$f_{ij}^{(0)}$ and a have the same dimension, where a is a learnable vector and z_{ij} is a single scalar. The output from the attention is calculated by summing the value of v_{ij} and multiplying it with the corresponding a_{ij} over all neighboring nodes $j \in N(i)$. v_{ij} can be obtained by linear or non-linear transformations of f_{ij}.

Values v_{ij} are transformed with input-independent weights and are sent from one node to another. f_{ij} is split into $f_{ij}^{(0)}$ and $f_{ij}^{(L)}$, where the latter comprises of type-L vectors with $0 \leq L \leq L_{\text{max}}$. As discussed above, $f_{ij}^{(0)}$, consists of scalars only. Non-linear transformation is performed on $f_{ij}^{(L)}$ to obtain the non-linear message:

$$\mu_{ij} = \text{Gate}(f_{ij}^{(L)}) \tag{2.37}$$

$$v_{ij} = \text{Linear}([\mu_{ij} \bigotimes_{w}^{DTP} SH(\overrightarrow{r_{ij}}]) \tag{2.38}$$

To retrieve μ_{ij}, gate activation is applied to $f_{ij}^{(L)}$. One DTP and a linear layer are used to enable interaction between the non-linear type-L vectors, similar to when transforming x_{ij} into f_{ij}. $f_{ij}^{(L)}$ can also be used directly for v_{ij} for linear messages. Weights w are input-independent.

We have introduced group theory and representation theory and given examples of how to use this mathematical frameworks to build neural network architectures

that preserve equivariance to the Euclidean symmetry group (E(3)). Lastly, we have detailed the central concepts of the two equivariant model architectures we employ in this thesis, the Transformer and the GCNN.

3

Methodology

This chapter provides a comprehensive overview of the experimental design and methodology applied in the thesis. It encompasses the datasets, pre-processing pipeline, network architecture, and software. Furthermore, this chapter outlines the training procedure, loss function, and relevant implementation details of the VAMPnets.

3.1 Data Description & Pre-Processing

This thesis uses a gradual progression of data, similar to how Mardt et al. [35] tackled multiple, progressively more complex datasets for their VAMPnets model. Initially, we test various models on a simple molecule trajectory dataset for the peptide comprising two amino acids called alanine dipeptide. Ultimately, we benchmark the best-performing model on the protein folding dataset.

3.1.1 Alanine Dipeptide

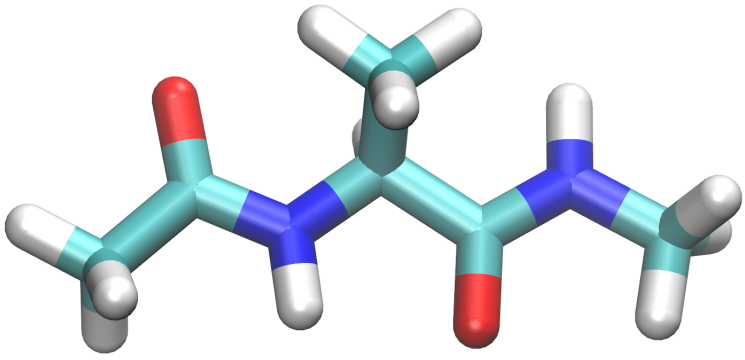


Figure 3.1: Illustration of alanine dipeptide, consisting of short chains of amino acids linked by peptide bonds.
Image Source: [63]

Data Description

The data originates from a molecular dynamics simulation set up of alanine dipeptide (Ac-A-NHMe) in explicit water by Nüske et al. [43]. It is well known that the dynamics of alanine dipeptide can be described by the twodimensional space of backbone dihedral angles ψ and ϕ. The aim is to detect the energetically allowed regions for the backbone dihedral angles of amino acid residues in protein structures, using the 3D coordinates to infer ψ and ϕ.

The alanine dipeptide data consists of three trajectories and in total 750,000 timesteps where only the peptide configuration is saved while the water molecules are integrated out. Details of the simulation setup are provided below. For further information, we refer to the paper by Nüske et al. [43]. The data was retrieved using the MD share library, which is maintained by Frank Noé's group at FU Berlin.

Property	Value
Software	ACEMD
Force Field	AMBER ff-99SB-ILDN
Integrator	Langevin
Integrator Time Step	2 fs
Simulation Time	250 ns
Frame Spacing	1 ps
Temperature	300 K
Volume	2.3222nm^3 periodic box
Solvation	651 TIP3P waters
Electrostatics	PME
PME real-space cutoff	0.9 nm
PME grid spacing	0.1 nm
PME updates	Every two time steps
Trajectories	3

Table 3.1: Simulation setup of the alanine dipeptide dataset

ACEMD is a software framework for MD-based discovery [19] and the selected force field, which is a mathematical model used to describe the interactions between the atoms in a molecular system, is based on the work by Lindorff-Larsen et al. [34]. `MDshare` provides the complete trajectories of alanine dipeptide, consisting of all 22 atoms of the molecule, and a pre-processed dataset, with only the heavy atoms included. The heavy atoms consist of Oxygen (O), Nitrogen (N), and Carbon (C), excluding the 12 Hydrogen atoms (H). However, there is no corresponding Protein Data Bank (.pdb) file with the atomic features for the pre-processed dataset. Thus,

we use the complete trajectories and remove the hydrogen atoms to leverage the features in our training.

We choose to only use the heavy atoms as the hydrogen atoms display very fast evolutions. However, through the use of VAMPnets, we are detailing the slow dynamics of the molecule. As we do not want the fast dynamics to cloud our judgement, similar to previous research by Mardt et al. and Ghorbani et al. [16, 35, 36], we will use only the heavy atoms in our dataset, both for alanine dipeptide and the protein folding dataset.

Data Pre-Processing

We use `MDtraj` to load the trajectories from the `MDshare` files [38]. The atom type is considered a categorical feature and is pre-processed using one-hot encoding. One-hot encoding provides a binary representation rather than an ordinal one, facilitating learning for our model. Later in the thesis, we will differentiate between this type of atom encoding and encoding all atoms with a unique one-hot encoding. `TrajectoriesDataset` from `deeptime.util.data` is used to obtain time-lagged data. Essentially displacing the current trajectory with a time-step equivalent to τ.

Another important graph attribute to encode is the adjacency matrix, which details the connections between the nodes. This adjacency matrix is not explicitly encoded in the `.pdb` data we use. Hence, we set a radial cutoff to create our adjacency matrix based on the distance matrix obtained from the positions. For this, we use the function `torch_cluster.radius_graph`, which creates the adjacency matrix based on the nodal positions.

The last step of the data pre-processing is to convert the information retrieved into a `torch_geometric.data.Data` object, which can then be input into a `torch_geometric.loader.DataLoader` to implement advanced mini-batching. The `DataLoader` is now ready to be handled by our E(3) equivariant neural networks.

3.1.2 Protein Folding

We use the processed version of the protein folding dataset provided by DE Shaw Research [33]. Unlike the alanine dipeptide dataset, the protein folding data is not publicly available and was retrieved from our supervisor with the sole purpose of pursuing this thesis.

Data Description

The data stems from MD simulations of varying time lengths. Each time-step is 200 ps between subsequent frames. All frames have been superimposed to a folded protein structure state. In total, the data covers 12 different proteins.

Protein Structure	Heavy Atoms
BBA (1FME)	251
Villin (2F4K)	287
Trp-cage (2JOF)	144
BBL (2WAV)	352
a3D (A3D)	572
Chignolin (cln025)	93
WW domain (GTT)	287
NTL9 (2HBA)	301
Protein G (NUG2)	435
Protein B (1PRB)	362
Homeodomain (UVF)	466
λ-repressor (1LMB)	618

Table 3.2: Protein structures in the protein folding data set

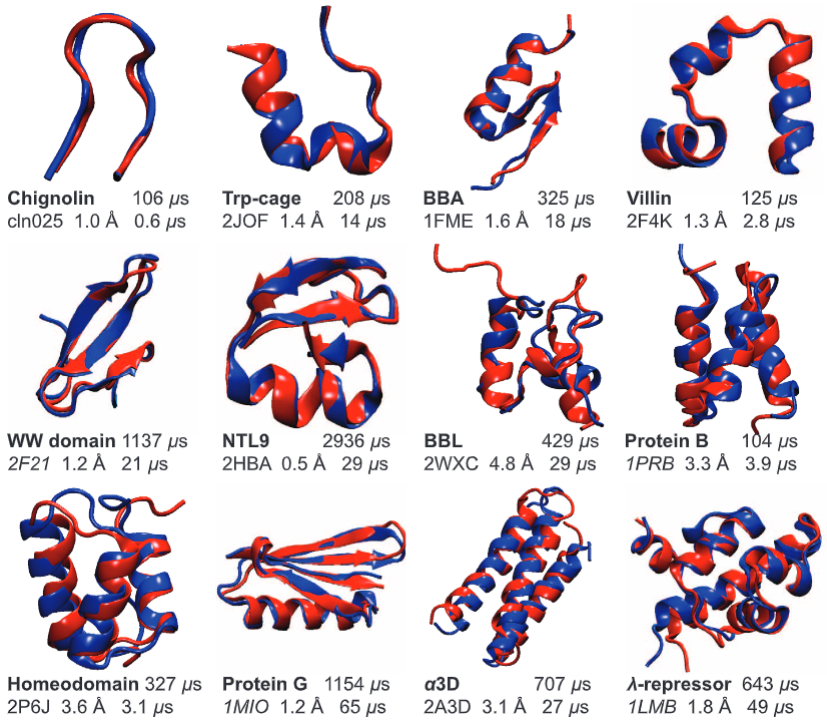


Figure 3.2: Representative structures of the folded state were observed in reversible folding simulations of 12 proteins. The folded structure is provided from simulation for each protein (blue) overlaid over the experimentally determined structure (red), together with the total simulation time, the experimental structure's PDB entry, the Cα-RMSD (over all residues) between the two structures, and the folding time (obtained as the average lifetime in the unfolded state observed in the simulations). **Image Source:** [33]

Using equilibrium MD simulations, the data investigates the fast-folding protein mechanism, examining 12 protein domains with sizes ranging from 10 to 80 amino acids, no disulfide linkages, no prosthetic groups, and representatives from each of the three main structural types (α-helical, β-sheet, and mixed α/β) [33].

The dataset for the 12 proteins includes simulations conducted close to the melting point, where the melting point represents the temperature at which stability is compromised, leading to the unfolding of the protein, thus, folding and unfolding could be frequently seen in an extended equilibrium MD simulation. As a part of the experiments, one to four simulations were performed, lasting between 100 μs and 1 μs for each of the 12 proteins. A minimum of ten foldings and ten unfolding events for each simulation were recorded, and over 400 folding or unfolding events occurred throughout approximately 8 ms of simulation time. The simulations are best-in-class and rival previous state-of-the-art simulations while providing high-quality data for multiple proteins obtained under homogeneous conditions, previously never done. Each protein simulation is stored in two files, one in `.h5` format, containing all the aligned coordinates of the simulation, and one in `.pdb` format comprising atomic information such as bonds and residues.

Data Pre-Processing

Data of the 12 proteins are stored in the format of HDF5 files, binary data format, and their topology data as .pdb files. Using the Python library `h5py` and `mdtraj`, we extract the data and the topology.

Similarly to the alanine dipeptide data, a time-lagged trajectory is created for each trajectory by displacing it with a time-step equivalent to τ. One-hot encoding is also performed to deal with the categorical nodal features. Lastly, the data is packaged into data loaders that can be handled by our equivariant neural network, just like for the alanine dipeptide dataset.

3.2 Experimental Setup

This section covers the experimental setup, including VAMPnets and neural network architectures. Important choices in the implementation of these are also covered, such as advanced mini-batching, edge length embedding, directional encoding using spherical harmonics, message passing, graph convolution, and feature reduction using a multilayered perceptron. Lastly, an overview of all the experimental models used is provided.

3.2.1 VAMPnets Structure

Deeptime [21], a Python library, is used for implementing VAMPnets similarly to Mardt et al. [35], with the major difference being the neural network architecture. Mardt et al. [35] used a multilayered perceptron, while our implementation builds on an equivariant neural network. We experiment with two types of networks that preserve equivariance, a GCNN as provided by the `e3nn` library [14] and equivariant Transformers as proposed by Liao and Smidt [32]. Research has shown that converting from invariant to equivariant models, even with the same framework, has improved accuracy in the training tasks [2, 40].

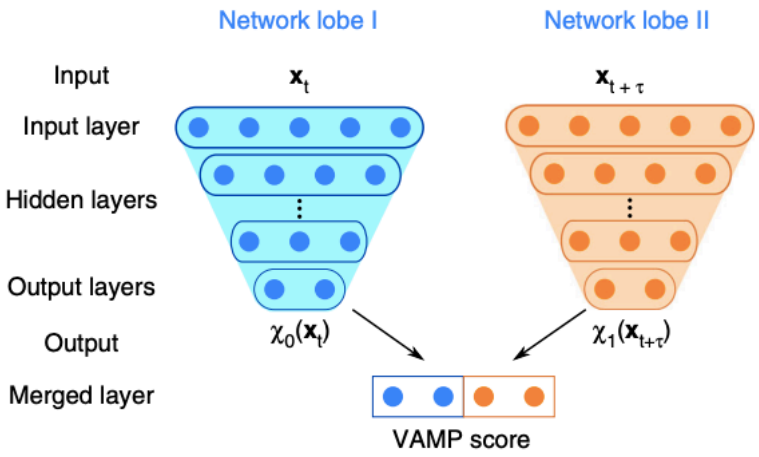


Figure 3.3: Overview of VAMPnets architecture. At each time step t of the simulation trajectory, the 3D coordinates x_t and $x_{t+\tau}$ are input to two deep neural networks that conduct a nonlinear dimension reduction. The lower dimensional outputs are then combined to compute the VAMP score. The negative value of the score is used to train the networks.
Image Source: [35]

In our implementation, the two network lobes are identical clones, the same design choice as Mardt et al. [35]. However, the lobes can also be trained separately. With k output nodes the networks are trained to learn the transformations $f(x_t)$ and $g(x_{t+\tau})$ in Eq.(2.3), where for our implementation $f = g$. Each forward pass of the network is performed with a batch of training data and the loss is then computed by the negative value of the VAMP-2 score.

