HYPERPARAMETER OPTIMIZATION OF AUTOENCODERS FOR DEEP LEARNING OF COLLECTIVE VARIABLES

by

Nurdan Özkaraaslan

B.S., Molecular Biology, Genetics and Bioengineering, Sabanci University, 2017

B.S., Computer Science and Engineering, Sabanci University, 2018

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computational Science and Engineering

Boğaziçi University

2024

ACKNOWLEDGEMENTS

This thesis represents not just an academic achievement, but a voyage of intellectual discovery and personal growth, made richer and more meaningful by those who have journeyed alongside me. At the forefront of this supportive network has been my supervisor, Assist. Prof. Betül Uralcan, whose intellectual rigor and meticulous guidance have been the cornerstone of my research. Her commitment to scholarly excellence and critical inquiry has profoundly influenced my approach to academic work. I am profoundly grateful to her for granting me the invaluable opportunity to delve into a topic that I am deeply passionate about. Her support not only fueled my academic pursuits but also allowed me to explore a field that truly captivates and inspires me.

I am deeply grateful to my committee members, Assoc. Prof. Burak Alakent and Assoc. Prof. Nazar Ileri Ercan, whose insightful feedback and broad knowledge base have significantly enriched my work. Their discerning advice has been pivotal in sharpening my arguments and deepening the analytical depth of my study.

The collaborative spirit of my peers in the Soft Matter Laboratory, especially MSc. Duygu Kaya and Özge Öztürk and MSc. Ayşe Saliha Korkut, has been a source of strength and inspiration. Our thought-provoking discussions and mutual support have been indispensable in this intellectual voyage.

On a deeply personal note, I am immensely grateful to the Erdem and Özkaraaslan families for their unwavering support and encouragement. Your collective belief in my abilities has been a source of constant strength throughout my academic journey. I owe an immeasurable debt of gratitude to my mother, whose infinite love and unwavering support have been the bedrock of my success. Her wisdom and sacrifices have not only shaped who I am but have also instilled in me the resilience and perseverance necessary to pursue my academic goals. Her strength, especially in the absence of my late father, whose memory and teachings continue to guide and inspire me, has been a constant

source of inspiration. My heartfelt thanks extend to my mother-in-law and father-in-law, who have embraced me with open arms and constant encouragement, enriching my life with their kindness and wisdom. My sisters and their children have added joy and laughter to my life, reminding me of the importance of balance and happiness outside the academic world. My nephews and niece, whom I hope to inspire to pursue their own paths of inquiry and learning: may you always be curious, fearless, and driven by a love of knowledge. Additionally, I am indebted to my childhood friends and BFFs, whose moral support and timely distractions have been essential to maintaining my equilibrium through the highs and lows of this academic endeavor.

Above all, I extend my deepest gratitude to my husband, my life companion, Mehmet Akif. His presence has been a source of boundless love. He has been not just a partner but a pillar of calm in this agonizing period, providing both intellectual companionship and emotional resilience. As my constant supporter, he encouraged me to chase my dreams with a smile. His empathy and understanding during the most demanding phases of this work have not only reinforced my determination but have also profoundly enriched our shared life. His involvement has been transformative, touching every part of this thesis and every aspect of my life.

This thesis is not only a reflection of my hard work but also a celebration of all the remarkable people who have made this journey possible. Thank you for walking this path with me, for every word of encouragement, every moment of doubt dispelled, and every burst of laughter shared. Your influence is indelibly etched into the pages of this work, and for that, I am forever grateful.

ABSTRACT

HYPERPARAMETER OPTIMIZATION OF AUTOENCODERS FOR DEEP LEARNING OF COLLECTIVE VARIABLES

Understanding the conformational space of proteins is integral to deciphering the mechanism of action of the proteins in drug-discovery studies. Molecular dynamics (MD) simulations have enabled the realistic simulation of protein dynamics. However, the computational demands of processing extensive MD data necessitate efficient dimensionality reduction techniques. An automated method for extraction of collective variables (reduced representations of protein movements) for the accurate representation of inherent dynamics of proteins is important. For this purpose, this thesis involves implementation of a semi-supervised autoencoder (SAE) for the extraction of collective variables from MD simulations of MurD and adenosine kinase (AdK) proteins, selected for their distinct dynamic states. This model, which incorporates extra output nodes coming from the latent layer and encoder hidden layers carrying the extra information such as opening angle, significantly outperforms the traditional vanilla autoencoder by achieving better results in reconstructing protein dynamics. The research also explores the optimization of the semi-supervised autoencoder architecture using genetic algorithm and enhances the quality of the extracted collective variables. By focusing on features such as Cα coordinates and dihedral angles, and adjusting the model's hyperparameters, the study advances our ability to capture the essential movements within proteins. Moreover, the semi-supervised approach shows an important improvement in handling the highly flexible regions of proteins, particularly the C-terminal domain of MurD, which is crucial for its functional transitions. The findings also shows the potential of machine learning in bioinformatics to refine simulation data analysis.

ÖZET

KOLLEKTİF DEĞİŞKENLERİN DERİN ÖĞRENMESİ İÇİN OTOKODLAYICILARIN HİPERPARAMETRE OPTİMİZASYONU

Proteinlerin konformasyon şekillerini anlamak, ilaç keşif çalışmalarında proteinlerin etki mekanizmasını çözümlemek için gereklidir. Moleküler dinamik (MD) simülasyonları, protein dinamiği gerçekçi simülasyonunu mümkün kılmıştır. Ancak, geniş MD veri setlerini işlemenin yüksek hesaplama gereksinimleri, etkili boyut indirgeme tekniklerini zorunlu kılmaktadır. Bu amaçla ilgili tez çalışmasında, MurD ve adenozin kinaz (AdK) proteinlerinin MD simülasyonlarından kolektif değişkenlerin (protein hareketlerinin indirgenmiş temsilleri) çıkarılması için yarı denetimli bir otokodlayıcı (SAE) uygulanmıştır. Bu model, geleneksel otokodlayıcıya göre, protein dinamiklerini yeniden oluşturmada daha iyi sonuçlar elde ederek önemli ölçüde üstün performans göstermiştir. Araştırma, genetik algoritma kullanılarak yarı denetimli otokodlayıcı mimarisinin optimizasyonunu da keşfetmiş ve elde edilen kolektif değişkenlerin kalitesini artırmıştır. Cα atomu koordinatları ve dihedral açıları gibi özelliklere odaklanarak ve modelin hiperparametrelerini ayarlayarak, proteinler içindeki esas hareketleri yakalama kabiliyetimizi ilerletmiştir. Ayrıca, yarı denetimli yaklaşım, özellikle fonksiyonel geçişler için kritik olan MurD'nin C-terminal bölgesi gibi yüksek derecede hareketli protein bölgelerini ele almakta önemli bir iyileşme göstermiştir. Bu bulgular, biyoinformatikte makine öğreniminin simülasyon veri analizini geliştirme potansiyelini ortaya koymaktadır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

$a(t)$	Trajectory of simulation system
a_{ix}	x-coordinate value of ith atom
a_{iy}	y-coordinate value of ith atom
a_{iz}	z-coordinate value of ith atom
$\hat{a}(x)$	Reconstructed simulation trajectory
c	Small constant to prevent division by zero in Adam optimization
Cα	Carbon-alpha atom
d	Latent space dimension
D	Dataset
f	Unknown black-box function
g	Gradient
k	Penalty term
L	Loss function
L_{Total}	Total loss function
L_{coords}	Loss function from coordinates
L_{angles}	Loss function from angles
m_t	Exponentially decaying average of past gradients
$\hat{m}_t$	Bias corrected version of m_t
M	Number of frames in a trajectory
n	Population size in genetic algorithm
N	Number of atoms
$\mathcal{N}$	Normal distribution
R	Number of residues
v_t	Exponentially decaying average of past squared gradients
$\hat{v}_t$	Bias corrected version of v_t
x^*	Global optimum of an unknown function
x_i	Sample points of objective function

α	Constant for SELU function
β_1	Exponential decay rates for first moment estimates
β_2	Exponential decay rates for second moment estimates
γ	Scaled constant of SELU function
Δt	Time interval for storing trajectory
ϵ	Trade-off parameter of expected improvement
η	Learning rate
θ_{t+1}	Update rule of Adam optimization
κ	Non-negative parameter that controls the trade-off between exploration and exploitation
λ	Regularization parameter
$\mu(x)$	Mean function
ξ	Latent space vectors
$\sigma(x)$	Sigmoid function
$\sigma^2(x)$	Variance function
ϕ	Phi dihedral angle
ψ	Psi dihedral angle
ω	Weights
Å	Angstrom

LIST OF ACRONYMS/ABBREVIATIONS

AdK	Adenosine Kinase
AMP	Adenosine Monophosphate
ATP	Adenosine Triphospate
CV	Collective Variable
cRMSE	Combined Root Mean Squared Error
DHL	Decoder Hidden Layer
DIMS	Dynamic Importance Sampling
DSSP	Dictionary of Protein Secondary Structure
EHL	Encoder Hidden Layer
FEBILAE	Fee Energy Biasing and Iterative Learning with Autoencoders
FES	Free Energy Surface
fRMSE	Root Mean Square Error per Frame
GA	Genetic Algorithm
GMVAE	Gaussian Mixture Variational Autoencoder
GPU	Graphical Processing Unit
LASSO	Least Absolute Shrinkage and Selection Operator)
LDA	Linear Discriminant Analysis
L1	LASSO Regularization
L2	Ridge Regularization
MD	Molecular Dynamics
MESA	Molecular Enhanced Sampling with Autoencoders
ML	Machine Learning
MurD	UDP-N-acetylmuramoyl-L-alanine-D-glutamate ligase
PCA	Principle Component Analysis
PDB	Protein Data Bank
ReLU	Rectified Linear Unit
REMD	Replica Exchange Molecular Dynamics
RMSD	Root Mean Squared Distance

RMSE	Root Mean Squared Error
rRMSE	Root Mean Squared Error per Residue
RMSF	Root Mean Squared Fluctuation
SAE	Semi-supervised Autoencoder
SASA	Solvent Accessible Surface Area
SELU	Scaled Exponential Linear Unit
SSE	Secondary Structure Elements
VAE	Vanilla Autoencoder

1. INTRODUCTION

Atomic-level conformational changes are essential to many biological processes such as the folding of proteins and interactions with ligands. Understanding these changes is core to deciphering protein-ligand interactions that provide information about possible drug candidates for enzyme-related diseases [1, 2]. The study of these conformational transitions presents challenges for experimental approaches that lead to the development of theoretical methods. Molecular dynamics simulations have emerged as an effective computational technique for investigating the dynamic behaviours of biomolecules [3, 4]. The goal of MD simulations is to characterize conformational space of molecules. Conformational space refers to the possible spatial arrangements of atoms in a molecule. Through the use of all-atom classical molecular dynamics simulations, a vast range of biological systems, from small molecules to large protein complexes are investigated [5–7]. These studies provided more insights into the dynamic behaviour of proteins and paved the way for understanding their structural characteristics.

Atomic arrangement information about the interatomic interactions is needed as the starting structures for the MD simulations. For this purpose, X-ray crystallography [8] or nuclear magnetic resonance techniques [9] can be utilized for understanding the atomic arrangement information of proteins. The resulting structural information can be used as an initial structure to start the simulations and from there MD simulations generate the dynamical structure of the protein in an iterative manner with a time trajectory. The fundamental principle of the MD simulations is the numerical integration of Newton's equations of motion acting on the atoms of the proteins or any biological system. Newton's equations of motion use the initial configuration of atoms to compute the forces acting on each atom iteratively for discrete time steps using empirical force fields [10]. The new Cartesian coordinates and velocities generated by these forces for each atom are updated at each iteration. The resulting iterative process is a three-dimensional trajectory showing the dynamic behaviour of the atoms, hence the protein, throughout the simulated time interval [11]. For maintaining the numeri-

cal stability in the integral operation, the time steps taken should be shorter than the biochemically significant events. These events occur over nanoseconds, microseconds, or longer [12]. Therefore the time steps should be in the order of femtoseconds. Due to the length of the biochemical events and the short time steps, the resulting simulation involves millions or billions of time steps for atomic interactions and this leads to computationally demanding MD simulations. To overcome this issue, advanced algorithms with parallel computing and GPU acceleration [13] are developed to distribute the workload across multiple processors. In addition to these developments, approximation techniques such as coarse-grained models [14] can also be employed. These methods can solve the computational complexity problem without losing the essential information and accuracy of the simulations.

1.1. Collective Variables

The free-energy landscape of biological molecules reveals their dynamic behaviour. In the case of proteins, understanding the free-energy landscape is crucial for understanding their folding dynamics [15]. Biological molecules exhibit rugged free-energy landscapes containing several local minima along with high-energy barriers [16]. The complexity of the landscapes is a challenging task for standard MD algorithms that are designed for sampling the conformational space of proteins. The rugged-free energy landscapes can trap the simulations in local minima regions causing the simulation to fail ergodic sampling which is the sampling of full spectrum of molecular conformations. To address this issue and achieve ergodic sampling, enhanced sampling algorithms [17] have been developed such as metadynamics [18–22], umbrella sampling [23, 24], adaptive biasing force [25, 26] and replica-exchange molecular dynamics (REMD) [27, 28]. The former three of these are in general known as "biased sampling" methods where a bias potential is applied to overcome energy barriers and explore specific regions of conformational space. Metadynamics, for instance, overcomes the trapping in the local minima by introducing a history-dependent biased potential on selected collective variables (CVs) lowering energy barriers between minima. They are also known as reaction coordinates, order parameters, or slow degrees of freedom [29, 30]. However, achieving

a converged free-energy surface (FES) in metadynamics poses challenges. Due to this, careful selection of appropriate collective variables that capture the relevant features of the system's configuration is necessary. Commonly used CVs include what is called a "geometric CVs" which include dihedral angles, distances between molecular groups, radius of gyration or root-mean-square-deviation (RMSD) which is the deviation of atomic coordinates from their initial conformation [31–33]. There are software tools, such as Plumed [34], that contains built-in features for conducting MD trajectory analysis using geometric CVs. In simpler systems, the selection of these geometric collective variables can be guided by experience or intuition [35]. However, in more complex systems, the selection of collective variables becomes less straightforward. Therefore, more systematic methods are needed for their identification.

1.1.1. Collective Variable Discovery with Machine Learning Methods

Geometric collective variables are useful in many biased sampling algorithms. However, their identification is mostly intuition-driven or based on previous knowledge of the protein under consideration [36]. This hinders the enhanced sampling of proteins without prior knowledge about their dynamics. To address these issues, a paradigm shift has occurred from intuition-driven strategies to data-driven approaches in recent years. These data-driven approaches aim to automate the process of CV identification. Advanced machine learning (ML) methods such as deep neural networks can be integrated into CV discovery processes. These techniques allow systematic CV identification methods with accurate results. In a recent study, authors showed that CV discovery can be carried out retrospectively through a process known as MD/ML resampling [37]. In this approach, first, the relevant CVs are identified with the data analysis of exploratory simulations. Second, the biased sampling methods are conducted using the CVs from the previous step. This process is retrospective in nature. And this results in efficient biased sampling. For other techniques which are not retrospective, real-time methods exist. For instance, a real-time method called on-the-fly MD/ML was developed by Chen et al. allows simultaneous CV discovery and biased sampling simulations [38, 39]. With this method, the real-time identification of CVs

allows dynamic adjustments to the biased simulation. The integration of ML methods to the CV discovery enhances the efficiency of the biased simulations whether applied retrospectively or in real-time. In general, the synergy between the ML and MD allows researchers to optimize simulations and improve the sampling strategies. So that a deeper insight into the complex behaviour of protein dynamics is achieved.

Machine learning methods are categorized into two main types: supervised learning and unsupervised learning. Supervised learning involves dependent and independent variables. And the aim is to reveal the relationship between these variables. In supervised learning, the algorithm is trained on a labeled data set. These labels "supervise" the algorithm to differentiate between different categories of data. After learning these patterns from the training process, the algorithm is expected to make predictions on a new dataset called test data. In unsupervised learning, on the other hand, there is no labeled data provided to the algorithm. It is trained on an unlabeled data and expected to discover the hidden patterns within the data. Both of these methods have a wide range of applications for many different problems. The details and the applications of these within the realm of collective discovery will be discussed in the following section.

1.1.2. Supervised Learning

Supervised learning consists of two primary categories: regression [40] and classification [41]. In regression tasks, the objective is to develop predictive models that characterize the connection between continuous dependent variables and independent variables. From the labeled training data, the regression algorithm learns the relationship between the variables and plots a best-fit line between them. On the other hand, classification tasks aim to differentiate between the labeled categorical variables. And it is expected to predict the category of the new observations after learning the training data. Some of the classification methods are employed in collective variable discovery. For instance, Sultan et al. [42] demonstrated the use of classification algorithms, such as support vector machines and logistic regression as effective initial CVs

for accelerated sampling by using alanine dipeptide and Chignolin mini-protein as test cases. Another classification method known as linear discriminant analysis (LDA) is used in a different study to construct one-dimensional collective variables for studying rare transitions between two metastable states [43]. Although these methods are powerful in extracting the collective variables, they are limited in the sense that they are particularly applicable to only molecular systems with well-defined known end states.

1.1.3. Unsupervised Learning

Unsupervised learning algorithms constitute a diverse category in machine learning. One subset of unsupervised methods is clustering algorithms [44]. Clustering is a data-mining technique to identify groups of inputs that share inherent similarities. By analyzing patterns and relationships within the data, clustering methods aim to categorize inputs into distinct classes. This approach is particularly valuable in scenarios where the underlying structure of the data is not explicitly known. Clustering approaches can be used in collective variable discovery. Tribello et al. introduced the 'reconnaissance metadynamics' method that employs an advanced clustering procedure to determine collective variables [45]. This approach involves the identification of clusters within the trajectory at regular intervals. These identified clusters are then utilized to fine-tune a one-dimensional collective variable to be used in subsequent metadynamics.

Another aspect of unsupervised learning is the dimensionality reduction techniques [46]. These techniques are designed to address the challenge of high-dimensional data, also known as the curse of dimensionality [47]. The high-dimensional data is not straightforward to analyze and understand. Therefore, reducing the dimension of the data into more compact form without losing the essential information is necessary. For this purpose, the dimensionality reduction operates by reducing the number of features in the input vectors without losing the essential information. Methods such as principal component analysis (PCA) [48] fall under this category. PCA is a valuable tool for reducing the data into statistically significant variables while preserving the essential

information. Linear transformation of feature vectors is used in PCA with the aim of representing data variance. This process results in eigenvectors, commonly referred to as principal components, arranged in descending order based on the proportion of total variance they represent. When the PCA method is applied to the MD trajectory data, the resulting principal components can be used as CVs for the downstream biased sampling methods such as metadynamics [49]. This shows that the collective variables can be obtained in an automated manner from PCA to be used in downstream biased sampling methods. While principal components can be effective order parameters for distinguishing relevant states, they might not always be optimal CVs for biased sampling due to certain limitations. Since PCA is a linear transformation, it may not fully represent the intricate nonlinear interactions in molecular dynamics. Particularly processes like protein folding involve non-linear dynamics [50]. Due to the limitations of linear techniques, there is a growing need for non-linear approaches in CV discovery to better represent the complexity of molecular interactions. Nonlinear techniques, such as isomap [51, 52] and diffusion maps [53–55] offer more flexibility in discovering nonlinear CVs, but they lack explicit mappings from molecular coordinates to CVs, which are required in biased sampling methods to transmit biased forces from CVs to atomic forces. There are other non-linear dimensionality reduction techniques developed based on neural networks such as autoencoders. In the next section, autoencoders and their role in collective variable discovery will be discussed.

1.2. Autoencoders and Collective Variables

In recent times, the advent of deep learning methodologies centered around artificial neural networks have introduced a novel approach for uncovering nonlinear collective variables. These techniques excel in providing explicit mappings connecting the identified CVs to the atomic coordinates. Auto-associative neural networks or autoencoders, are a specific category of artificial neural networks explicitly crafted for the purpose of nonlinear dimensionality reduction [56]. Autoencoders are composed of two components: an encoder and a decoder (Figure 1.1). The encoder is responsible for mapping input data into a lower-dimensional subspace called the latent layer, while

the decoder tries to reconstruct the original input from the latent layer. The latent layer vectors serve as the nonlinear collective variables that represent the inherent variance in the protein dynamics. In essence, the utilization of autoencoders in molecular simulations offers a powerful approach for uncovering meaningful collective variables.

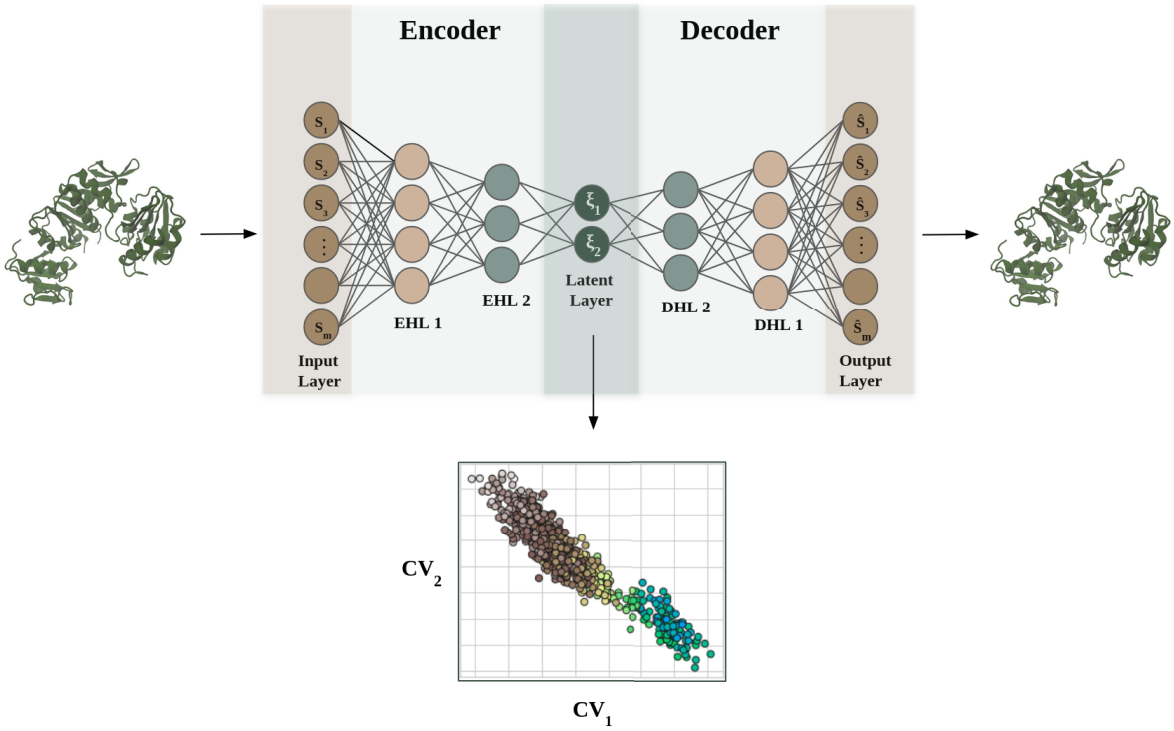

Figure 1.1. Illustration of autoencoder architecture for molecular dynamics simulation trajectory data reconstruction. The input from the protein simulation data is passed through the encoder hidden layers (EHLs) to the latent layer. Information from latent layer is passed to the output through decoder hidden layers (DHLs). The latent layer vectors can be utilized as the collective variables in the subsequent analysis.

The simplicity and efficacy of autoencoders have led to numerous studies for extracting collective variables from simulation data through autoencoders [57–59]. Belkacemi et al. introduced an iterative approach, FEBILAE (free energy biasing and iterative learning with autoencoders), using autoencoders to learn CVs iteratively [60]. This iterative framework involves a cyclical process wherein the initially learned CV is employed for free energy adaptive biasing, leading to the generation of new data. Subsequently, the autoencoder is tasked with learning a new set of CVs in each iteration.

The efficacy of this approach is demonstrated through molecular systems such as the alanine dipeptide and solvated Chignolin mini-protein. In a different article, Ferguson and his team introduced a novel methodology called MESA (molecular enhanced sampling with autoencoders) for on-the-fly discovery of low-dimensional collective variables [38, 39]. Similar to FEBILAE, it requires only a brief initial simulation and then it iteratively trains autoencoders to learn collective variables. Then the resulting CVs were used in the downstream umbrella sampling. They demonstrate this approach using molecular systems like alanine dipeptide and Trp-cage, utilizing an open-source plugin with the OpenMM [61] simulation package. In another study, Bandyopadhyay et al. propose using inter-residue Cα-distances as potential input CVs, deriving nonlinear combinations of latent space embedded CVs using autoencoders on molecular dynamics simulations [62]. The resulting latent space variables effectively characterize the conformational landscape in diverse macro molecular systems, including a bead-in-a-spring polymer, Trp-cage mini-protein, and cytochrome P450. Continuing with this line of research, Ghorbani et al. present GMVAE (Gaussian mixture variational autoencoder), that combines dimensionality reduction and clustering of biomolecular conformations from molecular dynamics simulations [63]. It incorporates a categorical variable to identify the mode of each data point. Using normalized distance maps between Cα atoms of proteins as features. The model is tested on long-timescale simulations of Trp-cage, BBA, and villin. They demonstrated that it accurately learns the funnel-shaped landscape of protein folding. Although these methods are powerful in extracting collective variables and reconstructing the input data, they suffer from limitations associated with the selection of optimal hyperparameters for the autoencoder architecture. Fine-tuning hyperparameters addresses challenges such as overfitting, underfitting, and sensitivity to variations in input data characteristics [64]. This is because the efficient performance of autoencoders in general relies heavily on appropriately tuned hyperparameters. When it comes to biomolecular data, researchers rely on insights into the specific characteristics of data and the intricacies of molecular dynamics simulations to make informed decisions about hyperparameter configurations. However, this intuitive approach is inherently limited by the subjective nature of human judgment and the potential challenges posed by the high-dimensional and complex nature of biomolecu-

lar datasets. Since biomolecular systems exhibit inherent complexity and variability, a systematic approach to hyperparameter optimization becomes important to extract meaningful collective variables. In this thesis, our primary objective is to implement hyperparameter optimization techniques, specifically employing genetic algorithm in the context of a semi-supervised autoencoder. The resulting best hyperparameters are tested for processing of molecular dynamics simulation trajectory data of two different proteins; MurD and adenosine kinase. By systematically evaluating the performance across these set of protein systems, the study aims to show the importance of hyperparameter tuning of autoencoders and specifically the significance of semi-supervised autoencoder in the context of collective variable extraction from diverse biomolecular dynamics scenarios. This thesis is organized as follows: Methods and Materials section details the implementation of the semi-supervised autoencoder, dataset preparation, hyperparameters, hyperparameter optimization techniques in general and the details of genetic algorithm employed in this study. The Results and Discussion section presents the resulting best hyperparameters. The performance of these best hyperparameters in the semi-supervised autoencoder for MurD and AdK are presented and compared. The resulting collective variables from latent space vectors are also presented for each protein. In addition, semi-supervised autoencoder is compared with the baseline autoencoder (vanilla autoencoder [65]) through reconstruction errors of both models and applying secondary structure analysis to each. In the last section, a summary of findings with future implications are discussed.