While a rigorous derivation of the gradient of the loss function can be found in [35], it suffices for our purpose to note that the loss has properties that ensure it is easily differentiable. We rely on PyTorch's automatic computation of gradients of any function computed by the `Autograd Engine`, which leverages efficient computational graphs to compute analytical derivatives using the chain rule, allowing us to easily perform backpropagation in stochastic optimization.

Adam is chosen as an optimizer and is implemented through PyTorch's `torch.optim` package as one of many stochastic optimization algorithms included. Kingma and Ba demonstrate by empirical results that Adam works well in practice and compares favorably to other stochastic optimization methods [28]. The algorithm can be found in Appendix B.1.

We reduce the output dimensions using an MLP after the forward pass to the equivariant neural network. From one scalar per atom or node, we reduce it to the number of predicted metastable states, using a softmax layer as our final activation layer. The output can be interpreted as a probability, and thus the network effectively performs featurization, dimension reduction, and clustering to metastable states.

The process provides a single end-to-end learning framework. An example of the efficiency of end-to-end learning frameworks in this domain is the effectiveness of

AlphaFold2 compared to AlphaFold, where the latter used an intermediary step of computing distance matrix and then used optimization to compute its three-dimensional positions. The former used an end-to-end framework and saw great improvements [26].

While traditional gradient descent trains the network by computing the loss for all training samples and then computing the gradient, stochastic gradient descent approximates the gradient by performing the calculation on a randomly selected subset of the training data, commonly denoted as a batch. This approach reduces the computational burden and scales to large datasets, trading slower convergence for faster iterations. Furthermore, it reduces the risk of getting stuck in local minimas through its stochasticity.

3.2.2 Network Architecture

Our equivariant network architectures are implemented using `e3nn` which builds on PyTorch. `e3nn` is an open-source Python library that allows researchers to conveniently explore the realm of equivariant neural networks for 3D data by providing the tools to create E(3) equivariant neural networks. For a detailed breakdown of the `e3nn` syntax and functionality, see Appendix B.4.

We employ three different equivariant network architectures:

1. A GCNN based on the Simple Network [14]
2. A DP Transformer which is an equivariant version of the original Transformer [61] implemented by Liao and Smidt [32]
3. A GA Transformer based on the Equiformer by Liao and Smidt [32]

All of our equivariant network architectures are concatenated with an MLP that performs feature reduction ending with a softmax layer of which the output can be interpreted as the probabilities of belonging to each metastable state. To ensure equivariance, the features fed to the feature reduction MLP are scalars ($l = 0$). The network scheme is illustrated in Figure 3.4.

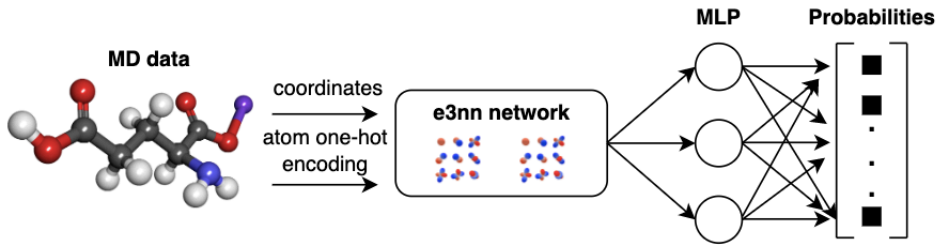


Figure 3.4: High-level scheme of the neural network architecture

To provide implementational details, we have chosen to present each part of the pipeline in the sections below. An overview of the network pipeline is as follows:

1. The neural network is trained using PyG advanced mini-batches, concatenating nodal features and stacking adjacency matrices, allowing the network to process an entire batch in parallel.
2. The adjacency matrix is used to estimate nodal distance vectors by computing the norm to obtain edge lengths and subsequently embedding the edge lengths using a basis of cosines or Gaussians, depending on architecture.
3. The basis of the spherical harmonics up to a specified degree l_{max} is used for encoding directional node attributes.
4. The one-hot encoded nodal information, the edge length embedding, and the directional node attributes expressed using spherical harmonics are all fed to the E(3) Equivariant Neural Network which embeds the input features. This procedure is repeated for the number of layers specified, while the last layer ensures that the ultimate nodal encoding is as specified at initialization.
5. The output of the E(3) Equivariant Neural Network is fed to a simple MLP that performs feature reduction into the desired number of metastable states, using a softmax as the final layer to give a probabilistic interpretation.

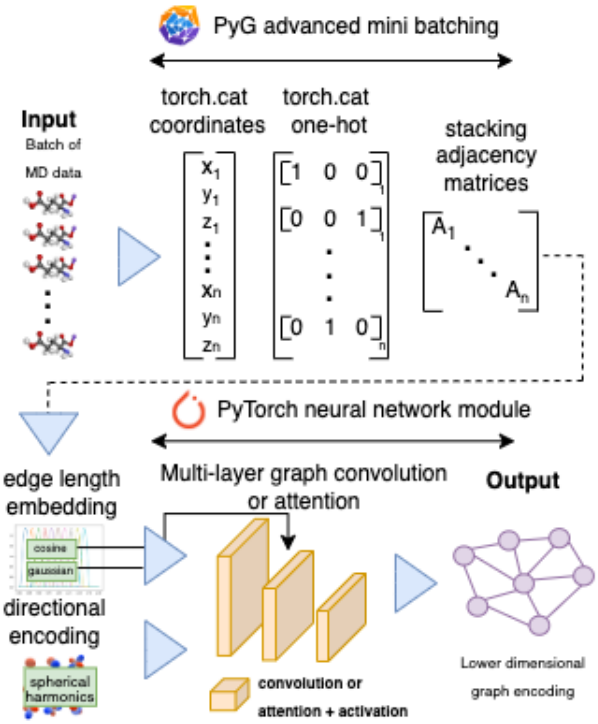


Figure 3.5: High-level scheme of the `e3nn` network, which forms the first block of the entire network pipeline

3.2.2.1 Advanced Mini-Batching

As introduced in Section 3.1, we use PyG's data class to store the data and its data loader class to perform batching. However, what is worth noting is how this

advanced mini-batching is executed. The mini-batching procedure is essential for scaling the training capacity of deep learning models to large amounts of data.

Instead of processing samples individually, mini-batch groups a set of samples into a unified representation to efficiently process them in parallel. In the domain of images or language, this process is usually done by rescaling or padding each example into a set of equal-sized shapes and then grouping them into an additional dimension. However, these methods do not scale well when working with graphical structures, as graphs can hold any number of nodes or edges.

PyG is designed and developed to handle graph neural networks. Hence, it employs an efficient method of batching together multiple graphs in the training data. In PyG, parallelization across the batch is achieved by diagonally stacking adjacency matrices, resulting in a giant graph that holds multiple isolated subgraphs, and simply concatenating node and target features in the node dimension, illustrated as:

$$D = \begin{bmatrix} D_1 & & \\ & \ddots & \\ & & D_n \end{bmatrix}, \; X = \begin{bmatrix} X_1 \\ \vdots \\ X_n \end{bmatrix}, \; Y = \begin{bmatrix} Y_1 \\ \vdots \\ Y_n \end{bmatrix} \tag{3.1}$$

The procedure allows message-passing and attention schemes to function on the giant graph as the subgraphs are not connected and avoid memory overhead as the adjacency matrices can be stored sparsely while avoiding padding.

3.2.2.2 Edge Length Embedding

In contrast to the invariant MLP, the equivariant neural network architectures are fed an embedding of the edge lengths rather than the quantity itself. We embed the distances using a basis function and then feed this embedding to the neural network. The Transformer architectures use a Gaussian basis for the embedding, while the GCNN instead uses cosine.

The practice of using continuous filters was initially proposed by K. Schütt et al. [57] where they proposed a continuous-filter convolutional neural network called SchNet for modelling quantum interactions. In similarity with SchNet, the `e3nn` framework uses these radial model filters.

The set of basis functions is smooth and is strictly zero beyond the chosen max radius (a maximum radius for where nodes in the atomic graph are connected through edges). This is a useful property to obtain smooth operations even though the cutoff at max radius is discrete.

The embedding, returning a projection onto the basis function $\{y_i(x)\}_{i=1}^{N}$, is performed as follows:

$$y_i(x) = \frac{1}{Z} f_i(x) \tag{3.2}$$

Where x is the edge lengths, f_i is the i_{th} basis function and Z is constant normalizing the second moment.

Throughout the thesis, we use a maximum radius of 0.5, we scanned this parameter and found strong results for this value, it is also commonly used in previous research. Figure 3.6 illustrated how this max radius looks for the two different set of bases used in the thesis, cosine and Gaussian.

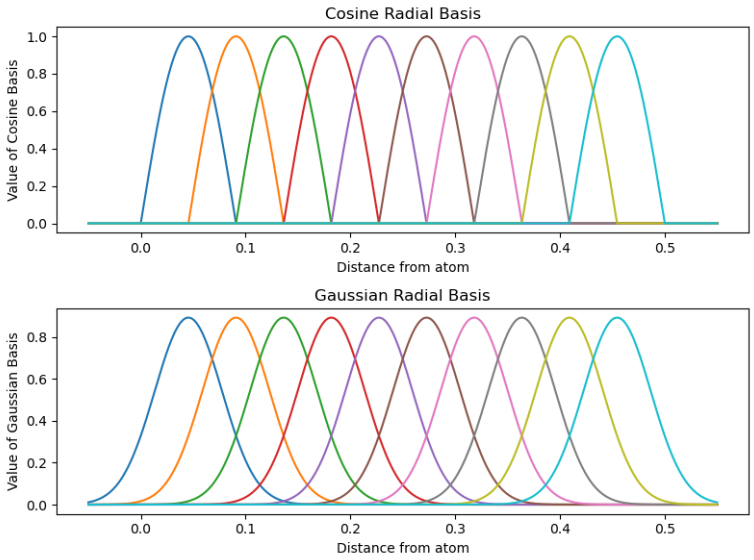


Figure 3.6: Illustration of the difference between a cosine and Gaussian edge length embedding using 10 functions with a max radius of 0.5.

3.2.2.3 Directional Encoding Using Spherical Harmonics

As mentioned in Section 2.3.5, the directional vectors are encoded using the spherical harmonics as basis. The spherical harmonics are functions defined on the sphere and form a basis of the space. Consequently, the spherical harmonics are $E(3)$ equivariant. The directional vectors are encoded using this basis for all equivariant neural networks used in this thesis with corresponding l_{max}. If explicit mention of the l_{max} is omitted, readers are to perceive it as $l_{max} = 2$.

3.2.2.4 Feature Reduction with a Multilayered Perceptron

There was a distinct design choice to avoid any type of average pooling in the final layer of the `e3nn` network. Our implementation of the `e3nn` network instead outputs ξ scalars per node ($\xi = 2$ for the GCNN and 1 for the Transformers), and the forward call is then supplemented by a simple MLP that performs feature reduction into k states, where k denotes the predicted number of metastable states for the protein structure trained on. As the number of nodes and the predicted metastable states vary with protein, we provide an illustration considering alanine dipeptide below:

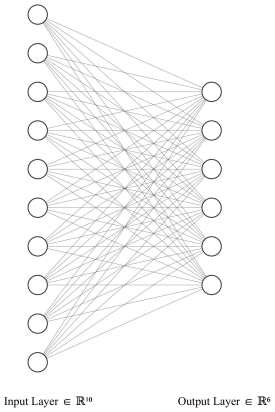


Figure 3.7: Illustration of the architecture used for the multilayer perceptron to conduct feature reduction for alanine dipeptide. Softmax is used as the last activation function to normalize the output of a network to a probability distribution

BatchNorm1d

For the MLP architecture we first use `BatchNorm1d` which applies Batch Normalization over a 2D or 3D input [24].

$$y = \frac{x - E[x]}{\sqrt{Var[x] + \epsilon}} * \gamma + \beta \tag{3.3}$$

The calculation of the mean and the standard deviation is performed per dimension across mini-batches. γ and β are vectors of learnable parameters with size C, where C represents the number of features or channels of the input data. By default, the elements of γ are initialized to 1, and the elements of β are set to 0. The standard deviation is computed using the biased estimator, which is equivalent to `torch.var(input, unbiased=False)`.

We use the default parameters for all our Batch Norm layers, this is entails a momentum of 0.1.

ELU Hidden Layer

The benchmark MLPs have 4 hidden layers with intermediary activation functions being the Exponential Linear Unit (ELU), with input and output nodes equal to 20. This was chosen in accordance with Mardt et al. [35] to benchmark against their work. ELU is a popular activation function, and for $\alpha < 0$ it is described as:

$$\begin{aligned} &f(x) = x \text{ if } x > 0 \\ &\alpha(\exp(x) - 1) \text{ if } x \leq 0 \end{aligned} \tag{3.4}$$

It is similar to the Rectified Linear Unit (ReLU) function but has a smooth transition for negative inputs. Unlike ReLU, ELU can produce negative outputs, which helps

to reduce mean unit activations and speed up learning. This reduction in mean unit activations is similar to batch normalization but with lower computational complexity. Mean shifts toward zero speed up learning by bringing the normal gradient closer to the unit natural gradient because of a reduced bias shift effect.