2. MATERIALS AND METHODS

2.1. Semi-Supervised Autoencoder

The classical autoencoder structure is a fully connected neural network as illustrated in Figure 1.1. It has an input layer, encoder hidden layers, a latent layer, and a symmetric set of decoder hidden layers. The input data gets transformed into a compressed representation through encoder layers in the latent layer. The latent layer, also known as the bottleneck layer, contains fewer nodes than the input layer for dimensionality reduction purposes. Using the compressed representation in the latent layer the decoder aims to reconstruct the original input.

The autoencoder used in this study, shown in Figure 2.1., is a semi-supervised autoencoder. It deviates slightly from the classical architecture through the integration of extra output nodes coming from the each encoder hidden layer and the latent layer. While the simulation trajectory is fed to the autoencoder in a unsupervised manner from the input nodes, extra information of the protein's dynamics is guiding the autoencoder through these extra output nodes. The autoencoder model is adjusting itself so that the output values from these extra nodes are the same as this extra information about the protein (such as opening angle in a folding protein). While the results of the decoded output nodes from decoder should approximate to the coordinates data from the trajectory, the extra output nodes from the encoder and latent layers should be the same as the opening angle or some other inherent information about the protein. This extra information results in improved latent space representation and helps the model to reconstruct the coordinates input better. This semi-supervised autoencoder model aims to enhance the autoencoder's capability of dimensionality reduction by improving the model prediction by adjusting the output according to these extra output nodes and result in better feature extraction performance.

The autoencoder model has been implemented using Python 3.8 with the Keras framework [66] together with the TensorFlow library [67]. For each protein under study, distinct train and test data sets are prepared from their simulation trajectories. After compiling the autoencoder, the training of the model is conducted using the fit function from the TensorFlow library. In supervised training, this fit function takes two inputs to train the model: the training data and the corresponding labels of the training data. When it comes to unsupervised autoencoder training, fit function takes two identical inputs which are the training data. This is because the autoencoder aims to result in output values that are exactly the same as input values. For the semi-supervised autoencoder model employed in this study, the inputs for the fit function are different. While one of the inputs remains as the training data which are the Cartesian coordinates of the simulation trajectory, the other input (designated as the expected output) is given as an array consisting of training data (coordinates) for the expected output of the decoder output nodes, and another order parameter such as opening angle. The autoencoder is expected to learn that the first set of training data (simulation trajectory) in this array are expected to be the same as the output nodes coming from the decoder. The latter sets of training data in different order parameters are the expected outputs from the extra nodes coming from the encoder hidden layer and the latent layers. These order parameters are specific to each protein under investigation and should be chosen from known order parameters representing the major conformational changes in the protein. For this purpose, the opening angle of the MurD protein and the NMP and LID angles of adenosine kinase is chosen as the designated expected outputs. A detailed discussion of these is given in the following sections.

2.2. Data Preparation

For the model development process, the dataset is split into training and test sets. The model is expected to learn from the training data and make correct predictions using unseen test data. After the model is compiled, the train data is used in the training process through the fit function so that the model learns from the train data

and it optimizes its parameters to make correct predictions. After training the model, the test set is used to evaluate the accuracy of the prediction capability of the model. The test set should be a new, unseen data which is distinct from the training set. In this study, the distinct training and test data sets are prepared from the MD simulation trajectory data of different proteins (MurD and adenosine kinase) independently to extract the collective variables. The obtained dataset for each protein underwent a systematic data preprocessing which will be discussed next.

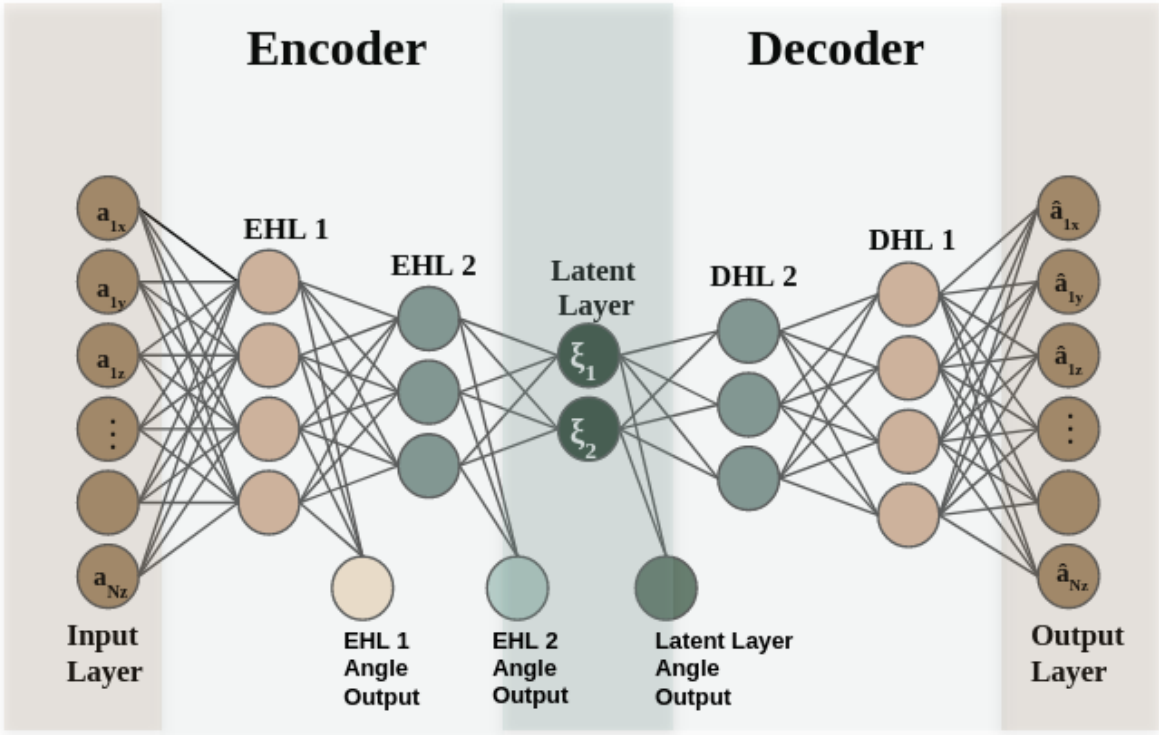

Figure 2.1. Semi-supervised autoencoder with extra output nodes coming from encoder hidden layers and latent layer.

2.2.1. Data Preprocessing

Data preprocessing is a crucial step in the development of machine learning models for constructing accurate models. The use of clean and standardized data is required for correctly understanding the patterns and relationships in the data. In this study, the Cartesian coordinates and dihedral angle data from the simulation trajectories along with opening angle information are used for different cases of different proteins

to train and test the autoencoder. Due to the different units these features have, a standardization is required to prevent certain features from dominating the model because of their larger magnitude. The normalization techniques are developed and used extensively for overcoming these issues [68]. The min-max normalization technique is used in this study to overcome the problem of different units in data. This normalization method was chosen to standardize the numerical features and it ensures that all values are within the range of 0 to 1. The applied min-max normalization formula [69] given below

$$X_{\text{normalized}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}}, \tag{2.1}$$

where X_{min} and X_{max} are the minimum value and maximum values, respectively. The dataset in this study was effectively rescaled in the range $[0, 1]$ using the MinMaxScaler function from the Scikit-learn library [70].

For the training data features, two cases are tested for both of the proteins. In case one, the input features are chosen as the x, y, z coordinates of the Cα atoms for each residue number, R, making the dimension of input nodes as $3R$. In case two, the input features are chosen as the coordinates data and the sine-cosine pairs [71, 72] of phi, ϕ, and psi, ψ, dihedral angles corresponding to each frame, making the dimension of the input nodes as $3R + 2(\phi + \psi)$.

2.2.2. MurD

MurD (UDP-N-acetylmuramoyl-L-alanine-D-glutamate ligase) is an enzyme required for the biosynthesis of bacterial cell wall, peptidoglycan (Figure 2.2) [73]. It has been investigated for understanding the bacterial resistance mechanisms, thus contributing to antibiotic development [74, 75]. Structural studies by Bertrand and his group have revealed crystalline snapshots of this protein in atomic resolution for its open, unbound state (PDB ID: 1E0D) [76] and its closed state bound to substrates (PDB ID: 3UAG) [77].

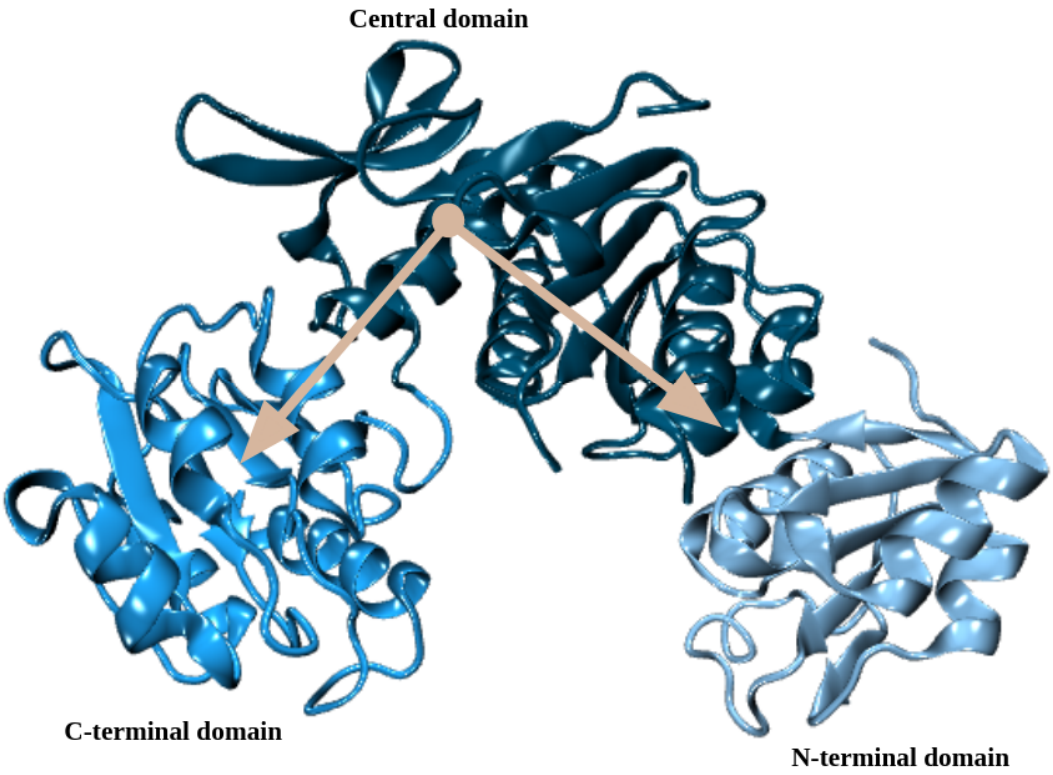

Figure 2.2. Tertiary structure of MurD shown with its three domains: N-terminal, C-terminal and central domains. The opening angle of the protein is demonstrated by the two vectors.

Degiacomi has contributed to our understanding of MurD by employing molecular dynamic simulations of its open, closed, and apo state (closed-state with its ligands removed) [78]. He used these simulation trajectories for training an autoencoder for the purpose of reconstructing similar conformations to explore new conformations. The simulation trajectory dataset of these simulations was obtained from Degiacomi to be used in this study for the training and testing phases of the autoencoder. MurD consists of 437 amino acid residues with three distinct domains; N-terminal (1-93), central (94-298), and C-terminal domains (299-437). Before the simulations, Degiacomi reconstructed the missing residues present in the crystal structure of closed (residues 221-224 and 242-244) and open structures (residues 221, 222, and 183-188) using MODELLER [79]. After running the simulations, Degiacomi prepared a dataset of structures by extracting the frames from the simulations at every 100 ps. For the closed and open structure, he prepared 2607 and 1913 frames, respectively. For the closed-apo state,

he combined 1000 frames for each 100 ps from a 100 ns simulation. It was reported that the closed and open structures exhibited stability throughout their simulation trajectories, while the apo state demonstrated a transition from closed to open conformation. Therefore, the training set for the autoencoder was prepared by combining two different simulations of closed and open states. The test set was constructed using the simulation data of the transition from close to open structure, namely apo state.