In comparison, other activation functions like Leaky ReLU and Parametric ReLU also have negative values but do not provide a robust deactivation state like ELU. ELU saturates to a negative value for smaller inputs, which decreases the forward propagated variation and information.

Softmax Activation Function

In order to consider the output as a probability vector, we apply the softmax activation function to the final layer. The softmax function $\sigma : \mathbb{R}^k \to (0,1)^k$ is defined by the formula:

$$\sigma_i(z) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \tag{3.5}$$

The function applies the standard exponential function to each element z_i of the input vector and normalizes the values by division with the sum of the exponential of each z_i. Thus, the output can be interpreted as a probability vector, as its sum is 1.

3.2.3 Experimental Models

Several experimental models are trained and tuned on the alanine dipeptide dataset to select an optimal model for the protein folding data. Amongst these experimental models, we choose the most promising one based on validation accuracy and data robustness. We validate the model using Chapman-Kolmogorov and Implied Timescales tests. Subsequently, we use the model for the protein folding dataset.

Model	Description
Multilayered Perceptron, Distance	MLP using BatchNorm1d, and 4 hidden layers with 20 nodes each and ELU activation. Softmax is used as a final activation layer. Input data is based on the pairwise distance between the heavy atoms, with a matrix shape of (10×10), which is flattened.
Multilayered Perceptron, XYZ	Design of neural network is equivalent to the MLP above but does not preserve invariance as the input data is based on the x,y,z coordinates of the heavy atoms.
Graph Attention Transformer	Equivariant Transformer using MLP attention with non-linear message passing followed by a feature reduction MLP. Model is tested for $l_{max} \in \{0, 1, 2\}$.
Dot Product Transformer	Equivariant Transformer using dot product attention with linear message passing, followed by a feature reduction MLP. Model is tested for $l_{max} \in \{0, 1, 2\}$.
GCNN	Equivariant graph convolutional network, followed by a feature reduction MLP. Model is tested for $l_{max} \in \{0, 1, 2\}$.

Table 3.3: Configurations of experimental models tested on the alanine dipeptide dataset

3.3 Algorithm for Detecting Metastable States

While the toy example of alanine dipeptide is a well-studied molecule with the number of metastable states well established, deploying our model in a novel context requires a systematic way to identify the number of metastable states of a molecule. For this reason, we have developed a non-parametric algorithm that leverages our VAMPnets architecture to estimate the number of metastable states.

Algorithm 1 Detecting the number of metastable states for an arbitrary molecule using MD data trajectories.

Require: k : initial guess of the number of states
Require: ψ : proportion threshold
Require: N : network
Require: V : vampnet implementation
Require: D : MD data
Require: λ : lower interval of binary search
Require: υ : upper interval of binary search
Require: τ : time-lag of data
Require: $**kwargs$: hyperparameters of the network
  while $\upsilon - \lambda > 0$ **do**
    $\mu \leftarrow (\upsilon - \lambda)//2$ (Set mid point of interval)
    $N \leftarrow N(**kwargs, k)$ (Initialize network with μ output states)
    $V \leftarrow V(N)$ (Initialize VAMPnet using the specified network)
    $V \leftarrow V.fit(D)$ (Fit the VAMPnet to the provided MD data)
    $\xi_\mu \leftarrow V.val$ (Store validation score)
    Repeat procedure for υ output states
    $\xi_\upsilon \leftarrow V.val$ (Store validation score)
    if $\xi_\upsilon/\xi_\mu - 1 > \psi$ **then**
      $\lambda \leftarrow \mu$
    else
      $\upsilon \leftarrow \mu$
    end if
  end while
  $x, y \leftarrow V.transform(D)$ (Compute inference on data)
  $K \leftarrow koopman\_matrix(x, y)$ (Compute the Koopman matrix)
  $\Lambda \leftarrow torch.linalg.svdvals(K)$ (Compute singular values)
  $n \leftarrow \text{sum}(-\frac{\tau}{|log(\Lambda)|} \geq \tau)$ (Count the number of singular values above the time-lag)
  Return n

3.4 Downsampling Procedures

3.4.1 Random Downsampling

Random downsampling is performed by randomly selecting indices at each downsampling step such that only $1 - \alpha$ data remains, where α represents the downsampling share. The models are then trained and tested on the reduced data sample. We make sure that the data partitioning is the same for the equivariant models and its benchmark in order to provide a fair evaluation.

3.4.2 Systematic Downsampling

Systematic downsampling instead removes one metastable state. First, a pre-trained model is allowed to assign the data to its states. Next, each of the assigned states is

removed once, letting an untrained model learn from the data missing a state and being validated on the full dataset. While the model is trained on data missing a metastable state, the number of states that the model is forced to predict is still equal to the true number of metastable states. In this manner, we explore if the model can accurately learn to predict metastable states that it has not witnessed during training.

To quantitatively measure the results of the systematic downsampling and compare them we use two measures:

1. Kullback-Leibler Divergence [30]
2. Shannon Entropy [58]

We use the Kullback-Leibler divergence [30] to examine how much the predictions on test data by the models trained on systematically downsampled data differ from the predictions made by the reference model, trained on all the data. The Kullback-Leibler divergence provides an asymmetric statistic distance measure between two distributions and is defined as:

$$D_{KL}(P||Q) = \sum_{x \in \chi} P(x) \log(\frac{P(x)}{Q(x)}) \tag{3.6}$$

The Shannon Entropy [58] is used to measure the information or the uncertainty in the outcomes of a random variable. We use this to determine the uncertainty in the state probability assignments for the models. The Shannon Entropy is defined as:

$$H(X) = -\sum_{x \in \chi} P(x) \log(P(x)) \tag{3.7}$$

Note that we use the natural logarithm to calculate the Kullback-Leibler divergence and log_2 for the Shannon Entropy due to convenience with easier integration to PyTorch.

As there is no canonical ordering of states, the state indexation will occur randomly at initialization. Thus, we must align the indices to a common ordering for the assigned probabilities to be comparable. For details and verification on this procedure, we refer the reader to Appendix A.3.

These are the four different models involved in the systematic downsampling experiment:

1. Reference model: A DP Transformer ($l_{max} = 2$) trained on the entire dataset. This is used to assign states for removal and comparing various properties.
2. Benchmark model: An Invariant MLP trained on the systematically downsampled data.
3. Distinguishable model: A DP Transformer ($l_{max} = 2$) that distinguishes between atom types in its feature encoding

4. Indistinguishable model: A DP Transformer ($l_{max} = 2$) that does not distinguish between atom types in its feature encoding, i.e. all atoms have a unique encoding

3.5 Training and Testing Procedures

In order to optimize hyperparameters we use the platform Weights & Biases (W&B). This platform enables the user to carry out multiple sweeps in an effort to locate the optimal hyperparameters. However, unlike grid search, W&B is capable of terminating sweeps with subpar parameters, resulting in a more efficient search process. A dictionary that specifies discrete or range-based values contingent on the nature of the parameter is provided for each search.

For details of the exact hyperparameters used as well as the details of the training setup, we refer to Appendix C.

The performance measure throughout this thesis is the validation VAMP-2 score. The negative VAMP-2 score is the loss used to train the neural networks. Additionally, the models are evaluated by how many backward passes are required to reach VAMP-2 score convergence and its computational time per epoch.

All validation scores are calculated using k-fold cross-validation. This entails randomly dividing the data into k-folds. The model is trained on $k-1$ of the k folds, and the last partition is used for validation. This validation is performed k times, leaving out a different partition for validation each time. The reported results are always produced in this manner. k is chosen to produce a reasonably small confidence band, and we use $k = 5$ throughout the thesis.

The values presented are averaged over the five folds with 95% confidence intervals included for the validation score. The error bands throughout the thesis are calculated in the same manner. The standard deviation of the results from the cross-validation is computed and multiplied with the appropriate constant $\lambda_{0.95}$ from the standard normal distribution.

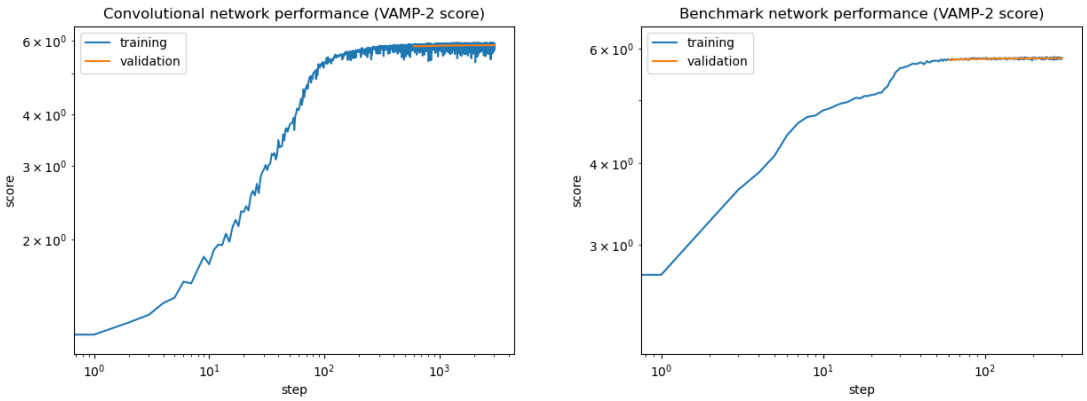
4

Results

To begin with, we present our results on the alanine dipeptide data for our experimental models. Next, we display the validation results for the best-performing model, as this is the model we ought to use for the protein folding dataset. Lastly, the results for the selected model on the protein folding dataset are presented.

4.1 Alanine Dipeptide

Notice, when applying the GCNN and the MLP to the complete trajectories of alanine dipeptide, a VAMP-2 score of 6.0 is achieved. This result is the theoretical maximum for alanine dipeptide which has $k = 6$, where k is the number of metastable states for a given molecule.



(a) e3nn network (b) MLP

Figure 4.1: Alanine dipeptide training results when using the entire dataset, including hydrogen atoms. Note: The model quickly converges to the theoretical maximum VAMP-2 score of 6.0.

Going forward, only the heavy atoms are included in subsequent experiments, removing the hydrogen atoms of the molecules.

4.1.1 Model Benchmark Results

Model Configuration	Number of Parameters	Seconds / Epoch	Convergence Steps	Validation Score
MLP, Distance	4026	7.4	226.6	**4.56** ±0.01
MLP, XYZ	2486	8.0	165.4	**4.56** ±0.02
GA Transformer ($l_{max} = 0$)	70965	64.4	264.8	4.47 ± 0.03[1)]
GA Transformer ($l_{max} = 1$)	82821	107.6	283.6	4.39 ± 0.04[1)]
GA Transformer ($l_{max} = 2$)	102861	303.6	96.8	**4.55** ±0.02
DP Transformer ($l_{max} = 0$)	72149	68.3	447.6	4.22 ± 0.05[1)]
DP Transformer ($l_{max} = 1$)	84005	115.1	111.8	**4.55** ±0.02
DP Transformer ($l_{max} = 2$)	105581	318.4	99.4	**4.56** ±0.02
GCNN ($l_{max} = 0$)	5850	52.7	Non-convergent	4.19 ± 0.12[1)]
GCNN ($l_{max} = 1$)	11150	62.2	Non-convergent	4.21 ± 0.10[1)]
GCNN ($l_{max} = 2$)	21890	79.3	Non-convergent	4.05 ± 0.08[1)]

[1)] Confidence interval disregards outliers

Table 4.1: Overview of model performance results. 'Convergence Steps' details the number of backward passes required to achieve a VAMP-2 training score of 4.50 and 'Number of Parameters' the number of trainable parameters in the network.

Based on the benchmark results above, we can see that the GA and DP Transformer both perform well. The GA Transformer has slightly quicker computational time and convergence speed.

Henceforth, we will use the DP Transformer with $l_{max} = 2$, as it provides consistent results. We select this model for further validation, using Chapman-Kolmogorov and Implied Timescales plots. While on the protein folding data the GA Transformer with $l_{max} = 2$ will also be included for further comparability.

4.1.2 Model Validation Results

We leverage Implied Timescales plots and the Chapman-Kolmogorov test to validate the chosen model. Results for these tests can be viewed below for each corresponding section.

4.1.2.1 Implied Timescales

The implied timescale plots should be flat for a region where the model is approximately Markovian.

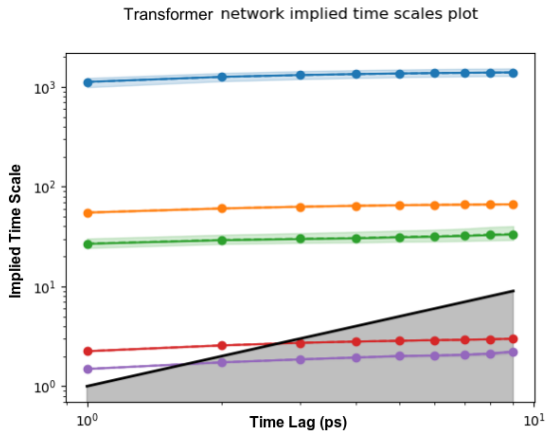


Figure 4.2: Implied timescales plot for our model on alanine dipeptide. Lines should be flat in regions where the model behaves close to what is expected from a Markovian model

4.1.2.2 Chapman-Kolmogorov

A Chapman-Kolmogorov test is a way to verify that the model obeys the dynamics that are assumed for its construction. In this case, the model should follow the lines predicted by the Chapman-Kolmogorov equation for a good fit. From the implied timescale plot above, we deduce that $\tau = 2$ is a suitable lag time. Thus, we compute the Chapman-Kolmogorov test using this lag time.

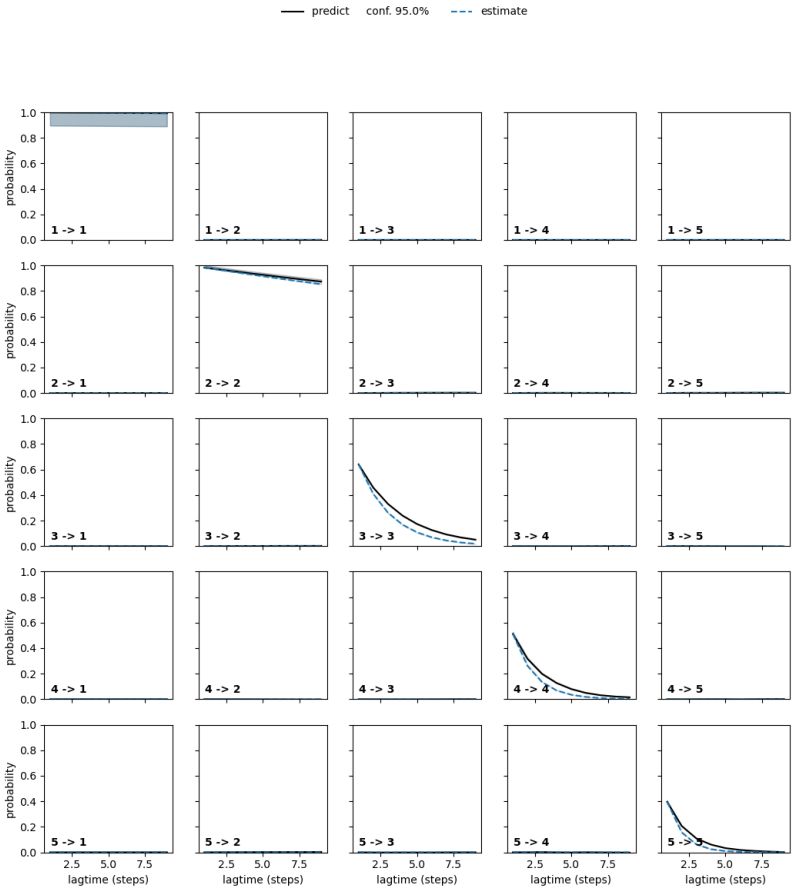


Figure 4.3: Chapman-Kolmogorov test for our model on the alanine dipeptide molecule. A good model is expected to follow the lines implied by the Chapman-Kolmogorov equation.

4.1.3 Detecting Metastable States With Selected Model

In order to detect the meta-stable states of alanine dipeptide accurately, a high VAMP score is required. An illustration of our model's predicted meta-stable states is shown below in the plot. The brightness of the cells represents the probability assigned to that state by the model.

Using a software package that provides web-based molecular graphics for large complexes, introduced by [53], we can get a sense of how each of the predicted meta-stable states is connected to the alanine dipeptide molecule.

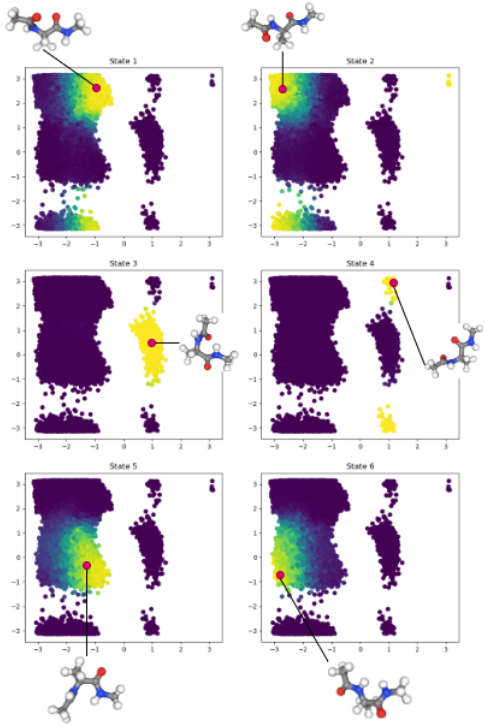


Figure 4.4: Illustrating the alanine dipeptide molecule structure for metastable states

As illustrated by the molecules, the dihedral angles differ greatly between the identified meta-stable states.

These 'representative' molecules were chosen by finding the frames of the molecule where the network predicted with the highest confidence that the molecule belonged to a particular state. Thus, model's best bet for each state is included. Below is a table illustrating the probabilities assigned to each state for the representative frames.

State	Index	Probability assigned
$k = 0$	587,638	0.97
$k = 1$	643,837	0.99
$k = 2$	656,737	0.98
$k = 3$	21,063	1.00
$k = 4$	190,195	0.99
$k = 5$	667,977	1.00

Table 4.2: Output of the predictions assigned the highest probability for each metastable state and the corresponding frame indices

4.1.4 Algorithm Verification

In this section, we present the results of our metastable states detection algorithm on alanine dipeptide. The theoretical number of metastable states for alanine dipeptide is already known. Hence, the results serve as a validation of our algorithm's performance on an arbitrary given peptide structure. The first step in investigating a peptide molecule with our framework is to establish the number of metastable states the molecule exhibits. Only then can we detect the metastable states of an MD trajectory.

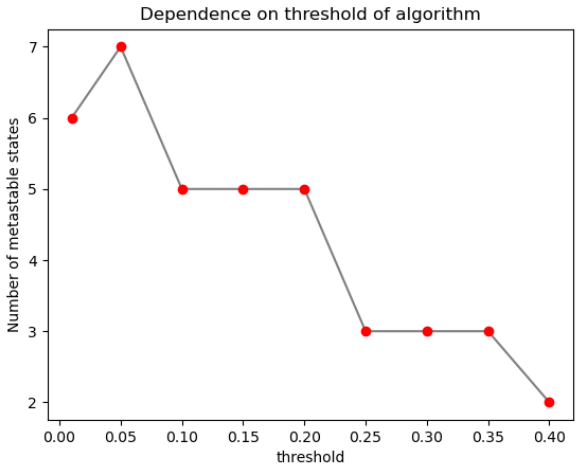


Figure 4.5: Illustration of importance of VAMP-2 score change threshold on number of metastable states detected using our proposed algorithm.

4.1.5 Random Downsampling

By randomly downsampling the data, we can examine the data robustness of our network compared to the benchmark.

Model	50%	10%	5%	1%
MLP, Distance	4.36 ± 0.41	3.57 ± 0.05	2.88 ± 0.21	1.74 ± 0.11
MLP, XYZ	4.55 ± 0.02	3.58 ± 0.07	3.16 ± 0.15	1.59 ± 0.15
DP Transformer ($l_{max} = 2$)	4.53 ± 0.04	3.82 ± 0.31	3.46 ± 0.12	2.82 ± 0.40

Table 4.3: VAMP-2 score for the different models at different levels of remaining data share

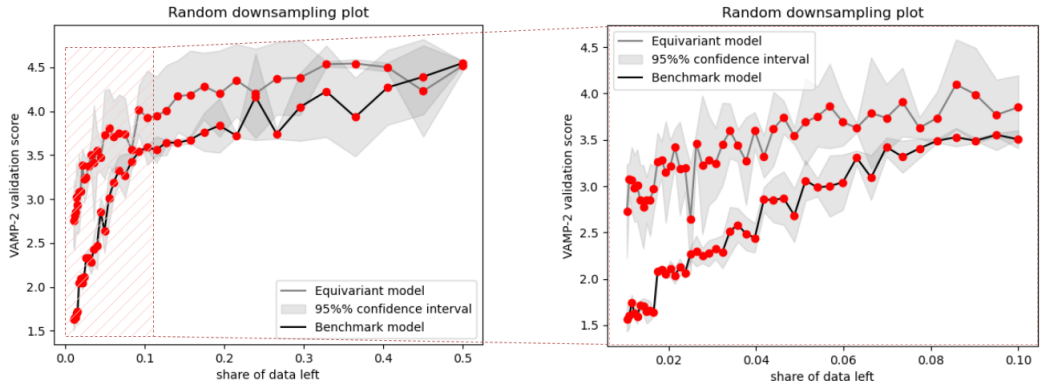


Figure 4.6: Random downsampling plot displaying model performance vs benchmark for varying levels of data remaining. Note: figure on the right examines the shaded region further.

4.1.6 Systematic Downsampling

The systematic downsampling procedure systematically removes one metastable state, as assigned by a reference model, and trains the other models on the down-sampled data. This is performed for each of the 6 states of alanine dipeptide. The models are then tested on data containing the, during training, unseen state.

4.1.6.1 Number of Frames Removed

To get a sense of the size of each metastable state, we plot the number of frames removed in systematic downsampling, these are the number of frames assigned to each state by the reference model.

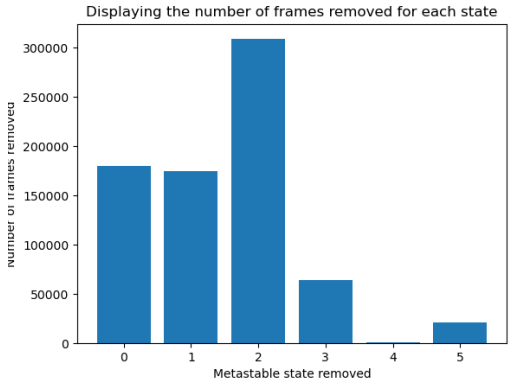
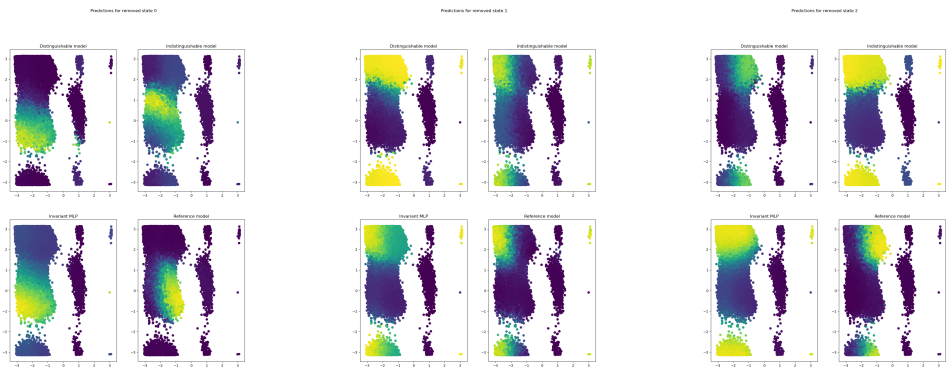


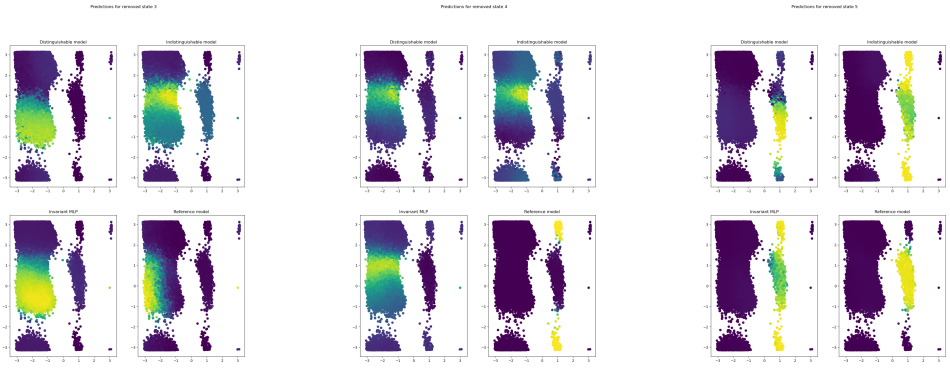
Figure 4.7: Number of frames removed for each of the 6 metastable states.

4.1.6.2 Predicting the Removed State

We examine the ability of the models to accurately locate the removed state by a dihedral plot colored by the probabilities assigned to the removed state.



(a) State 0 (b) State 1 (c) State 2



(a) State 3 (b) State 4 (c) State 5

Figure 4.8: Figure displays the dihedral plot colored by the probability assigned to the removed state for each model. The top left is the distinguishable model and the top right is the indistinguishable model. The bottom left is the invariant MLP while the bottom right is the reference model. Note: the reference model should be regarded as the accurate prediction as this model has been exposed to the state during training.

4.1.6.3 Kullback-Leibler Divergence

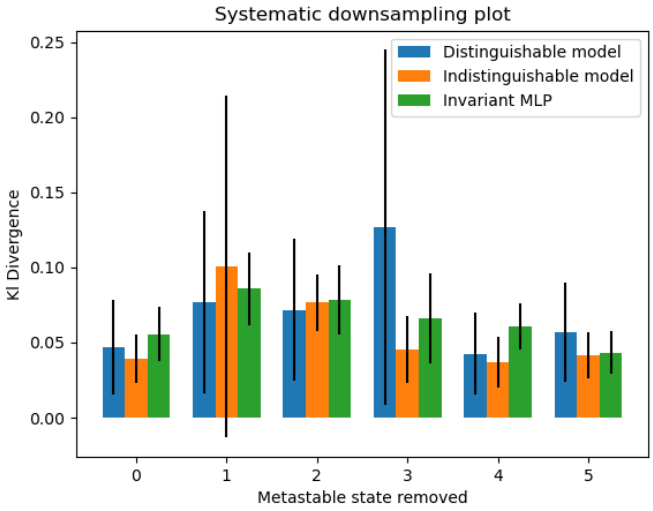


Figure 4.9: Kullback-Leibler Divergence for the different models. They were trained on data with one state removed and then tested against the reference model trained on the full data.