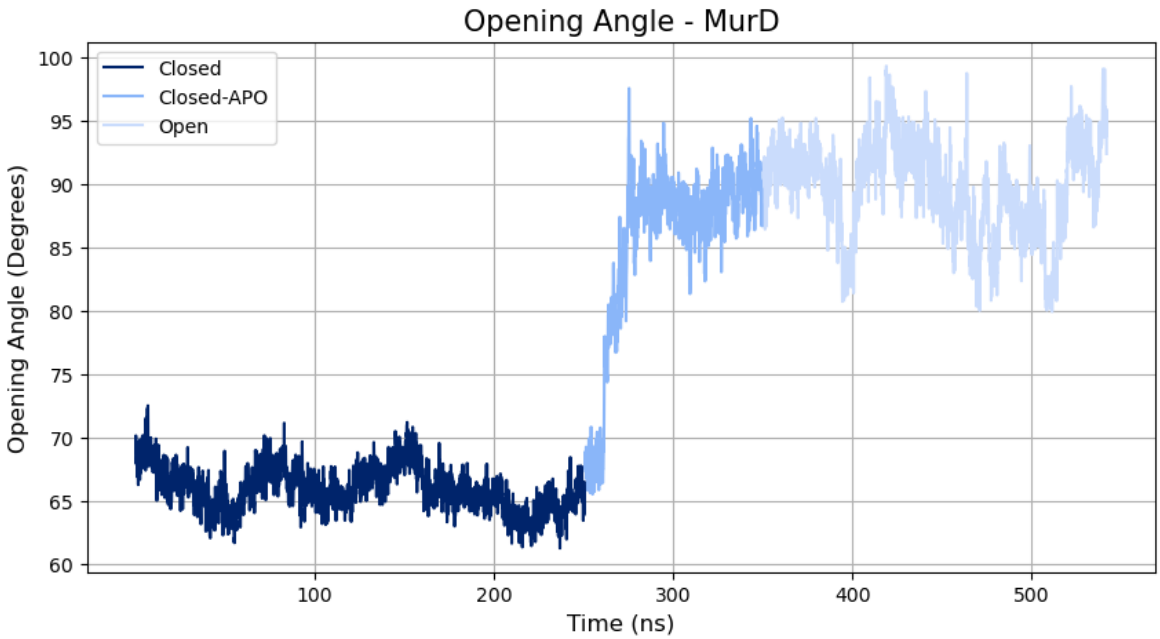

Figure 2.3. Angle of opening for MurD is plotted with respect to time for closed, closed-apo and open state.

For each set of data, a set of 1311 features was prepared using the Cartesian coordinates of 437 Cα atoms. In case one, these coordinate data were given to the input layer of the autoencoder and passed through the encoder layer to the latent layer. In case two, along with the 1311 features coming from the coordinates, the sine and cosine pairs of dihedral angles are also prepared as input data making 3055 features in total. The resulting decoded output is expected to be highly similar to the original input features. The output nodes coming from the encoder hidden layers and latent layers are expected to be similar to the opening angle of the MurD (shown in Figure 2.2) for each frame. The opening angle was computed for frames of all simulations by determining the angle between the center of mass of three groups of atoms (residues

130-230, 230-299, 299-437) using Gromacs [80]. The opening angle values per frame for closed, closed-apo and open states with their corresponding frames are given in Figure 2.3. The purpose of this extra information of opening angle is to increase the performance of the autoencoder for extraction of collective variables.

2.2.3. Adenosine Kinase

Adenosine kinase is 214-residue monomeric protein found in all three kingdoms of life [81]. It regulates the intracellular and extracellular adenosine levels by catalyzing its phosphorylation. Adenosine kinase uses a phosphate group from adenosine triphosphate (ATP) to convert adenosine into adenosine monophosphate (AMP) [82].

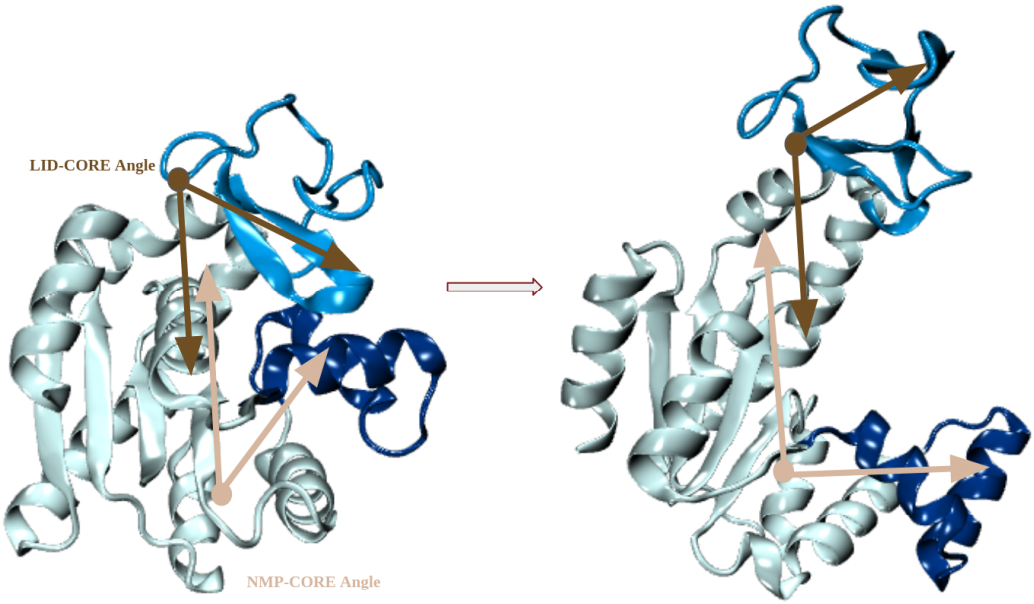

Figure 2.4. Cartoon representation of Adenosine kinase protein with its closed (left) and open (right) forms. LID-CORE and NMP-CORE angles are demonstrated by arrows.

Cellular adenosine levels are important in many metabolic pathways and by controlling the AdK activity, adenosine levels can also be controlled. For this reason, AdK is a potential drug target [83]. Thus, understanding its structural dynamics is important. It consists of three domains: stable CORE domain, ATP-binding LID domain

(residues 118-167) and an AMP-binding NMP domain (residues 30-67) [84]. During its catalytic cycle, it undergoes a transition from open (PDB ID: 4AKE [85]) to closed (PDB ID: 1AKE [86]) state with or without a substrate (Figure 2.4). AdK goes from open to closed state through the motions of LID and NMP domains in subsequent order where LID closure followed by the closure of NMP domain [87].

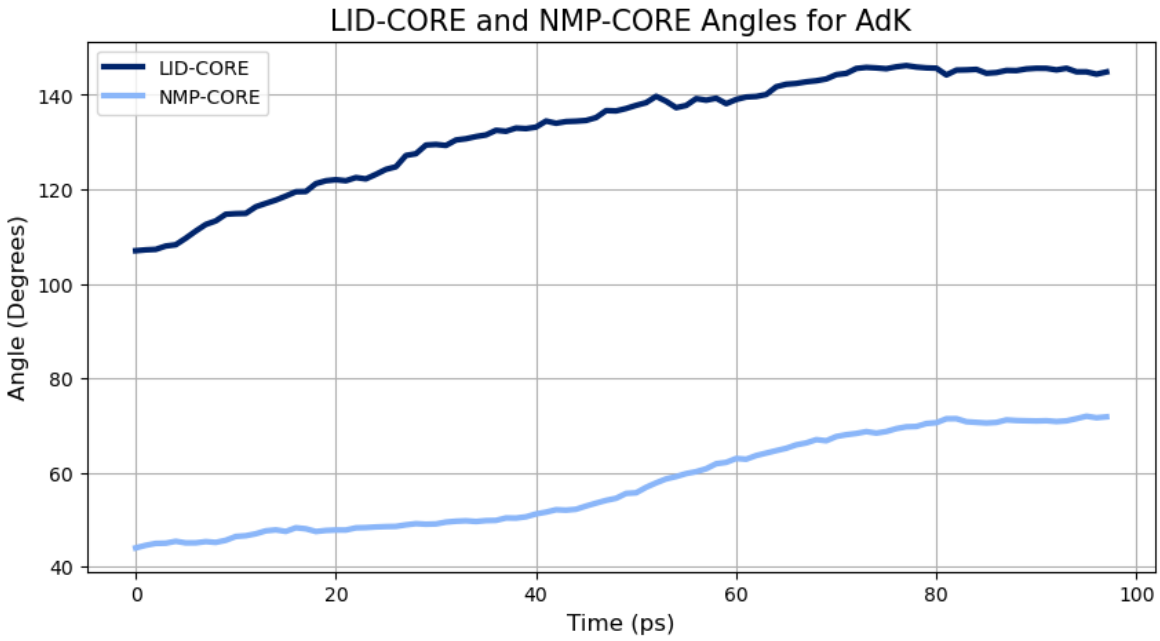

Figure 2.5. LID-CORE and NMP-CORE angles for AdK shown.

The RMSD between the open and the closed state is around 7 Å. This distinct conformational states make AdK a suitable protein for the extraction of collective variables from autoencoders. Therefore, the simulation trajectory of AdK was obtained from the publicly available simulation trajectory data present in the MDAnalysis Python toolkit that was developed by Beckstein and his colleagues [88, 89]. The simulation of AdK is showing a transition from close to open state by DIMS (dynamic importance sampling) method [90]. DIMS is a biased sampling method which is used in the simulation of proteins where there is a transition between two metastable states. The close to open transition of apo-AdK happens without requiring a ligand. Therefore, it was chosen for a test case for DIMS simulation. This method enables a quick transition from close to open state and resulting in a trajectory consisting of 98 frames with each frame corresponding to 1 ps. This 98 ps simulation was obtained from MDAnalysis toolkit

and then only the Cα atoms are selected for the downstream data preparation. The train and test data for the AdK was prepared in a similar manner to MurD. The first and last 40% of trajectory corresponding to the closed and open states, respectively. And this 80% of trajectory was used as the training data while the transition state making the remaining 20% was used a test data. For semi-supervising the autoencoder and improving its collective variable extraction, the LID-CORE and NMP-CORE angles (Figure 2.5) are used in a similar manner to opening angles from MuRD. There is one extra output nodes coming from the latent layer and encoder hidden layers in MurD for the opening angle. However, in the case of AdK, since there are two angles involved, the number of extra output nodes coming from the layers is two in total and each corresponding to LID-CORE or NMP-CORE angle. The tested cases for the AdK again similar to MurD, where x, y, z coordinates of Cα for 214 residues making 642 features with 98 frames in total for case one. And for the case two, along with 642 features coming from coordinates, the sine and cosine pairs of ϕ and ψ angles are used making the features 1494 in total.

2.3. Loss Function

The simulation trajectory of proteins is used as the input of the autoencoder model in the present study. The trajectory of the simulation system, $a(t)$, is defined by the Cartesian coordinates (x, y, z) of N atoms denoted as

$$a(t) \equiv a_{1x}(t), a_{1y}(t), \ldots a_{Nx}(t), a_{Ny}(t), a_{Nz}(t). \tag{2.2}$$

The dimension of the simulation trajectory, $3N$, determines the dimensions of the input layer in the autoencoder. The encoder takes the input coming from the simulation trajectory, $a(t)$, and passes it through the hidden layers to the latent layer. There, $a(t)$ gets mapped into a vector of latent variables with length d, $\xi = (\xi_1, \xi_2, ..., \xi_d)$, where $d < 3N$. This vector constitutes what we call the collective variables. The decoder component then uses this vector of latent variables to produce an output trajectory, $\hat{a}(t)$, similar to the input trajectory provided. A loss function is defined as a crucial performance metric of the autoencoder for the quantification of the difference between the original input and its reconstructed counterpart. The autoencoder adjusts its

parameters during the training process by minimizing this loss function. The loss (L) measured using the root-mean square error (RMSE) is given by

$$L(\hat{a}(j\Delta t), a(j\Delta t)) = \sqrt{\frac{1}{M}\sum_{j=0}^{M}[\hat{a}(j\Delta t) - \omega_j a(j\Delta t)]^2}, \tag{2.3}$$

where M is the number of frames in the trajectory and Δt denotes the time interval for storing the trajectory and ω_j represents the feature weights corresponding to each feature [91]. The main goal of an autoencoder is to reduce this loss function. For the semi-supervised autoencoder model, the loss function is a combination of the decoded coordinates output and the opening angle output from the extra output nodes. To account for the scale of importance for these two types of outputs, a penalty term, $k = 0.1$, is multiplied with the loss of angle outputs to reduce the impact of opening angles with respect to the coordinates. With this property, the semi-supervised autoencoder will adjust itself based on this combined loss function

$$L_{total} = L_{coordinates} + k \cdot L_{angles}. \tag{2.4}$$

The loss function can be improved by adjusting the configurational parameters of the architecture of the autoencoder model specific to the features used in the training. These configurational parameters are also called hyperparameters. Their importance and the algorithms for fine-tuning them for the best performance of the loss function will be discussed in the following sections.

2.4. Hyperparameters of Autoencoders

Hyperparameters are important for the performance of the autoencoder models. The hyperparameters investigated in this work include the activation function, regularizers, learning rate and momentum of the optimizer function, epoch number, batch size and the number of layers, and the number of nodes for each layer. The details for these hyperparameters are explained below.

2.4.1. Activation Functions

A node in a neural network is similar to a neuron in the human brain and it acts like a processor unit with an activation function transforming the input signals into output signals. An activation function can be a linear or non-linear transformation function applied to each node in the autoencoder and it calculates the output for that node given the inputs and weights entering into the node. The type of activation function affects the accuracy of the prediction. Due to the non-linear and complex relationships in the datasets, the non-linear activation functions are preferred [92]. The activation functions in this type will be discussed here: the sigmoid function, rectified linear unit (ReLU) and scaled exponential linear unit (Figure 2.6). Activation function can be used as a hyperparameter to be optimized. In this study, the activation functions sigmoid, ReLU and SELU was tested for encoder and decoder nodes to find the best performing activation function for the model used here. The activation function for the extra output nodes coming from the encoder hidden layers and the latent layer were set to sigmoid as default.

2.4.1.1. Sigmoid Function. The sigmoid function is a non-linear activation function used for mimicking the probability values in the output because it has output values between 0 and 1. The sigmoid function formula [93] is shown as

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{2.5}$$

Although it is popular in many neural network applications, it has drawbacks due to the vanishing gradients problem where the gradients become too small when the inputs move away from zero. With too small gradients, time spent for the model convergence increases. To solve this problem other non-linear activation functions are introduced.