4.1.6.4 Shannon Entropy

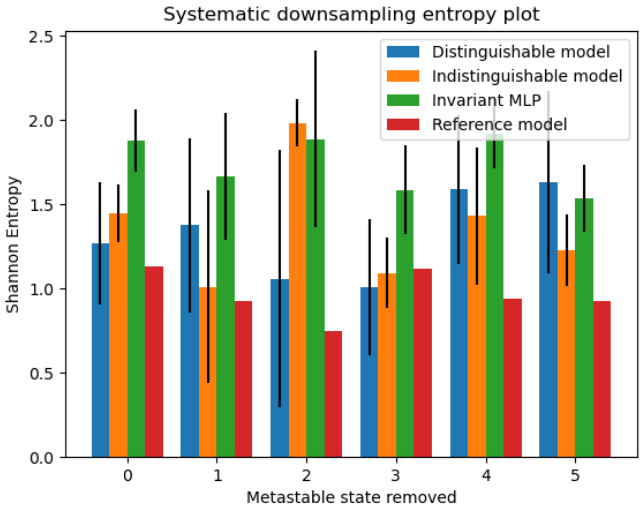


Figure 4.10: Shannon Entropy for the different models.

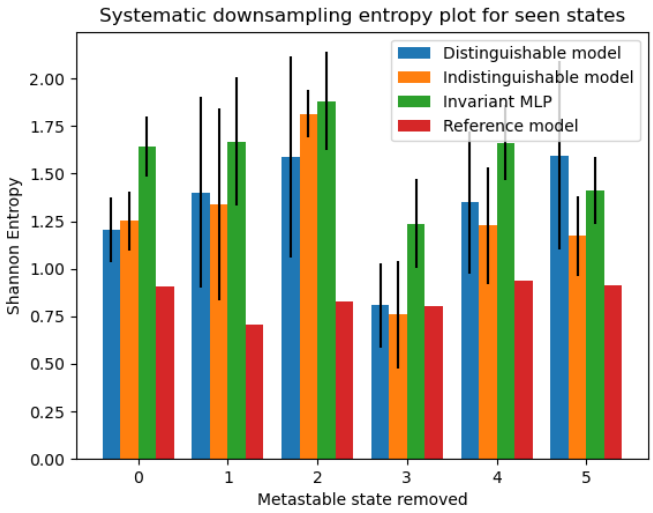


Figure 4.11: Shannon Entropy for the different models not including the unseen state, i.e. clipping the vector to remove the unseen state and re-normalizing.

4.2 Protein Folding

The following section details the results obtained on the protein folding dataset.

4.2.1 Detecting the Number of Metastable States

We use the algorithm we developed previously to detect the number of metastable states for each of our 12 proteins. We chose to use the VAMP-2 score change threshold to be 20% based on the results in Section 4.1.4. The number of metastable states detected as well as the time-lag used can be viewed Table 4.4, the time-lag was derived from the implied time scale plots shown in Appendix A.2.

Protein	Heavy Atoms	Time-lag (ns)	Detected Metastable States
BBA (1FME)	251	20	8
Villin (2F4K)	287	40	7
Trp-cage (2JOF)	144	100	7
BBL (2WAV)	352	500	8
a3D (A3D)	572	1000	7
Chignolin (cln025)	93	10	3
WW domain (GTT)	287	150	7
NTL9 (2HBA)	301	160	7
Protein G (NUG2)	435	500	8
Protein B (1PRB)	362	250	7
Homeodomain (UVF)	466	250	8
λ-repressor (1LMB)	618	100	8

Table 4.4: Metastable states detected for protein folding dataset

4.2.2 Benchmarking Results

After identifying the number of metastable states per protein, we test the VAMP-2 score of the two different equivariant Transformer structures ($l_{max=2}$) against the MLP benchmark on each protein of the proteins for which the models can be run in reasonable time on the complete data. Proteins requiring too many computational resources for benchmarking are excluded.

Protein	MLP xyz	DP Transformer	GA Transformer
2F4K	3.31 ± 0.22	**4.25**±0.20	**4.35**±0.04
2JOF	3.97 ± 0.11	**4.81**±0.22	**4.66**±0.05
cln025	**2.13**±0.01	**2.10**±0.07	**2.06**±0.02
PRB	**2.65**±0.16	**2.90**±0.23	**2.81**±0.03

Table 4.5: Protein folding VAMP-2 score benchmark results

5

Discussion

In this Chapter, we discuss the empirical results obtained in Chapter 4 to provide further insights and propel future research. This discussion includes a comparison between the different model architectures in conjunction with a broader analysis of the implications, strengths, and limitations of the various model architectures. Finally, this chapter includes suggestions for future research.

Research Context: The methods developed in this thesis build on previous work in two separate domains. Firstly, it utilizes the contributions made to VAMPnets and conformational dynamics by [35, 36, 42, 68]. Secondly, it leverages the technical and theoretical strides made in popularizing equivariant neural networks, especially for atomistic systems, by [2, 12, 14, 32, 60, 65]. This thesis unites the two research areas by implementing VAMPnets trained by equivariant neural networks. In doing so, it provides both a technical (through the free distribution of our source code on Github) and a theoretical foundation for future research and industrial development on the topic. While incurring a significantly higher computational burden, equivariant neural networks provide increased data efficiency, convergence speed, and robustness to the training of VAMPnets. In addition, it provides an initial attempt at systematizing the detection of the number of metastable states of molecules by the implementation of our algorithm.

Benchmark Results: The results obtained on the protein-folding data (Section 4.2.2) differ from those obtained on the alanine dipeptide molecule (Section 4.1.1) in that the equivariant models significantly outperform the benchmark in maximum VAMP-2 score on the protein folding data, while at convergence being equal for the alanine dipeptide molecule. The difference is likely due to the improved data efficiency of the equivariant models, not an enhanced accuracy at convergence. Due to limited computing resources, and the exponential increase in compute required for larger molecules, the results are on the protein folding data are for 1 epoch of training, likely not reaching convergence for all the models, while the results on alanine dipeptide trained until convergence (only requiring 5 epochs). Thus, the protein-folding results confirm the greater data efficiency of the equivariant models found on alanine dipeptide but not the claim of superior accuracy. However, it is possible that the equivariant models display greater accuracy at convergence on the more complex proteins, but could not be tested due to a lack of computing resources.

Domains of Equivariant Superiority: While the equivariant models did not achieve significant improvements to the benchmark model in terms of convergent results, this could have been the case had the dataset been more diverse and required the training of VAMPnets to generalize between molecules. This thesis deals with single molecule data observed at multiple timeframes, this does not force the model to generalize well to other molecules during training. Equivariant neural networks have proven to produce state-of-the-art results for predicting force fields and energies on the $QM9$, $MD17$, and $OC20$ datasets [12, 32]. While it is a different type of task, it indicates that the equivariant model's advantages are enhanced on diverse datasets, forcing the model to learn higher-order symmetries that generalize between molecules.

Algorithm: The proposed novel algorithm provides a straightforward and quantitative way to detect the number of metastable states for any molecule, in contrast to prior methods which rely heavily on expert opinions [17, 35, 36, 42, 68]. The proposed algorithm thus frees up time and limits subjectivity, enabling the analysis of vastly more as well as novel molecules. Interestingly, when tested on alanine dipeptide, the algorithm provides the consensus answer of the molecule having 6 metastable states [4, 25] when omitting the procedure of removing the states for which the implied timescale is below the time lag. However, we still included this algorithmic step for it being a logically consistent and not wanting the influence of one molecule (alanine dipeptide) to out-weight this logical consistency. This proved useful as when ran on the protein folding data we achieved results similar to previous results on those proteins that we found had previously been studied. For instance, on Chignolin (cln025), the algorithm detects 3 metastable states, the consensus correct answer for the molecule, on Trp-cage (2JOF) our algorithm also agreed with previous research, the same for Protein-G (NUG2), WW domain (GTT) and Homeodomain (UVF) [56]. These were all the proteins we found prior estimates for. Lastly, the number of metastable states is not a rigorously defined concept. The timescale on which something is considered to be metastable is arbitrary, and thus, the algorithm can be said to produce a result consistent with our definition.

Stability Issues: The results show that the GCNN architecture (as proposed in the Simple Network by [14]) suffer significant stability issues. While at times converging very rapidly to a high VAMP-2 score, there are times when the model is simply stuck at a lower level, regardless of the number of epochs it is allowed to run. Intrigued by this phenomenon, we sought to study how the initialization affected the stability of the model (Appendix A.1). However, we could not find any systematic differences between the strong and weak models in their initial weights using PCA. The equivariant tranformer architectures (as proposed by [32]) do not seem to experience this issue. While for larger parameter models (1,000,000+ trainable parameters) the models could never seem to converge to a high VAMP-2 score on alanine dipeptide, when reducing the number of trainable parameters, the equivariant Transformers provided consistent high results. This was done without relying on a complex MLP to perform the feature reduction, indicating that the equivariant Transformer embeddings can accurately learn the conformational dynamics of the

molecules.

Strength of MLPs: A general theme throughout this thesis is the strength of the simple MLPs. They provide consistently strong results without any stability issues, though being less data efficient and having slower convergence speed. It seems as though the task of training a model to detect the metastable states of a single molecule with abundance of data does not warrant the introduction of much more complex neural network architectures; the task appears quite simple. However, if data is sparse or the task involves deeper learning that generalize between molecules it is likely that this added complexity is warranted.

Contrasting to previous VAMPnets Papers: The most recent paper exploring training VAMPnets with a novel neural network architecture is the GraphVAMPNet paper by Ghorbani et al. [16]. While studying similar protein folding data as in this thesis, the authors have chosen to present results not directly comparable to ours. Following the same procedure as Mardt et al. [35], they do not report the VAMP-2 score, but rather the state assignments and free energy landscape. As such, it is hard to quantitatively benchmark our network against theirs. A solution to this could be to use the source code reported by the paper and implement the model for direct comparison, as we did for Mardt et al.'s [35] model, however, we did not have the time to follow through with such ambitions.

Comparing the GA and DP Transformers: Comparing the GA Transformer and the DP Transformer, MLP attention does not provide a significant improvement compared to DP attention on the alanine dipeptide dataset, as can be seen in Figure 4.1, similar to the findings of [32]. This is likely due to the low complexity of the alanine dipeptide molecule, containing few atoms with a small diversity in type, making linear attention sufficient. However, on the more complex protein folding dataset, the MLP attention provides significantly more stable results.

Improving the Loss Function: As seen from the results in Section 4.1.6, the models quite confidently predict the wrong metastable state, as indicated by a low Shannon entropy combined with a high KL divergence. To resolve this issue, it is possible to augment the loss function, appending an entropy term to the VAMP-2 score, penalizing high confidence predictions that are wrong. Additionally, Section 4.1.6.2 shows that all models do quite a good job at finding the densely populated states, even when not included in the training set, though none of the models can accurately detect the less populated states.

Practical Implications: While this thesis leverages simulation data and does so in a sandbox setting, there are plenty of practical applications which derive use from our research findings. Proteins are important targets in drug discovery. AlphaFold2 now allows us to get access to low-energy structures of proteins routinely [26]. However, proteins are flexible and adopt multiple different meta-stable structures, all potentially important for biological function. Consequently, detecting and enumerating these metastable protein structures allow us to target proteins more

specifically and directly, and with this thesis we are contributing to detecting and enumerating these metastable protein structures. We have shown that the detection can be made in a more data efficient manner and provided a novel algorithm for the enumeration. Practical use-cases of this thesis could possibly include:

1. Drug-discovery research using neural networks, which could be compelled to switch to equivariant neural networks following the results of our research, provided, that the computational complexity is not detrimental to their operations.

2. Detection of protein folding patterns could be improved by the VAMPnets structure overall, providing an end-to-end learning framework and removing much of the expert labor involved. In addition, the introduction of equivariant neural network to this framework could prove advantageous to provide more data efficient learning.

5.1 Limitations

This section provides an overview of the most important limitations imposed by our thesis and a description of their origins.

Neural Network Architectures: A significant limitation of this thesis is the neural network architectures considered. Only three distinct equivariant neural network architectures are used in this thesis, and the benchmark is of significantly lower complexity than the equivariant models (the largest equivariant model used has around 25 times more trainable parameters than the largest benchmark model). This has introduced a heightened level of ambiguity into the outcomes, thereby attenuating the overall conclusiveness of the findings as it is hard to establish what performance increases are attributable to the inclusion of higher-order irreps features and what is attributable to smarter model architectures. We attempt to resolve this by varying the l_{max} while keeping the model architecture constant. However, the large dependency on the hyperparameters makes this a difficult endeavor.

Hyperparameter Tuning: For comparable results for different l_{max} there needs to be separate hyperparameter tuning for each degree. Due to lack of time, we simply truncate the feature embeddings and removes the higher order representations. This procedure potentially favors the higher-order representations as these were the ones used for hyperparameter tuning.