2.4.1.2. Rectified Linear Unit (ReLU). ReLU is the most effective and widely used activation function given by Equation (2.5) [94–96]. It ensures that only certain sets of nodes are activated at the same time with the formula

$$ReLU(x) = \max(0, x). \tag{2.6}$$

ReLU is a solution to the vanishing gradient problem [97]. Although it is widely used and effective, there is an issue with ReLU which is called the dying ReLU problem where the negative input of the node becomes zero and from that point, these nodes will not be activated anymore [98]. To overcome this other activation functions are used that introduce negative slopes for the input values less than zero.

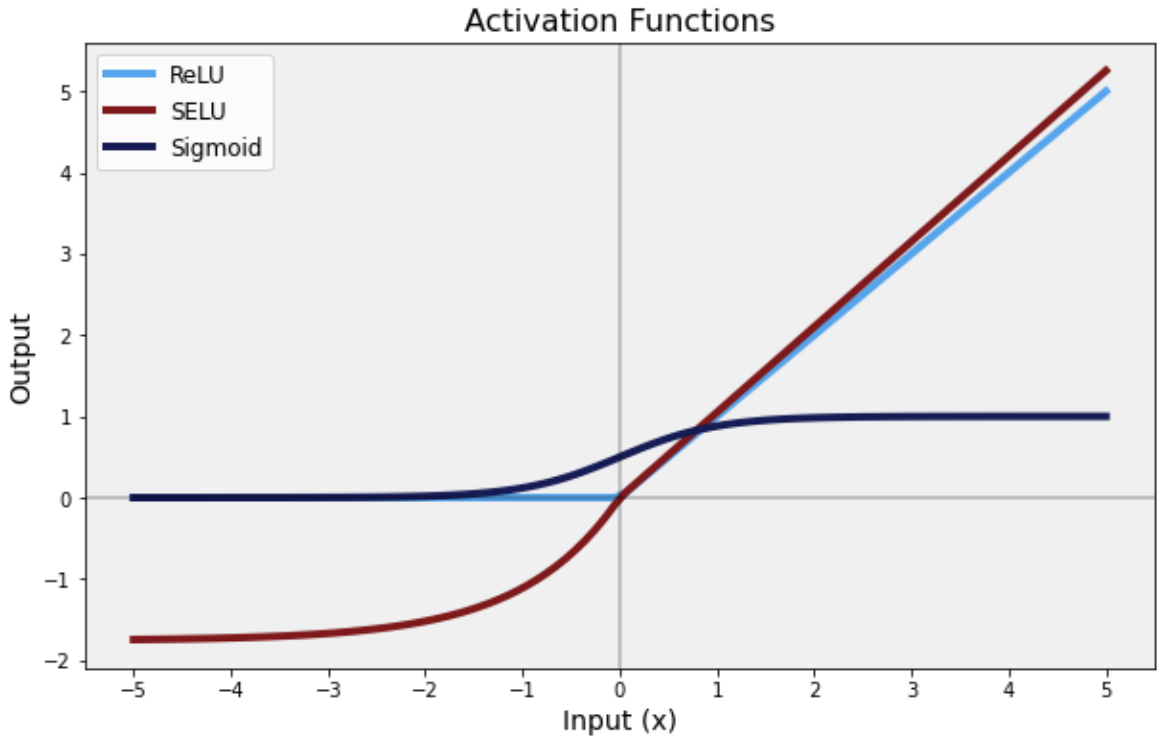

Figure 2.6. Different types of activation functions are shown: Sigmoid, ReLU and SELU.

2.4.1.3. Scaled Exponential Linear Unit (SELU). SELU is another non-linear activation function, introduced by Klambauer et al. [99]. It is designed to produce a normally distributed output with

$$\text{SELU}(x) = \begin{cases} \gamma \cdot x, & \text{if } x > 0 \\ \gamma \cdot \alpha \cdot (e^x - 1), & \text{if } x \leq 0 \end{cases}, \tag{2.7}$$

where γ is scaled constant and set to 1.0507 and while α is 1.6732. Like ReLU, it also eliminates the vanishing gradients problem. In addition, it is better to deal with the problem of dying ReLU by introducing negative slopes for the input values below zero.

2.4.2. Learning Rate

The learning rates are the step sizes taken for updating the weights. It is particularly important to choose a correct learning rate because a higher size can provide a fast convergence but may not result in minimizing the error correctly. A lower learning rate is best for minimizing the error but may get trapped in local minima. Correctly updating the weights are crucial in getting a converged model with a minimum error. The weights are the numerical values associated with each input signal entering a node. The importance of an input feature for predicting the output values is determined through these weights and they get updated during the model training process. The weights also carry the information of how strong an input value is in determining the output of the node. During the training process of the model, the weights are adjusted dynamically to predict the correct output. These adjustments are carried out through learning algorithms such as backpropagation [100] along with an optimization technique such as gradient descent [101]. During the training, the information flows through the network in the direction of from input to outputs. Then the resulting output is compared to the expected output and the error between them is back propagated to the network to update the weights accordingly. The optimization technique is responsible for how these weights are updated. Although the gradient descent optimization method is the most common approach used in training neural networks, it has some challenges such as slow convergence rate.

Other optimization algorithms such as Adam (Adaptive Moment Estimation) [102] are developed to overcome these and it is the most recommended optimization method [103]. Contrary to the gradient descent algorithm's constant learning rate, Adam uses adaptive learning rates. Given the performance of the Adam optimizer over others, it is chosen as the optimization method for the training of the autoencoder for this study. The configuration parameters of Adam optimization such as learning rate are chosen as one of the hyperparameters to be tested. The hyperparameter values tested for learning rate are given in Table 3.2. Due to the implementation of Adam optimization method, these given learning rates are initial values and are adapted based

on the momentum components of the optimizer which will be discussed next.

2.4.3. Momentum

Adam optimizer adjusts the learning rates based on the exponentially decaying average of past gradients (m_t) and the exponentially decaying average of past squared gradients (v_t) given in

$$\begin{aligned} m_t &= \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t \\ v_t &= \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2 \end{aligned}, \tag{2.8}$$

where g_t is the gradient and β_1 and β_2 are exponential decay rates for the first and second moment estimates, respectively [103]. When these decay rates are small, it is observed that m_t and v_t are biased towards zero. Authors corrected these equations and those bias-corrected versions are given in

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1-{\beta_1}^t} \\ \hat{v}_t &= \frac{v_t}{1-{\beta_2}^t} \end{aligned}, \tag{2.9}$$

which are used in the update rule of Adam optimization in

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + c} \cdot \hat{m}_t, \tag{2.10}$$

where η is the learning rate and c is a small constant to prevent division by zero. As a hyperparameter to be optimized, the β_1 is selected to be tested in this study. The values tested for β_1 are given in Table 3.2.

2.4.4. Regularizers

The aim of training a neural network is to learn from the training dataset to make generalized predictions on the unseen test dataset. When the model learns every detail of the training data too much, then its prediction performance on the test data declines and this is called overfitting [104]. Overfitting usually happens when the variance of the training dataset is high and the model learns every detail of the dataset. To address the problem of overfitting, regularization methods can be used to introduce a penalty term to the loss function using weights. The most commonly used regularization methods

are LASSO (least absolute shrinkage and selection pperator) regularization, L1 [105], and Ridge regularization, L2 [106]. L1 regularization adds a penalty term to the loss function using the sum of absolute values of the weights, ω, and the loss function in Equation (2.3) becomes the given formula in

$$L(\hat{a}(j\Delta t), a(j\Delta t)) = \sqrt{\frac{1}{M}\sum_{j=0}^{M}\left[\hat{a}(j\Delta t) - \omega_j a(j\Delta t)\right]^2} + \lambda\sum_{j=0}^{M}\left|\omega_j\right|, \tag{2.11}$$

where λ is the regularization parameter. For L2 regularization, the penalty added to the loss function is the sum of the squared values of the weights, and the loss function becomes the formula below

$$L(\hat{a}(j\Delta t), a(j\Delta t)) = \sqrt{\frac{1}{M}\sum_{j=0}^{M}\left[\hat{a}(j\Delta t) - \omega_j a(j\Delta t)\right]^2} + \lambda\sum_{j=0}^{M}\omega_j^2. \tag{2.12}$$

The choice of λ is important for controlling the effect of regularization on the loss function. When λ is too high, the model becomes underfit where the model is too simple to learn the training data. When it is zero, then the model can overfit the data. λ is an important hyperparameter that is investigated in this study to find the optimum value for the autoencoder model. The values tested for λ are listed in Table 3.2.

2.4.5. Hidden Layer Number

The design of neural network topology plays an important role in the performance of the model [107]. The number of hidden layers determines how deep the network is. The depth of the network increases along with the complexity while the processing speed decreases. Therefore, the optimum topology should be determined for a neural network considering the dataset [108].

In the autoencoder architecture, the number of hidden layers in the encoder and decoder are the same. There is no rule of thumb to determine how many hidden layers an autoencoder should have. Therefore, the search for an optimum hidden layer number should be done systematically based on the nature of the dataset. For this purpose, different cases are tested with different hidden layer numbers in the hyperparameter search algorithm implemented in this study. The tested set layer numbers are $(1, 2, 3, 4)$.

2.4.6. Hidden Layer Dimensions

The other component of neural network topology is the nodes present in each hidden layer. If the node number is dense, such a topology would lead to overfitting [109]. The number of nodes in each layer, the dimension of the layer, should be determined carefully. In general, the number of nodes should be less than the number of input features in the autoencoder [110] . This is because the aim of the autoencoder is dimensionality reduction. The nodes in the layers should decrease towards the latent layer so that the feature extraction and reduced dimensions would become possible. The number of nodes in the hidden layers should be determined systematically and they are included in the hyperparameter search space of the hyperparameter algorithm used in this study. The tested set of hidden layer nodes for each layer is given in Table 3.2.

2.4.7. Latent Layer Dimension

The autoencoder latent layers are where the compressed information about the input data is represented. The dimension of the latent layer is important for correctly capturing the features in the input data. Since autoencoder is a dimensionality reduction process, then the dimension of the latent layer should always be much smaller than the dimension of input data and smaller than the encoder and decoder hidden layer dimensions [111]. Given this, researchers identify the latent layer dimension through intuition based on their experience and the intrinsic nature of the input data. As an automated approach to find the latent layer dimension, hyperparameter optimization methods can be applied. Therefore, the latent layer dimension is used as a hyperparameter to be explored in this study. The range of values tested is given in Table 3.2.

2.4.8. Epoch Number

Epoch refers to the number of complete passes through the training dataset [112]. Usually, a single epoch is not sufficient for the model to learn the data and the model needs multiple learning cycles to result in an effective performance. Therefore, multiple

epochs should be used. However, training with a high number of epochs would lead to overfitting of the input data and poor performance on the test data. For that reason, the number of epochs is a hyperparameter and should be determined carefully. In this study, different numbers of epochs are tested to find the optimum for the given dataset. The tested numbers are listed in Table 3.2.

2.4.9. Batch Size

The training data is usually divided into smaller subsets known as batches. In one epoch, the data is divided into batches and then passed through the network in several iterations depending on the size of the input data and the batch. It was observed that training deep neural networks with smaller batch size performs better in terms of generalization [113]. In the context of autoencoders, Wang et al. reported that the smaller batch size is better in predicting the reconstructed output [114]. However, they investigated the effect of batch size on a regression task using an autoencoder. To find the optimum batch size for the autoencoder model for the dimensionality reduction purposes, different batch sizes are explored in this study as a hyperparameter. The investigated batch sizes are presented in Table 3.2.

2.5. Tuning Hyperparameters of Autoencoders

The goal of hyperparameter optimization is to find the global optimum x^* of an unknown, often complex, black-box function f. This function $f(x)$ can be evaluated for any arbitrary input x within a defined hyperparameter space X. The hyperparameter space X can be categorical, discrete, and continuous variables. Hyperparameter optimization approaches aim to explore the hyperparameter space to find the set of parameters x^* that maximizes the function f, given below [115]

$$x^* = argmax_{x \epsilon X} f(x). \tag{2.13}$$

This process involves evaluating the performance of the model with different hyperparameter configurations and identifying which configuration yields the best results. The evaluation criteria are often based on model accuracy or predefined performance

metrics. In the context of this study, the objective of hyperparameter optimization is to find the best hyperparameters of the autoencoders discussed above for improving the extraction of collective variables from the molecular simulation trajectory data of proteins.

An automated hyperparameter optimization is necessary for reproducibility of scientific studies. Early studies for hyperparameter tuning revealed that each different dataset requires a different combination of hyperparameters [116]. Contemporary studies showed that optimized hyperparameters yield superior results compared to the default configurations provided by the mainstream machine learning libraries [117–119]. Several approaches are used for hyperparameter tuning. The most basic approach is the grid search that involves comprehensive evaluation of every possible combination of hyperparameter set [97]. Given enough time and resources, grid search finds the global optimum. However, the number of evaluations needed grows exponentially with the dimensionality of the hyperparameter space causing slower convergence. As an alternate solution to this, random search can be utilized with faster convergence than grid search in finding the best hyperparameters [120]. It randomly samples configurations within the hyperparameter space with a certain range. However, random selection of hyperparameters would lead to the missing of optimum parameters. Both methods offer a good start to hyperparameter optimization but they can be exhaustive when applied to complex models with large hyperparameter space demanding significant computational resources. More sophisticated methodologies are needed to overcome this issue. Advanced techniques like evolutionary algorithms and probabilistic models are employed to fine tune hyperparameters in neural networks [**?**,121]. In the scope of this study, the genetic algorithm (GA), an evolutionary algorithm, is implemented to optimize the hyperparameters of the semi-supervised autoencoder for the extraction of collective variables from the MD simulation data of MurD and adenosine kinase proteins. The implementation and theory of this method will be explained in the following section.

2.6. Optimization with Genetic Algorithm

Evolutionary algorithms are population based algorithms that mimic the process of natural selection, a concept co-discovered by Darwin and Wallace [122,123], to find an optimum solution to a given problem. Genetic algorithm, introduced by John Holland [124], is a subset of evolutionary algorithms and used extensively for optimization problems [125–127]. The aim of the genetic algorithm is to find the best performing solution set for a given optimization problem. Each combination of parameters for a solution is considered a chromosome. Genetic algorithm follows a set of steps (see Figure 2.7) that are performed iteratively until a stopping criteria is met.

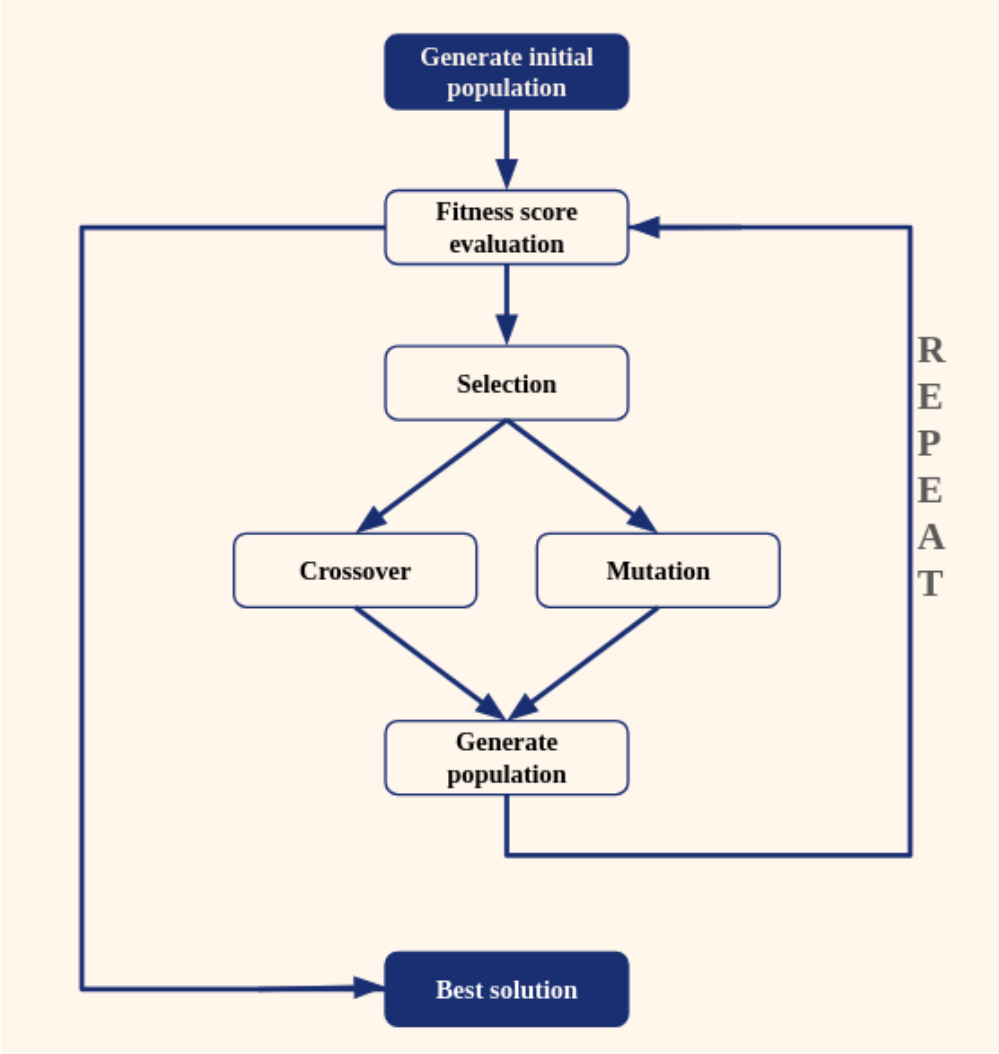

Figure 2.7. Genetic algorithm diagram.

After starting with a randomly generated population with size n, a fitness score for each solution set is calculated. The fitness score is a measure of how suitable each solution is for the given problem. In the selection step, the most promising solutions with the highest fitness score are called parents and they are carried to the next generation for further processing such as crossover or mutation. Crossover involves random recombination of the parameters of the parent solutions to generate new offspring.

In the case of mutation, it introduces random changes to the parameters of the parent solutions again resulting in new offsprings. This process generates a new generation of solutions until n of population size is reached. After that mutation and crossover processes stop and the same cycle of fitness evaluation and evolution applied to the new generation. The iterative process continues until a specified termination condition is met such as reaching a maximum number of generations or achieving a desired level of fitness. Ultimately, the algorithm converges towards the fittest solutions, which are then presented as the optimal or near-optimal solutions to the original optimization problem.

Although the genetic algorithm is powerful for finding optimal solution, there might be still some issues due to the process of crossover and mutation. This is because these methods are inherently random and might cause very low fitness score in the new offspring. As a solution to this issue, Kenneth De Jong introduced elitism [128]. Elitism ensures that there are chromosomes in the population who are not undergoing a crossover or mutation process. Elites are selected from the intermediate scoring chromosomes and passed to the next generation without crossover or mutation (Figure 2.8).

In the context of this study, the genetic algorithm is utilized for the hyperparameter optimization of semi-supervised autoencoder with the aim of extracting collective variables of MurD and adenosine kinase using geneticalgorithm2 of PyPI [129]. The geneticalgorithm2 has a built-in function that takes the objective function (function to optimize), the hyperparameter boundaries and input along with the genetic algorithm parameters such as generation and population size. As a fitness score, the output of the

objective function is calculated given the hyperparameters of the autoencoder model using root-mean-squared error of normalized predictions of coordinates with angles using test dataset. After the autoencoder is compiled with the given hyperparameter set and trained with the training data, the resulting autoencoder model is given the test data of coordinates to predict the coordinates along with the angles from the extra output nodes from the EHL and latent layer. The resulting predictions that are between 0 and 1 are then compared to the normalized values of actual test coordinates and the actual angles using RMSE. Using the proteins in this study and the genetic algorithm, a systematic optimization was conducted for each protein. The resulting values will be presented in the Results & Discussion section.

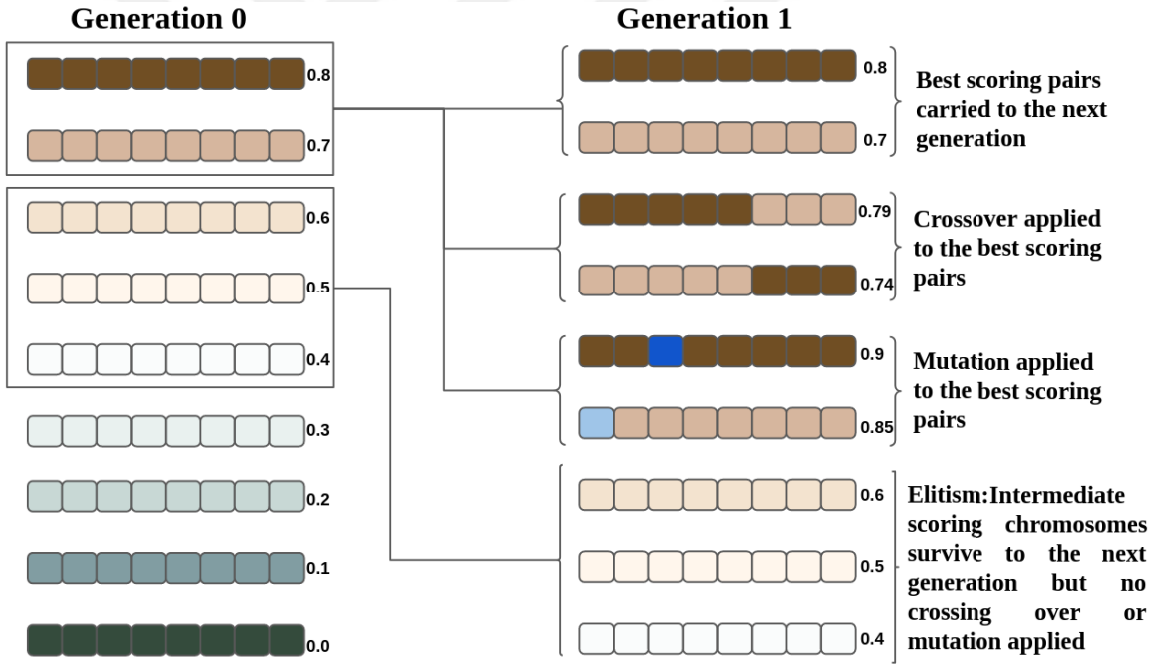

Figure 2.8. Crossover, mutation and elitism in genetic algorithm.

2.7. Performance Metrics

Performance metrics are developed to evaluate the objective function for the hyperparameter optimization method. Inside the hyperparameter optimization algorithm, the autoencoder is trained with the train data given an hyperparameter set. And then, the resulting autoencoder model is tested using the test data. Thus, from the reconstructed results and the actual test data, an objective function is calculated for the

optimization algorithm to act on. The objective function is obtained by calculating the sum of normalized RMSE of decoded coordinates, opening angles, and dihedral angles (for case two only) and using a penalty term $k = 0.1$ so that the objective function is similar to the loss function. The objective function for both of the optimization methods used in this study is the combined RMSE (cRMSE) of normalized coordinates, opening angles and dihedral angles which is given as

$$cRMSE = RMSE_{coords} + k \cdot RMSE_{\text{opening angles}}. \tag{2.26}$$

After obtaining the best hyperparameters from the optimization algorithms, next step is to train the autoencoder using these hyperparameters and training data. After the model is compiled and fit, the test data is given to the model for prediction. The performance of the prediction is measured by the RMSE of decoded output consisting of cooordinates (and dihedral angles for case two) is calculated. Using the RMSE of coordinates, residue per RMSE (rRMSE) is also computed as a metric for understanding the reconstruction performance of the autoencoder. The resulting latent layer vectors from the test data prediction, used as the collective variables and plotted against RMSD, radius of gyration, SASA (solvent accessible surface area) and the opening angles to understand whether the obtained CVs are representing the internal motions of the proteins being investigated.

2.8. Statistical Analysis

Statistical analysis applied in this study to understand whether the proposed semi-supervised autoencoder (SAE) architecture resulted in significantly better results than the basic vanilla autoencoder (VAE). To evaluate the statistical properties of the data obtained from both of the methods, initial analyses were conducted to check for the normality of the distribution. The Shapiro-Wilk test was used for this purpose for it is efficient in assessing the normality of data, particularly for small to medium sample sizes [130]. This test examines whether a sample comes from a normally distributed population by comparing the arrangement of data with that of a normal distribution.

For each data set, the Shapiro-Wilk test was performed using

$$W = \frac{\left(\sum_{i=1}^{n} a_i x_{(i)}\right)^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2}, \tag{2.27}$$

where x_i are the ordered sample values, $\bar{x}$ is the sample mean, and a_i are the coefficients generated from the expected values and variances of the order statistics of a standard normal distribution. If the Shapiro-Wilk tests resulted in p-values less than 0.05, leading to the rejection of the null hypothesis that the data are normally distributed. This outcome necessitates the use of non-parametric methods for further analyses since the assumption of normality was violated.

Subsequently, the Mann-Whitney U test was employed to determine if there were statistically significant differences between the two autoencoder architecture SAE and VAE. This test is appropriate for comparing two independent samples, especially when the data are not normally distributed, as it compares the medians of the groups rather than the means [131]. The Mann-Whitney U test calculates the test statistic, U, based on the ranks of the data in the combined samples, providing a robust comparison without the need for normality. The formula for the Mann-Whitney U statistic is

$$U = n_1 n_2 + \frac{n_1(n_1 + 1)}{2} - R_1, \tag{2.28}$$

where n_1 and n_2 are the sample sizes of the two groups, and R_1 is the sum of the ranks in the first group. A low value of U indicates a significant difference between the distributions of the two groups. Statistical significance was assessed at the 0.05 level, and all analyses were conducted using statistical software present in the Python's SciPy library [132]. The results of the statistical analysis will be presented in the Results & Discussion section.

2.9. Secondary Structure Analysis

The MD trajectory used in the semi-supervised autoencoder input consists of only Cα atoms. Before performing secondary structure analysis, it was essential to reconstruct the complete backbone atoms from the provided Cα coordinates. This reconstruction was accomplished using the PULCHRA software, which specializes in

rebuilding full-atom protein models from Cα trace data [133]. PULCHRA was selected for its efficiency in accurately reconstructing protein backbones, thereby facilitating a more detailed subsequent analysis. Following the reconstruction of the full protein backbone, both the input data and autoencoder-reconstructed output were subjected to secondary structure analysis using the Dictionary of Protein Secondary Structure (DSSP) algorithm [134]. DSSP is a widely recognized method for assigning secondary structure elements (SSEs) such as α-helices, β-sheets, and loops based on the hydrogen bonding patterns among the backbone atoms. The analysis was performed separately on each dataset to identify and compare the secondary structure elements throughout the protein trajectory. This comparison aimed to assess the impact of the autoencoder reconstruction on the preservation of biologically relevant structural features. The output from DSSP was quantitatively analyzed to determine the consistency of secondary structure assignments between the original and reconstructed output. The analysis included calculating the percentage of agreement in SSE assignments and visualizing the distribution of secondary structures along the protein sequence. The resulting analysis will be presented in the Results & Discussion section.