Compute Intensity: A limitation that is apparent, and further displayed in the results of this thesis, is the common limitation of all neural network architectures based on higher order irreps, namely, the compute intensity. The results display that the equivariant Transformers are, on average, approximately 1,000 times slower than the MLP on the protein folding data. The operation that drives the computing complexity is the tensor product. One reason why tensor products can be computationally expensive is that the kernels for these operations in common libraries, like

PyTorch [46], have yet to undergo extensive optimization and customization, unlike other operations. However, this problem relates to the software, not the network architecture employed.

Compute Resources: The computational complexity of the neural network architectures employed leads to limitations in the experimental results. This thesis does not compute very fine confidence intervals as there is a limitation on access to computing resources. Consequently, this would enhance the significance of the results and enable benchmarking on many of the larger protein folding datasets. Furthermore, the large confidence bands limit inhibit any strong conclusions from being drawn from Section 4.1.6, where additional compute could have provided more substantial results.

5.2 Future Research

This thesis leaves plenty of room for future research to explore the details of the topics discussed. We have compiled a list of potential study topics below, along with advice for those bold enough to pursue them.

Study l_{max} Dependence: An important contribution gap left is a fair and systematic comparison of the same model architecture for different l_{max}. While this is attempted in the thesis, there are plenty of confounding variables limiting the significance of the results on this particular topic. This involves finding better ways of hyperparameter tuning for these models and applying this to each of the degrees of the model independently.

Improve the Algorithm: The detection of the number of metastable states in an arbitrary molecule is a precursor to deploying VAMPnets effectively. While this thesis provides a rudimentary algorithm to do just that, we encourage future researchers to refine this algorithm and systematically deploy it on a large range of molecules. We started with an algorithm solely reliant on the singular values of the Koopman matrix but could not achieve consistent results. However, we urge others to examine this more carefully as well as get inspired to test radically different methods for the crucial task of enumerating the metastable states of proteins.

Generalize between Molecules: Future research could include analyzing the generalization of these VAMPnets trained by equivariant models, training multiple VAMPnets together on diverse datasets. Additionally, a paper implementing additional architectures, such as the GraphVAMPNet presented by Ghorbani et al. [16], and comparing them all in a systematic fashion would serve an important role in establishing what model architectures work well for various tasks.

Enhance the Compute Efficiency: We encourage further research on improving the computational efficiency of irreps' tensor products to enable more wide-scale adoption of equivariant neural networks and dissemination of their benefits. This

work is likely to build upon the frameworks developed by [14] and the speed-ups achieved along the way by works such as [12].

6

Conclusion

This chapter provides clear answers to the proposed research questions as well as a compilation of various other conclusions that were drawn during the course of the thesis.

6.1 Answers to Proposed Research Questions

Do the molecular physical symmetries encoded by E(3) Equivariant Networks improve prediction accuracy, robustness, or data efficiency in VAMPnets training compared to an MLP encoding?

The use of E(3) equivariant neural networks in training VAMPnets is shown to significantly improve the prediction accuracy on random downsampled data. Using only 1% of the dataset, the DP Transformer ($l_{max} = 2$) achieves almost twice the VAMP-2 score of the benchmarks. Furthermore, the model exhibits improved robustness. With only 20% data remaining, the model scores on par with the complete dataset. On average, the model requires significantly fewer backward passes, converging more than twice as fast as the benchmark models, showing enhanced data efficiency.

Do VAMPnets built on E(3) Equivariant Networks demonstrate improved accuracy in detecting metastable states in protein folding datasets compared to state-of-the-art implementations?

The results of this thesis do not conclusively show a greater ability to detect metastable states once convergence has been reached. Further research on diverse datasets and complicated molecules is required for a conclusive answer to this question.

6.2 Other Conclusions

1. The novel algorithm proposed provides a straightforward and efficient way to enumerate the metastable states of an arbitrary molecule. The algorithm shows promising results, agreeing on the consensus estimate on Chignolin and enumerating all but the least populated state on alanine dipeptide.

2. The proposed GCNN architecture suffers severe stability issues, providing an opportunity for research to study how the stability could be improved.

3. Due to not performing hyper-parameter optimization for each l_{max}, it is difficult to draw conclusions from their relative performance.
4. Equivariant neural networks, as constructed with today's state-of-the-art methods, incur significant computational costs, scaling poorly with increase in molecule size. For example, the equivariant Transformer architectures were approximately 1,000 times slower than the MLP on the protein folding dataset. This computational burden puts a big strain on the GPU, limiting batch size and requiring the use of gradient accumulation.
5. MLP attention does not provide a significant improvement compared to DP attention on the alanine dipeptide dataset, likely due to limited complexity making linear attention sufficient.
6. Potential to improve the loss function by adding an entropy term to the VAMP-2 score, penalizing high confidence predictions that are wrong. However, inducing further parameter tuning as it an additional penalizing factor is required.

Bibliography

[1] Christian B. Anfinsen. "Principles that govern the folding of protein chains". In: *Science (New York, N.Y.)* 181(4096) (1973), pp. 223–230.

[2] Simon Batzner et al. "E(3)-equivariant graph neural networks for data-efficient and accurate interatomic potentials". In: *Nature Communications* 13.1 (May 2022), p. 2453.

[3] Johannes Brandstetter et al. *Geometric and Physical Quantities Improve E(3) Equivariant Message Passing*. 2021.

[4] John D. Chodera et al. "LongTime Protein Folding Dynamics from ShortTime Molecular Dynamics Simulations". In: *Multiscale Modeling & Simulation* 5.4 (2006), pp. 1214–1226. eprint: `https://doi.org/10.1137/06065146X`.

[5] Taco S. Cohen and Max Welling. "Steerable CNNs". In: *CoRR* abs/1612.08498 (2016). arXiv: `1612.08498`.

[6] G. Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of Control, Signals, and Systems* 2.4 (Dec. 1989), pp. 303–314.

[7] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. "Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering". In: *CoRR* abs/1606.09375 (2016). arXiv: `1606.09375`.

[8] Didier Devaurs et al. "A multi-tree approach to compute transition paths on energy landscapes". In: July 2013.

[9] Ken A. Dill et al. "The Protein Folding Problem". In: *Annual Review of Biophysics* 37.1 (June 2008), pp. 289–316.

[10] Robert P Dobrow. *Introduction to stochastic processes with R*. en. Nashville, TN: John Wiley & Sons, 2016.

[11] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. *Sigmoid-Weighted Linear Units for Neural Network Function Approximation in Reinforcement Learning*. 2017. arXiv: `1702.03118 [cs.LG]`.

[12] Fabian B. Fuchs et al. "SE(3)-Transformers: 3D Roto-Translation Equivariant Attention Networks". In: *CoRR* abs/2006.10503 (2020). arXiv: `2006.10503`.

[13] Hongyang Gao and Shuiwang Ji. "Graph U-Nets". In: *CoRR* abs/1905.05178 (2019). arXiv: `1905.05178`.

[14] Mario Geiger and Tess Smidt. *e3nn: Euclidean Neural Networks*. 2022.

[15] Mario Geiger et al. *github.com/e3nn/e3nn*. Version v0.3-alpha. Mar. 2020.

[16] Mahdi Ghorbani et al. *GraphVAMPNet, using graph neural networks and variational approach to markov processes for dynamical modeling of biomolecules*. 2022.

[17] Debasish Kumar Ghosh and Akash Ranjan. “The metastable states of proteins”. In: *Protein science : a publication of the Protein Society* 29(7) (2020), pp. 1559–1568.

[18] William L. Hamilton, Rex Ying, and Jure Leskovec. “Inductive Representation Learning on Large Graphs”. In: *CoRR* abs/1706.02216 (2017). arXiv: 1706.02216.

[19] M. J. Harvey, G. Giupponi, and G. De Fabritiis. “ACEMD: Accelerating Biomolecular Dynamics in the Microsecond Time Scale”. In: *Journal of Chemical Theory and Computation* 5.6 (May 2009), pp. 1632–1639.

[20] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385.

[21] Moritz Hoffmann et al. “Deeptime: a Python library for machine learning dynamical models from time series data”. In: *Machine Learning: Science and Technology* (2021).

[22] Kurt Hornik. “Approximation capabilities of multilayer feedforward networks”. In: *Neural Networks* 4.2 (1991), pp. 251–257.

[23] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. “Multilayer feedforward networks are universal approximators”. In: *Neural Networks* 2.5 (1989), pp. 359–366.

[24] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015.

[25] Hangjin Jiang and Xiaodan Fan. “The Two-Step Clustering Approach for Metastable States Learning”. In: *International Journal of Molecular Sciences* 22.12 (June 2021), p. 6576.

[26] John Jumper et al. “Highly accurate protein structure prediction with AlphaFold”. In: *Nature* 596.7873 (July 2021), pp. 583–589.

[27] Saugat Kandel et al. “Overcoming potential energy distortions in constrained internal coordinate molecular dynamics simulations”. In: *The Journal of Chemical Physics* 144 (Jan. 2016), p. 044112.

[28] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014.

[29] Bernard Osgood Koopman. “Hamiltonian Systems and Transformation in Hilbert Space”. In: vol. 17(5). National Academy of Sciences of the United States of America, 1931, pp. 315–318.

[30] S. Kullback and R. A. Leibler. “On Information and Sufficiency”. In: *The Annals of Mathematical Statistics* 22.1 (1951), pp. 79–86.

[31] Juho Lee et al. *Set Transformer: A Framework for Attention-based Permutation-Invariant Neural Networks*. 2019. arXiv: 1810.00825 [cs.LG].

[32] Yi-Lun Liao and Tess Smidt. *Equiformer: Equivariant Graph Attention Transformer for 3D Atomistic Graphs*. 2022.

[33] Kresten Lindorff-Larsen et al. “How Fast-Folding Proteins Fold”. In: *Science* 334.6055 (2011), pp. 517–520. eprint: https://www.science.org/doi/pdf/10.1126/science.1208351.

[34] Kresten Lindorff-Larsen et al. “Improved side-chain torsion potentials for the Amber ff99SB protein force field”. In: *Proteins: Structure, Function, and Bioinformatics* 78.8 (Mar. 2010), pp. 1950–1958.

[35] A. Mardt et al. “VAMPnets for deep learning of molecular kinetics”. In: *Nat Commun 9* 5 (2018).

[36] Andreas Mardt et al. “Deep learning Markov and Koopman models with physical constraints”. In: (Dec. 2019).

[37] Robert T. McGibbon and Vijay S. Pande. “Variational cross-validation of slow dynamical modes in molecular kinetics”. In: *The Journal of Chemical Physics* 142.12 (Mar. 2015). 124105. eprint: `https://pubs.aip.org/aip/jcp/article-pdf/doi/10.1063/1.4916292/14045907/124105\_1\_online.pdf`.

[38] Robert T. McGibbon et al. “MDTraj: A Modern Open Library for the Analysis of Molecular Dynamics Trajectories”. In: *Biophysical Journal* 109.8 (2015), pp. 1528–1532.

[39] Igor Mezi. “Spectral Properties of Dynamical Systems, Model Reduction and Decompositions”. In: *Nonlinear Dynamics* 41.1-3 (Aug. 2005), pp. 309–325.

[40] Benjamin Kurt Miller et al. *Relevance of Rotationally Equivariant Convolutions for Predicting Molecular Properties*. 2020.

[41] Frank Noé and Cecilia Clementi. “Kinetic Distance and Kinetic Maps from Molecular Dynamics Simulation”. In: *Journal of Chemical Theory and Computation* 11.10 (2015). PMID: 26574285, pp. 5002–5011. eprint: `https://doi.org/10.1021/acs.jctc.5b00553`.

[42] Frank Noé and Feliks Nüske. *A variational approach to modeling slow processes in stochastic dynamical systems*. 2012. arXiv: `1211.7103 [math-ph]`.

[43] Feliks Nüske et al. “Markov state models from short non-equilibrium simulationsAnalysis and correction of estimation bias”. In: *The Journal of Chemical Physics* 146.9 (Mar. 2017), p. 094104.

[44] Feliks Nüske et al. “Variational Approach to Molecular Kinetics”. In: *Journal of Chemical Theory and Computation* 10.4 (Mar. 2014), pp. 1739–1752.

[45] Feliks Nüske et al. “Variational tensor approach for approximating the rare-event kinetics of macromolecular systems”. In: *The Journal of Chemical Physics* 144.5 (Feb. 2016), p. 054105.

[46] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *CoRR* abs/1912.01703 (2019). arXiv: `1912.01703`.

[47] Alberto Perez et al. “Advances in free-energy-based simulations of protein folding and ligand binding”. In: *Current Opinion in Structural Biology* 36 (Feb. 2016), pp. 25–31.

[48] Guillermo Pérez-Hernández and Frank Noé. “Hierarchical Time-Lagged Independent Component Analysis: Computing Slow Modes and Reaction Coordinates for Large Molecular Systems”. In: *Journal of Chemical Theory and Computation* 12.12 (Nov. 2016), pp. 6118–6129.

[49] G.N. Ramachandran, C. Ramakrishnan, and V. Sasisekharan. “Stereochemistry of polypeptide chain configurations”. In: *Journal of Molecular Biology* 7.1 (July 1963), pp. 95–99.

[50] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. *Searching for Activation Functions*. 2017. arXiv: `1710.05941 [cs.NE]`.

[51] Prajit Ramachandran et al. *Stand-Alone Self-Attention in Vision Models*. 2019. arXiv: `1906.05909 [cs.CV]`.

[52] Anton Robert et al. “Resource-efficient quantum algorithm for protein folding”. In: *npj Quantum Information* 7.1 (Feb. 2021).