3. RESULTS AND DISCUSSION

In this section, the results from the hyperparameter optimization method, genetic algorithm, for the proteins under investigation will be presented and discussed. First, the genetic algorithm results will be shown for the two cases of MurD and AdK proteins. Then, the semi-supervised autoencoder will be compared with the vanilla autoencoder to present the efficacy of this new autoencoder architecture.

3.1. Hyperparameter Optimization with Genetic Algorithm: MurD

The genetic algorithm for each of the two cases of MurD was run for 30 generations with population size of 30. Genetic algorithm parameters used are given in Table 3.1. The objective function calculation for each hyperparameter combination is done with 10 repetition and the mean value is given to the genetic algorithm as the objective function value. The convergence plot for the genetic algorithm is shown in Figure 3.1.

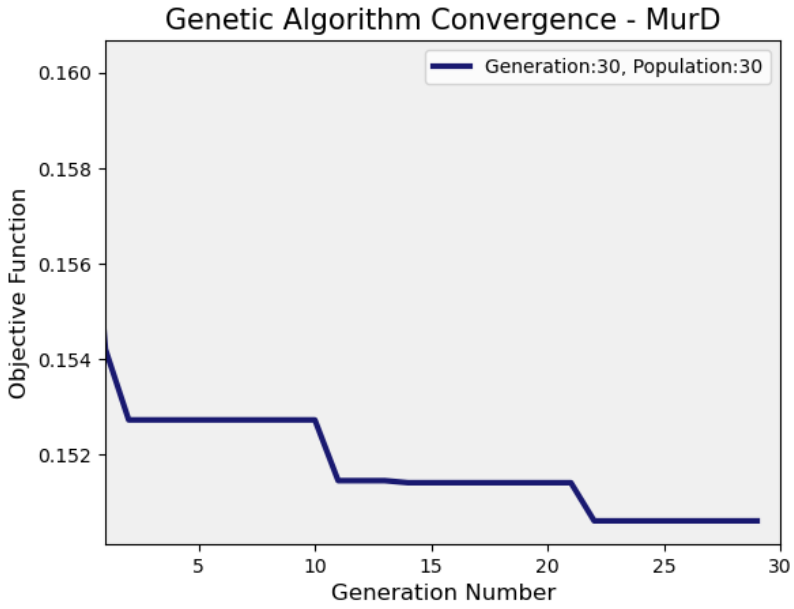

Figure 3.1. Convergence of genetic algorithm for MurD case two for 30 generations with 30 populations.

As the generation number increases, the objective function (cRMSE) is minimized so that the genetic algorithm converges more but with decreasing step sizes. Higher generation numbers are needed for better minimization of objective function. To increase the efficacy and speed of genetic algorithm, early stopping criteria is applied to the training process of the autoencoder [135]. With early stopping, when the performance of the autoencoder in a given validation data does not improve for some number of epochs, then the training halts without completing all the epochs defined. The minimum change quantity to be counted as an improvement is set to 0.0001. And the number of epochs with no improvements, also called patience, is set to seven. For this purpose, 20% of the training data is chosen randomly as a validation set to be used in early stopping.

Table 3.1. Genetic algorithm parameters.

Parameter Name	**Value**
Population Size	30
Number of Generations	30
Elite Ratio	0.01
Mutation Probability	0.1
Parents Portion	0.3

The hyperparameter space explored for MurD is given in Table 3.2 for case one and case two with minimal differences in range of hyperparameters explored. The best performing hyperparameter combination from genetic algorithm for each of two cases of MurD is listed in Table 3.3. Results showed that same activation function, regularizer, batch size and hidden layer are the best hyperparameters for both of the cases. The semi-supervised autoencoder is trained with these best hyperparameter combinations using the training data from MurD MD trajectory. Then, the fitted model is used to predict the test and train data separately. Using the reconstruction error from the decoded coordinates, the RMSE per residue for test and train data is calculated.

Table 3.2. Hyperparameters explored for MurD

Hyperparameters	**Case 1**	**Case 2**
Activation Function	Sigmoid ReLU SELU	Sigmoid ReLU SELU
Regularizer (λ)	1e-3, 1e-4, 1e-5	1e-3, 1e-4, 1e-5
Learning Rate (η)	1e-3, 1e-4, 1e-5	1e-3, 1e-4, 1e-5
Momentum (β_1)	0.7, 0.75, 0.80 0.85, 0.90, 0.95	0.7, 0.75, 0.80 0.85, 0.90, 0.95
Epoch	100, 200, 300 400, 500	100, 200, 300 400, 500
Batch Size	16, 32, 64 256, 512	16, 32, 64 256, 512, 1024
Hidden Layer Number	1, 2, 3, 4	1, 2, 3, 4
Dimension of Hidden Layer 1	100, 150, 180, 225, 250 300, 350, 410, 500, 650	100, 150, 180, 225, 265 350, 500, 1000, 1300, 1500
Dimension of Hidden Layer 2	20, 35, 60, 85, 110 130, 160, 200, 250, 300	20, 60, 85, 110, 130 160, 200, 250, 300, 500
Dimension of Hidden Layer 3	15, 18, 20, 25, 30 40, 50, 65, 80, 120	15, 20, 25, 30, 40 50, 65, 80, 120, 180
Dimension of Hidden Layer 4	5, 7, 9, 10, 12 15, 19, 24, 30, 40	5, 7, 9, 10, 12 15, 19, 24, 30, 40
Latent Layer Dimension	2, 3, 4, 5, 6 7, 8	2, 3, 4, 5, 6 7, 8, 9, 10

The resulting rRMSE graph for case one is presented in Figure 3.2. According to this graph, the training data performed better than test data as expected. The average rRMSE for train and test data are 0.83 Å and 1.57 Å, respectively. The highest peak for the test data is 10.04 Å at the residue 242. And the other peaks around residue 185 shows that the semi-supervised autoencoder performs poorly on these regions. These

peaks might be due to the inherent structure of the data where there are missing residues on those regions in the original data that was reconstructed by Degiacomi using MODELLER [78, 79]. In addition, the rRMSE values are low in the N-terminal domain while they are higher in the C-terminal domain. This behaviour is compatible with the RMSF (root-mean square fluctuation) per residue calculation for the closed-apo state which is the test data used in the predictions (Figure 3.3). Since the RMSF values for the C-terminal region is higher than the rest of the structure, predicting these highly unstable residues becomes challenging for the semi-supervised autoencoder. To make the autoencoder more informed about the structure and dynamics of the protein, the sine and cosine pairs of dihedral angle for each residue is given as an input feature along with the coordinates of Cα in case two.

Table 3.3. Best hyperparameters for MurD obtained from GA.

Hyperparameters	**Case 1**	**Case 2**
Activation Function	SELU	SELU
Regularizer (λ)	1e-4	1e-4
Learning Rate (η)	1e-3	1e-4
Momentum (β_1)	0.95	0.75
Epoch	100	300
Batch Size	32	32
Hidden Layer Number	1	1
Dimension of Hidden Layer 1	250	1500
Dimension of Hidden Layer 2	0	0
Dimension of Hidden Layer 3	0	0
Dimension of Hidden Layer 4	0	0
Latent Layer Dimension	8	9

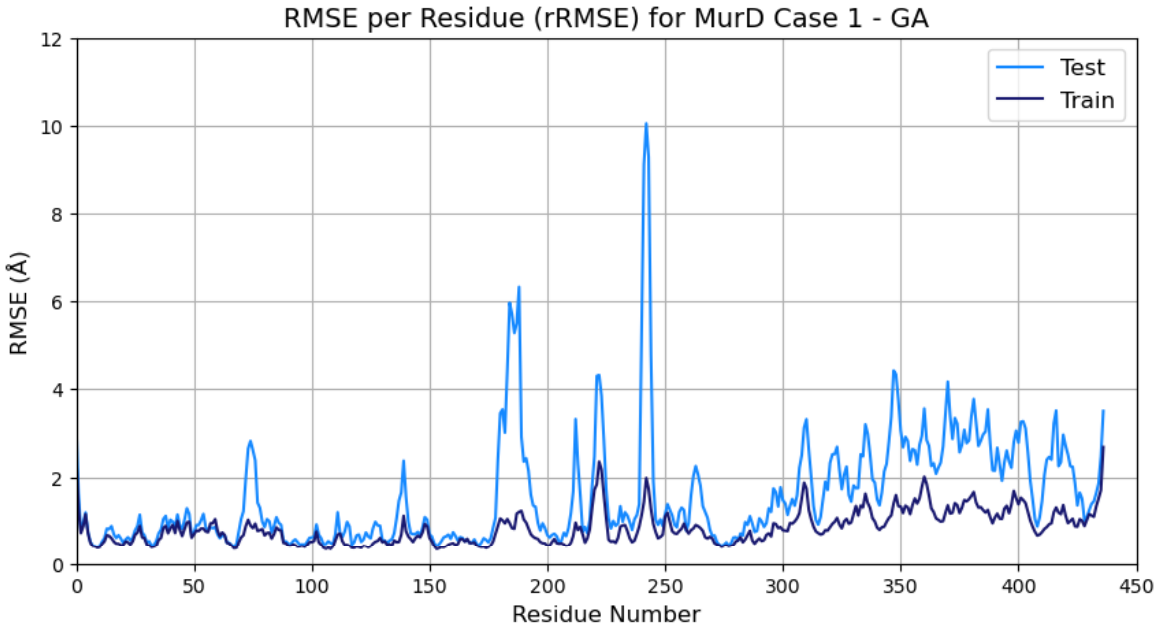

Figure 3.2. RMSE per residue (Å) for the MurD case one showing the prediction results for test (light blue line) and train (dark blue line) data.

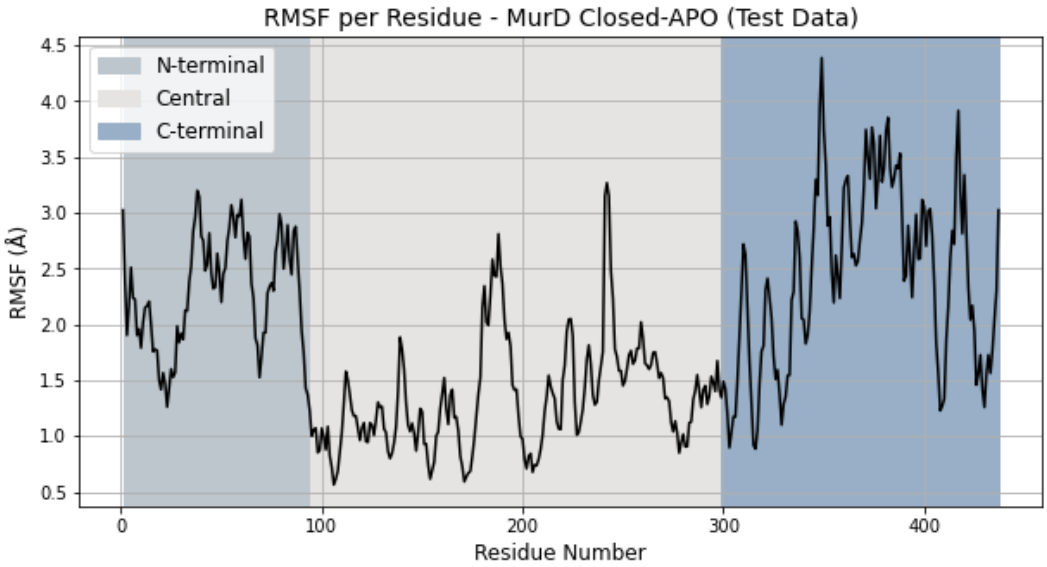

Figure 3.3. RMSF per residue (Å) for the MurD closed-apo state which is the data selected for the test data for the semi-supervised autoencoder. Residue regions are colored based on three domains of MurD.

The hyperparameter space for case two is slightly different than that of case one. Since the input features are increased from 1311 to 3055 for case two compared to case

one, the possible hidden layer dimensions are also changed accordingly (Table 3.2). The genetic algorithm is run with the parameters same as the case one (Table 3.1). For case two, the resulting best hyperparameters are shown in Table 3.3. Using these, the semi-supervised autoencoder is trained and then tested. The resulting rRMSE graph is shown in Figure 3.4 for train and test data.