[53] Alexander S Rose et al. “NGL viewer: web-based molecular graphics for large complexes”. In: *Bioinformatics* 34.21 (May 2018), pp. 3755–3758. eprint: https://academic.oup.com/bioinformatics/article-pdf/34/21/3755/48921270/bioinformatics_34_21_3755.pdf.

[54] Olga Russakovsky et al. “ImageNet Large Scale Visual Recognition Challenge”. In: *CoRR* abs/1409.0575 (2014). arXiv: 1409.0575.

[55] Svitlana Ruzhytska et al. “Identification of metastable states in peptide’s dynamics”. In: *The Journal of Chemical Physics* 133.16 (Oct. 2010), p. 164102.

[56] Martin K. Scherer et al. “Variational selection of features for molecular kinetics”. In: *The Journal of Chemical Physics* 150.19 (May 2019). 194108. eprint: https://pubs.aip.org/aip/jcp/article-pdf/doi/10.1063/1.5083040/15559031/194108_1_online.pdf.

[57] Kristof T. Schütt et al. *SchNet: A continuous-filter convolutional neural network for modeling quantum interactions*. 2017. arXiv: 1706.08566 [stat.ML].

[58] C. E. Shannon. “A mathematical theory of communication”. In: *The Bell System Technical Journal* 27.3 (1948), pp. 379–423.

[59] Philipp Thölke and Gianni De Fabritiis. *TorchMD-NET: Equivariant Transformers for Neural Network based Molecular Potentials*. 2022. arXiv: 2202.02541 [cs.LG].

[60] Nathaniel Thomas et al. “Tensor Field Networks: Rotation- and Translation-Equivariant Neural Networks for 3D Point Clouds”. In: *CoRR* abs/1802.08219 (2018). arXiv: 1802.08219.

[61] Ashish Vaswani et al. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL].

[62] Petar Velikovi et al. *Graph Attention Networks*. 2018. arXiv: 1710.10903 [stat.ML].

[63] Benjamin D. Madej Ross Walker. “An Introduction to Molecular Dynamics Simulations using AMBER”. In: (20126).

[64] Yue Wang et al. *Dynamic Graph CNN for Learning on Point Clouds*. 2019. arXiv: 1801.07829 [cs.CV].

[65] Maurice Weiler et al. “3D Steerable CNNs: Learning Rotationally Equivariant Features in Volumetric Data”. In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc., 2018.

[66] Matthew O. Williams, Ioannis G. Kevrekidis, and Clarence W. Rowley. “A DataDriven Approximation of the Koopman Operator: Extending Dynamic Mode Decomposition”. In: *Journal of Nonlinear Science* 25.6 (June 2015), pp. 1307–1346.

[67] Daniel E. Worrall et al. *Harmonic Networks: Deep Translation and Rotation Equivariance*. 2016.

[68] Hao Wu and Frank Noé. “Variational Approach for Learning Markov Processes from Time Series Data”. In: *Journal of Nonlinear Science* 30.1 (Aug. 2019), pp. 23–66.

[69] Hao Wu et al. “Deep Generative Markov State Models”. In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. NIPS’18. Montréal, Canada: Curran Associates Inc., 2018, pp. 3979–3988.

[70] Saining Xie et al. “Attentional ShapeContextNet for Point Cloud Recognition”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, pp. 4606–4615.

[71] Jiancheng Yang et al. “Modeling Point Clouds With Self-Attention and Gumbel Subset Sampling”. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 3318–3327.

[72] A. Zee. *Group Theory in a Nutshell for Physicists*. USA: Princeton University Press, 2016.

[73] R Zwanzig, A Szabo, and B Bagchi. “Levinthal’s paradox.” In: *Proceedings of the National Academy of Sciences* 89.1 (Jan. 1992), pp. 20–22.

A

Results Omitted

A.1 Initialization Study for `e3nn` Network

As we began using the SimpleNetwork implemented in the `e3nn` software library, we began noticing that the network was unstable. It would sporadically converge to different scores. To investigate this matter we ran c. 100 different initializations trained on the same data partitioning and saved the initializations as good if they exceeded a threshold validation score of 4.50 on alanine dipeptide and bad otherwise.

Next, we performed PCA on the c. 15,000 trainable parameters and projected it into 2 dimensions, hoping that we would notice a clear separation between the two classes (good and bad). The results are displayed below.

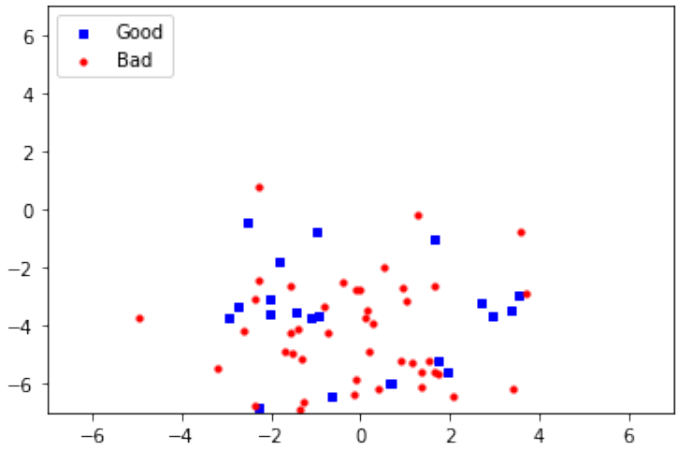
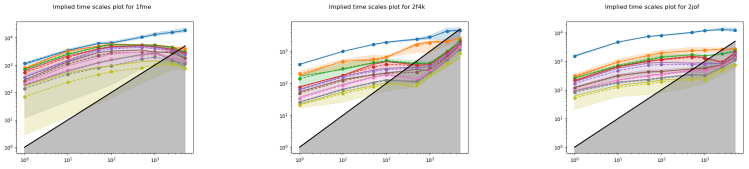


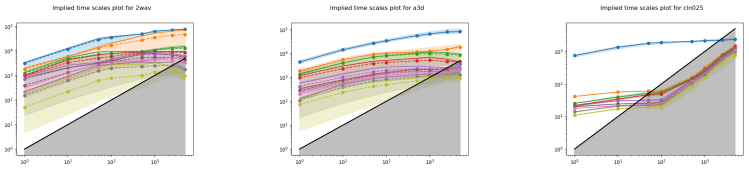
Figure A.1: PCA on the initialization of the 14,900 trainable parameters of the SimpleNetwork, labeled depending on the resulting VAMP-2 validation score.

However, no clear pattern emerged. Thus, we conclude that the instability of SimpleNetwork is likely not due to systematic differences in initialization.

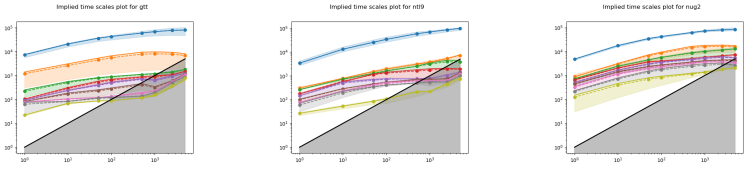
A.2 Implied Timescale Plots



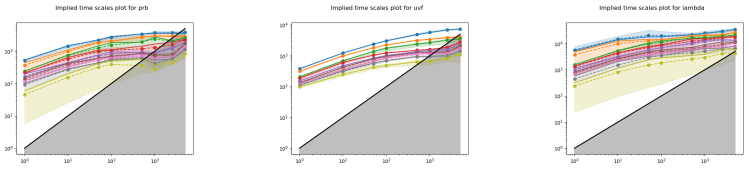
(a) 1FME (b) 2F4K (c) 2JOF



(a) 2WAV (b) A3D (c) cln025



(a) GTT (b) 2HBA (c) NUG2



(a) 1PRB (b) UVF (c) 1LMB

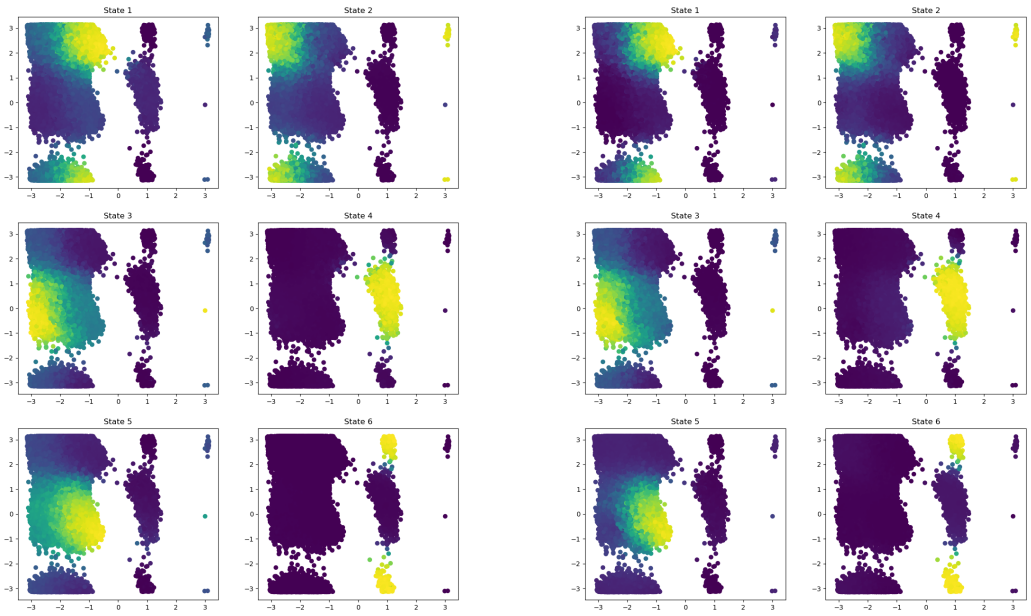
Figure A.2: Implied timescale plots for all protein folding datasets for a large range of timelags. Note: Each time-step is 200 ps between subsequent frames. I.e. $\tau = 100$ is equivalent to 20 ns.

A.3 Verifying the Alignment of Indices

As part of the systematic downsampling procedure we need to align the indices of different models. As the enumeration of the metastable states is arbitrary, each model initialization will have a different indices representing different metastable states. It is not a easy task to find a way of aligning the indices, in this thesis we use a linear sum assignment with the cost matrix being the squared distances between the matrices.

To check if this alignment procedure works, we initialize two different networks and train them on the alanine dipeptide molecule. We then use our alignment procedure

and display a dihedral plot colored by the probabilities assigned to each state. If the two plots show similarity in their ordering of the state assignments, it is an indication that the alignment procedure was successful.



(a) Network 1

(b) Network 2

Figure A.3: Side-by-side dihedral plots for the two networks displaying the assignment probabilities to each state after alignment, great similarity in the plots indicate a successful alignment procedure.

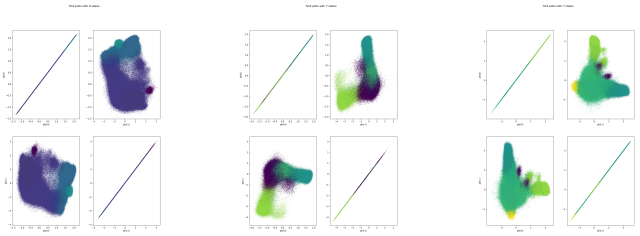
A.4 Detailed Convergence Rate

The study of the convergence rate examines the number of backward passes required to achieve a VAMP-2 training score of above 4.50. In contrast to what is displayed in the main report, here the error bars are included for the convergence speed.

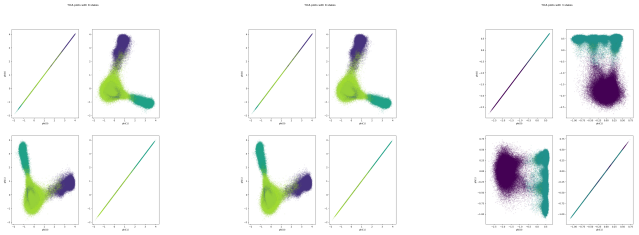
Model	Steps until convergence	VAMP-2 Score
MLP, Distance	226.6 ± 131.5	4.50 ± 0.03
DP Transformer ($l_{max} = 2$)	99.4 ± 25.8	4.49 ± 0.03

Table A.1: Results of benchmarking training rates

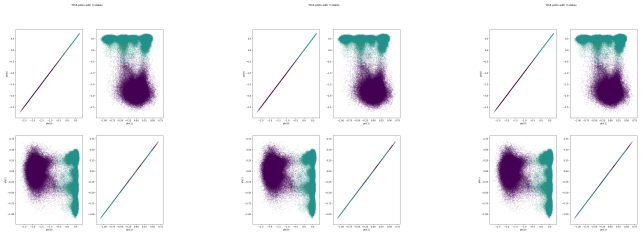
A.5 Visualizing Detected States



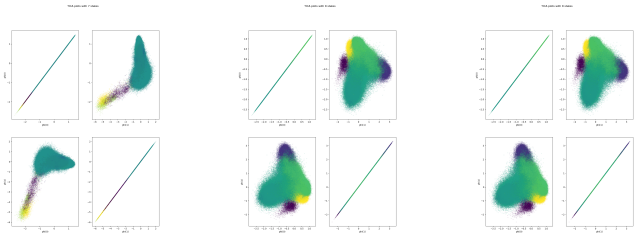
(a) 1FME (b) 2F4K (c) 2JOF



(a) 2WAV (b) A3D (c) cln025



(a) GTT (b) 2HBA (c) NUG2



(a) 1PRB (b) UVF (c) 1LMB

Figure A.4: TICA plots for all protein folding datasets, colored by the state assignments.

B

Software Libraries

B.1 PyTorch

PyTorch is a popular open-source machine learning library used for building and training deep neural networks. It was developed by Facebook's AI research team and is built on top of the Torch library. It was first introduced by [46].

PyTorch offers a range of tools for developing deep learning models, including a flexible and dynamic computational graph system, automatic differentiation, and a Pythonic interface that makes it easy to prototype and test models. This library also provides support for a variety of hardware platforms, including CPUs, GPUs, and specialized accelerators such as TPUs.

PyTorch has gained popularity among researchers and developers due to its ease of use and flexibility, as well as its strong community support and active development. It allows for fast and efficient prototyping of models, making it well-suited for exploratory research and experimentation. Additionally, PyTorch's dynamic computational graph system allows for easy debugging and the creation of complex models that may be challenging to implement in other libraries. The library's ability to seamlessly integrate with other Python libraries, such as NumPy, also makes it a popular choice among developers. Overall, PyTorch is a powerful and versatile library for deep learning that offers a range of tools and features that make it a top choice for machine learning practitioners, including much of the work on which this paper draws its foundations, e.g. [**3D_Steerable_CNNs**, 2, 3, 12, 14, 32, 36].

Implemented within the PyTorch library is the Adam optimizer which is based on the following algorithm:

Algorithm 2 Stochastic optimization using Adam. g_t^2 indicates the element-wise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$. $\beta_1 = 0.9, \beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise, where β_1^t and β_2^t are β_1 and β_2 to the power of t

Require: α : step size
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $J(\theta)$: Stochastic objective function with parameters θ
Require: θ_0 : Initial parameter vector
  $m_0 \leftarrow 0$ (Initialise 1st moment vector)
  $v_0 \leftarrow 0$ (Initialise 2nd moment vector)
  $t \leftarrow 0$ (Initialise time step)
  while θ_t not converged **do** $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta J_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at time step t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / \left(\sqrt{\hat{v}_t} + \epsilon\right)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

Source: [28]

B.2 PyTorch Geometric (PyG)

PyTorch Geometric (PyG) is an open-source Python library that provides support for developing deep learning models on graph-structured data, such as social networks, citation networks, biological networks, and more. PyG is built on top of the PyTorch deep learning framework and offers a comprehensive set of tools for creating, manipulating, and training graph neural networks (GNNs).

The main features of PyG include:

1. High-level abstractions for graph data: PyG provides a set of easy-to-use data structures that can represent graph-structured data in PyTorch. These structures include the Graph Data Object (GDO), which provides a unified interface for working with both node and edge features. This is the main use for this thesis.

2. Large-scale data handling: PyG supports the loading and processing of large-scale graph datasets, including datasets from popular repositories such as Open Graph Benchmark (OGB) and the Stanford Network Analysis Project (SNAP).

3. GNN model development: PyG offers a wide range of GNN models, including the Graph Convolutional Networks (GCN), Graph Attention Networks (GAT), and Message Passing Neural Networks (MPNNs), as well as tools for

developing custom GNN models.

4. GPU acceleration: PyG provides support for running computations on GPUs, which can significantly accelerate GNN training and inference.
5. Visualization: PyG includes tools for visualizing graphs, embeddings, and other features of GNN models.

The framework is powerful is actively maintained by a large community of developers and researchers. Furthermore, it has proven a powerful and flexible framework for developing deep learning models on graph-structured data. The above reasons has made it a popular choice for researchers, including works that are foundational for our paper [2, 3, 5, 12, 14, 32].

B.3 Deeptime

Deeptime is a Python library designed for estimating dynamical models based on time-series data presented by [21]. It offers a wide range of learning methods, including linear models like Markov State Models (MSMs) and Hidden Markov Models (HMMs), as well as kernel and deep learning approaches such as VAMPnets and deep MSMs. The library is largely compatible with scikit-learn and provides Estimator classes for various models, along with Model classes that offer a variety of analysis methods for computing thermodynamic, kinetic, and dynamical quantities.

Deeptime's Model classes allow users to compute interesting quantities such as free energies, relaxation times, and transition paths. This makes Deeptime a useful tool for researchers and practitioners working in fields that involve dynamical modeling and time-series analysis. In summary, Deeptime provides a comprehensive and flexible framework for estimating and analyzing dynamical models based on time-series data.

For further information on the library we refer to [21] as well as the Deeptime repository.

B.4 `e3nn`

Below is an outline of the syntax of `e3nn` in accordance with [14, 15]. This is done as the framework is a central part of the thesis and in order to understand our equivariant neural network, one has to be familiar with the structure that underpins it. It is for the curious reader to dive into, but not necessary to understand the core of our thesis.

The syntax of `e3nn` revolves around irreducible representations and spherical harmonics. In order for the network to understand how our data transforms under training we must provide details on how the input feature/output transforms under symmetry, which is encoded in the irreducible representations. The irreducible representations of $O(3)$ are represented by the class `e3nn.o3.Irrep` where the direct sum of irreducible representations are represented by the class `e3nn.o3.Irreps` [15].

Note that the output of an `e3nn` model must always be of equal or higher symmetry than the input.

In general, irreducible representations in `e3nn` are characterized by two properties, their dimension and how they are transformed under parity. As previously discussed, spherical harmonics have odd dimensions and the convention is thus to denote the dimension (when talking about irreducible representations) by an integer l for which $d = 2l + 1$. The other property, transformation under parity, refers to whether the data changes sign under the transformation of parity. For example, a scalar does not change sign under parity and is thus denoted even, while a vector does and is thus denoted odd. In the table below a summary of this concept is provided [15].

The irreducible representations			
l	Parity	d	Name
0	even	1	scalar
0	odd	1	pseudo-scalar
1	even	3	pseudo-vector
1	odd	3	vector
2	even	5	without name
2	odd	5	without name

The general notation is `Mxlp` where M represents the number of 3D coordinates per input, l is the spherical harmonic from the table above and p is the parity (even (e) or odd (o)). For example, if you want to represent 5 scalars and 3 vectors the syntax is `5x0e+3x1o` [15].

B.4.1 Spherical Harmonics in `e3nn`

The sign of the spherical harmonics in `e3nn` is chosen by the Clebsch-Gordan coefficients, through the following expression.

$$Y^{l+1}(\vec{x}) \propto C^{l+1,l,1} Y^{l}(\vec{x}) Y^{1}(\vec{x}) \tag{B.1}$$

Where $C^{l+1,l,1}$ represents the Clebsch-Gordan coefficients.

B.4.2 Testing Equivariance

In order to test for equivariance to the actions of translation and rotation, the `e3nn` library provides the function `e3nn.util.test.equivariance_error` which provides the network's deviance from equivariance when subject to the actions of SO(3).

For the convenience of testing for equivariance on a custom network (which can be regarded as a function) the wrapper function `assert_equivariance` is provided, which checks whether the assertion of equivariance is satisfied through the equivariance

error. This function will be used in order to ensure that our proposed network is equivariant.

B.4.3 Initialization

The initialization scheme of `e3nn` satisfies the following at initialization:

- The preactivation have $\mu = 0$ and $\sigma^2 = 1$.
- The post activations satisfy $E[\cdot^2] = 1$.
- The layers learn when the width (i.e. the multiplicities) is sent to ∞.

A more detailed discussion of how these properties are satisfied technically is provided by [14]. To exemplify these properties, a fully connected layer given by:

$$z = \phi(\frac{1}{\sqrt{d}}Wx) \tag{B.2}$$

where $z \in \mathcal{R}$ is a post activation, $W \in \mathcal{R}^{n \times d}$ is the parameter matrix and $x \in \mathcal{R}^d$ are the inputs or outputs of the previous layer. By choosing a random standard normal initialization of W and rescaling by the multiplicative constant $\frac{1}{\sqrt{d}}$ bring the variance to 1. If necessary, ϕ, the activation, is rescaled to satisfy a unit second moment.

C

Details of Experiments

C.1 Training Details

The data partition is randomly initialized, however, all models are ensured to be evaluated on the same data partitioning. For reproducibility, a manual seed was used allowing the same data partitioning to be created by following the directions on our Github repository.

We train the dot product and graph attention transformers with the same hyperparameters. After a detailed hyperparameter sweep we landed on using 4 transformer blocks with $l_{max} = 2$. We choose Gaussian radial basis [32], a dropout rate of 0.2 to attention weights a_{ij}. The atom one-hot encoding is indistinguishable, meaning that each atom is given a unique value. The number of epochs is 5 on alanine dipeptide but reduced to 1 for the benchmarking on the protein folding datasets due to computational constraints. The learning rate is 1×10^{-3}. The batch size is 5000 for experiments on alanine dipeptide and the same on the protein folding datasets, however, for these we adjusted the number of gradient accumulation steps to equate to this batch size. The weight decay is 0. Table C.1 summarizes the hyperparameters used for the equivariant transformer models.

Hyperparameters	Value or description
Optimizer	Adam
Learning rate	1×10^{-4}
Batch size	5000
Number of epochs	20
Weight decay	0
Dropout rate	0.2
Cutoff radius (nm)	0.5
Number of radial basis	128
Hidden size of radial function	64
Number of hidden layers in radial function	2
Atom feature encoding	Indistinguishable
Number of transformer blocks	4
Embedding dimension d_{embed}	$16 \times 0e + 8 \times 1e + 4 \times 2e$
Spherical harmonics embedding dimension d_{sh}	$1 \times 0e + 1 \times 1e + 1 \times 2e$
Number of attention heads	4
Attention head dimension d_{head}	$8 \times 0e + 4 \times 1o + 2 \times 2e$
Hidden dimension in feed forward network d_{ffn}	$128 \times 0e + 64 \times 1e + 32 \times 2e$
Output feature dimension $d_{feature}$	$8 \times 0e$

Table C.1: Hyperparameters for transformer networks, we use `e3nn`'s notation for irreps.

As for the graph convolutional neural network, the same training parameters (number of epochs, learning rate and batch size) are used for clear comparability. Cosine is chosen as the radial basis and no dropout is included. The atom one-hot encoding is distinguishable, meaning that each atom of the same type is given the same value. For computational efficiency, the number of nodes to typically convolve over is chosen as the average number of neighbors in the training set. As for activation functions, the SILU function is used for even parity and the hyperbolic tangent is used for odd parity. Gates are used to separate scalars from higher order vectors during activations. Table C.2 summarizes the hyperparameters used for the equivariant graph convolutional model.

Hyperparameters	Value or description
Optimizer	Adam
Learning rate	1×10^{-4}
Batch size	5000
Number of epochs	20
Cutoff radius (nm)	0.5
Number of radial basis	10
Number of hidden layers in radial function	2
Atom feature encoding	Distinguishable
l_{max}	2
Multiplicity	8
Irreps in	$3 \times 0e$
Irreps out	$2 \times 0e$

Table C.2: Hyperparameters for Graph Convolutional Network, we use `e3nn`'s notation for irreps.

We use one A40 GPU with 48GB to train each model and summarize the computa-

tional cost of training for one epoch on alanine dipeptide in Table C.3 and on the different protein folding datasets in Table.

Model	Training time (s/epoch)
Multilayered Perceptron, Distance	7.4
Multilayered Perceptron, XYZ	8.0
Graph Attention Transformer ($l_{max} = 0$)	64.4
Graph Attention Transformer ($l_{max} = 1$)	107.6
Graph Attention Transformer ($l_{max} = 2$)	303.6
Dot Product Transformer ($l_{max} = 0$)	68.3
Dot Product Transformer ($l_{max} = 1$)	115.1
Dot Product Transformer ($l_{max} = 2$)	318.4
Graph CNN ($l_{max} = 0$)	52.7
Graph CNN ($l_{max} = 1$)	62.2
Graph CNN ($l_{max} = 2$)	79.3

Table C.3: Training time using an A40 48GB GPU on alanine dipeptide.

Protein	MLP xyz	DP transformer	GA transformer
2F4K	14.4	16,450.3	15,615.8
2JOF	17.7	13,592.2	12,809.4
cln025	11.7	3,886.1	3,671.3
PRB	14.1	16,997.0	16,039.3
Average:	**14.5**	**12,731.4**	**12,034.0**

Table C.4: Training time using an A40 48GB GPU on various protein folding datasets, results in second per epoch.

C.2 Finding Suitable Hyperparameters

We decided on a learning rate and number of epochs that would provide all models with convergent results. However, we did not manage to get the convolutional network to always converge, regardless of number of epochs or learning rate (constrained to that we could not simply append a complex MLP at the end to achieve convergence).

C.3 Benchmark Models

The simple benchmark employed is a multi-layered perceptron (MLP) with 4 hidden layers. The benchmark model originates from the original VAMPnet paper [35] which employs this model. To improve this benchmark we also include a invariant version of this MLP which computes the pairwise distances between atoms before feeding it into the MLP.

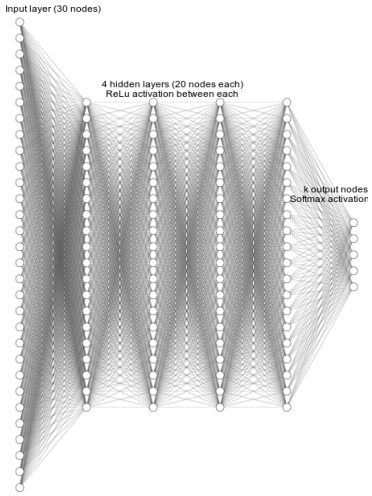


Figure C.1: Benchmark MLP structure

Furthermore, in order to understand the impact of including higher order symmetries we benchmark our model against the same architecture, but limiting the intermediary feature representation to only include scalars, thus, limiting the expressability of higher order symmetries.