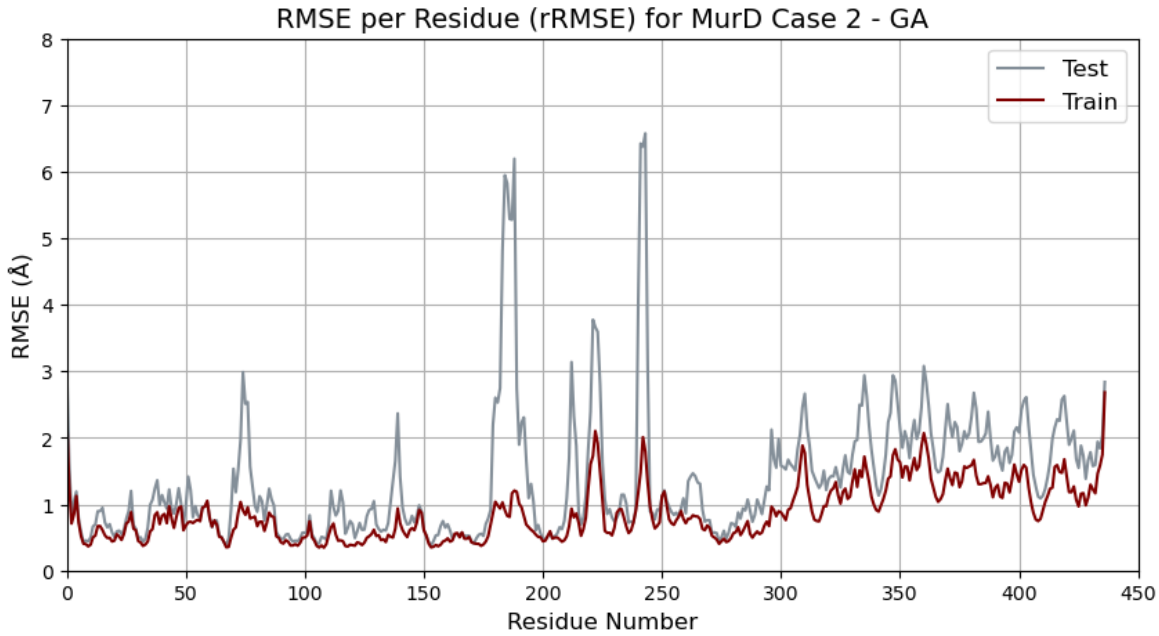

Figure 3.4. RMSE per residue (Å) for the MurD case two showing the prediction results for test (gray line) and train (red line) data. The best hyperparameter set is found by the genetic algorithm with 30 generations and 30 populations.

Average rRMSE values for train and test data are 0.87 Å and 1.37 Å, respectively. Again the training data performs better than the test data as expected. The peaks around residues 185 and 242 and the fluctuations in the C-terminal are consistent with the case one. A comparison of the test data rRMSE plots for two cases is given in Figure 3.5. In case two, the C-terminal prediction is better as well as the average rRMSE than the case one. The other RMSE values for reconstructed coordinates, opening angles, and others are compared for the two cases in Table 3.4.

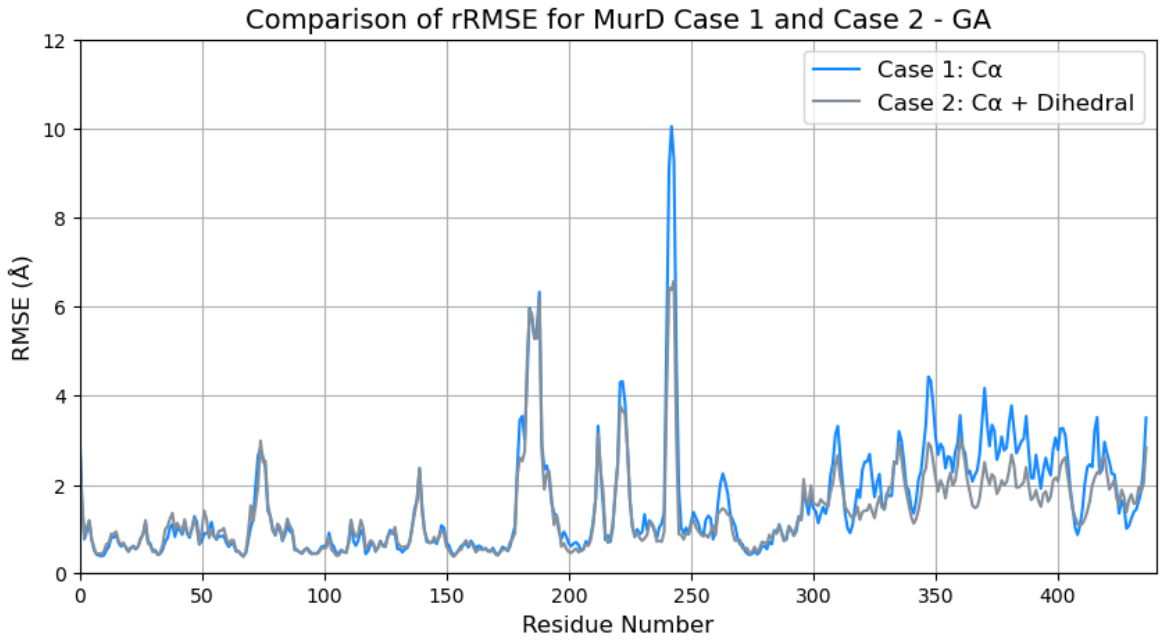

Figure 3.5. Test data prediction rRMSE comparison for case one (light blue line) and case two (gray line) of MurD using the hyperparameters obtained from running genetic algorithm for 30 generations with 30 populations.

Table 3.4. RMSE values for MurD case one and two with 30 generations and 30 populations.

RMSE Type	**Case 1**	**Case 2**	**Unit**
$rRMSE_{Avg}$ (Test)	1.57	1.37	Å
$rRMSE_{Avg}$ (Train)	0.83	0.87	Å
Coordinate $RMSE_{Avg}$ (Test)	1.162	0.97	Å
Coordinate $RMSE_{Avg}$ (Train)	0.53	0.56	Å
Opening Angle $RMSE_{Avg}$ from Latent Layer (Test)	1.36	2.35	Degree
Opening Angle $RMSE_{Avg}$ from EHL1 (Test)	1.41	1.73	Degree
Dihedral Angle $RMSE_{Avg}$ (Test)	-	0.23	Radian
Dihedral Angle $RMSE_{Avg}$ (Train)	-	0.16	Radian
cRMSE (GA Objective Function)	0.124	0.157	-

Likewise, when the absolute reconstructing errors are compared for case one and case two test data, the second case showed better results. This suggests that addition

of dihedral angle information improved the reconstruction of input data in the semi-supervised autoencoder output. Again the errors in the C-terminal region and residues 183 and 242 are present in both of the cases but improved in the second case.

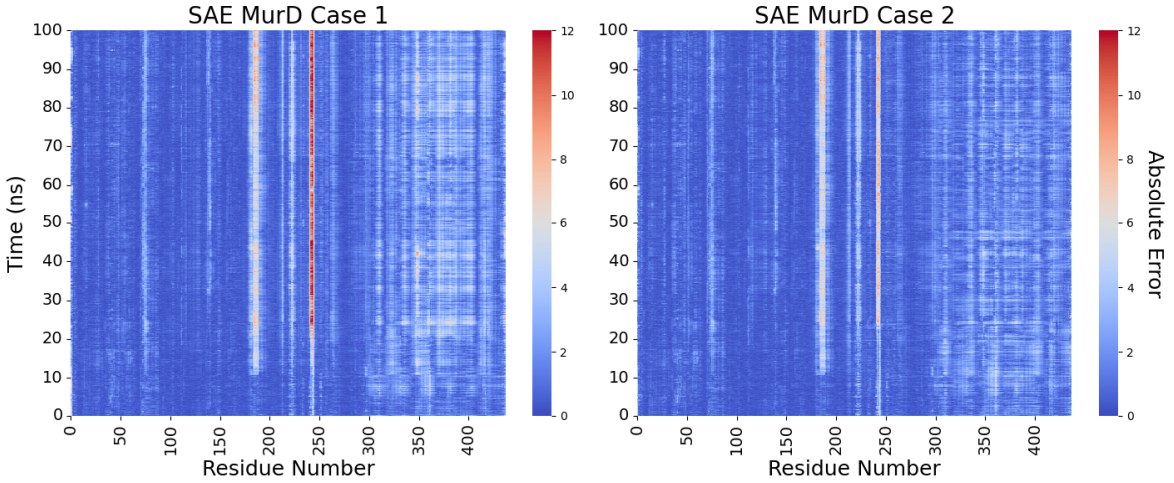

Figure 3.6. Comparison of absolute errors of MurD case one and case two results with a heat map.

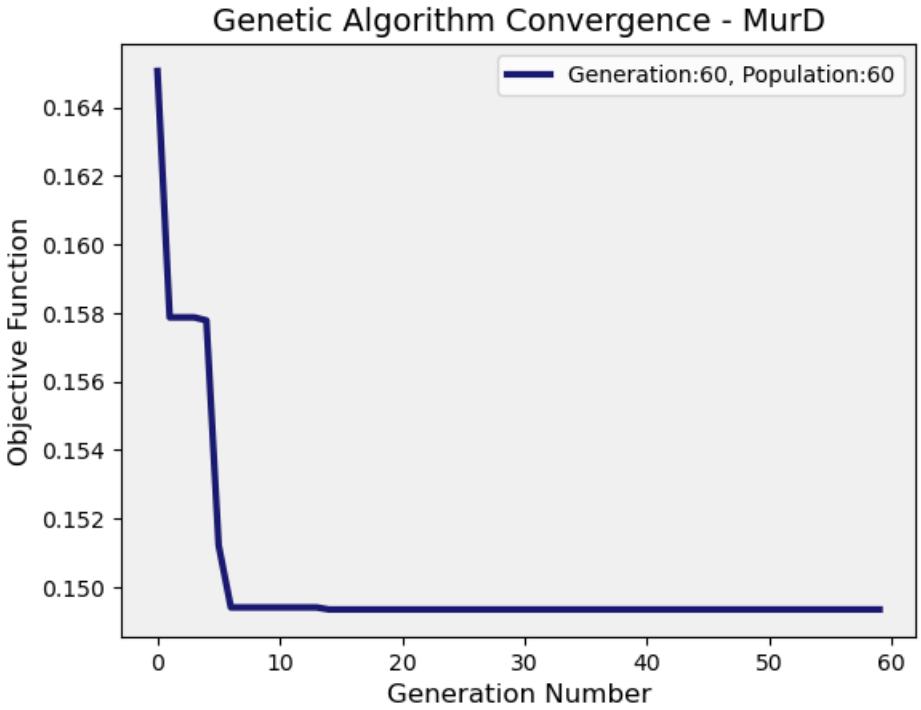

Figure 3.7. Convergence of genetic algorithm for MurD case two for 60 generations with 60 populations.

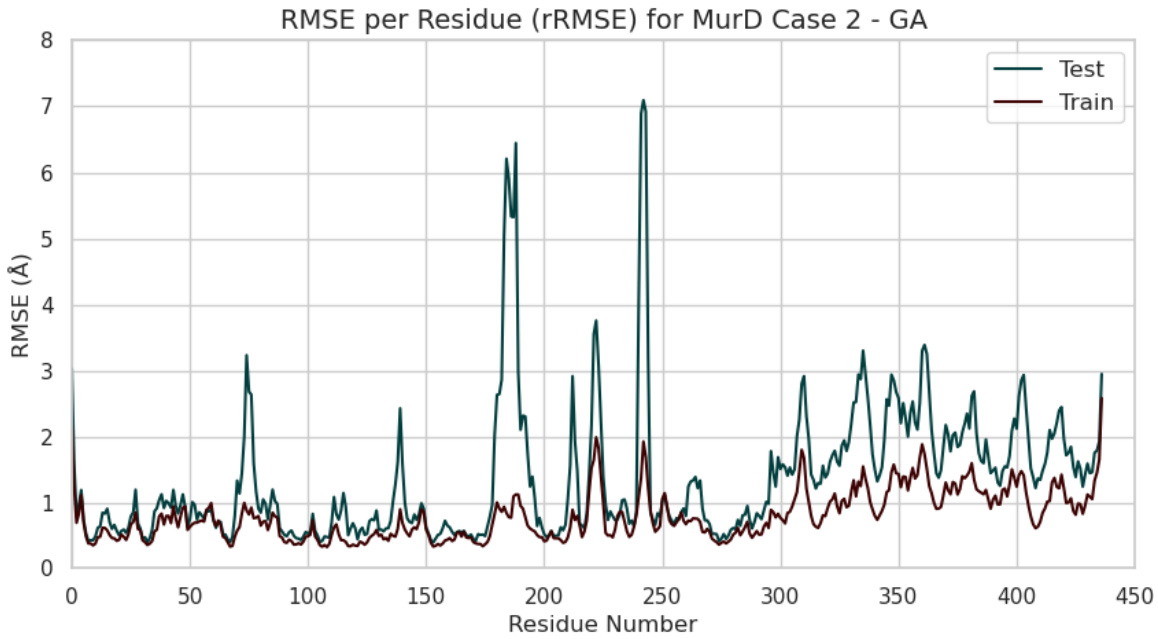

Figure 3.8. RMSE per residue (Å) for the MurD case two showing the prediction results for test (green line) and train (red line) data. The best hyperparameter set is found by the genetic algorithm with 60 generations and 60 populations.

Table 3.5. Best hyperparameters for MurD obtained from genetic algorithm for 60 generations with 60 populations.

Hyperparameters	**Case 2**
Activation Function	SELU
Regularizer (λ)	1e-4
Learning Rate (η)	1e-4
Momentum (β_1)	0.75
Epoch	200
Batch Size	16
Hidden Layer Number	1
Dimension of Hidden Layer 1	150
Dimension of Hidden Layers 2, 3 and 4	0
Latent Layer Dimension	10

Given the improved performance of semi-supervised autoencoder with the second case of MurD, another genetic algorithm run was employed for 60 generations with 60 populations for case two. The remaining genetic algorithm parameters stayed the same. The convergence of the genetic algorithm is shown in Figure 3.7. It is interesting that the algorithm converged before 10 generations. The resulting best hyperparameters are given in Table 3.5. These best hyperparameters are used in the downstream semi-supervised autoencoder training. The obtained model is then used to predict the test data. The rRMSE graphs for the train and test data is shown in Figure 3.8. The average rRMSE values for train and test data are 0.80 Å and 1.37 Å, respectively. The resulting RMSE table is given in Table 3.6. When the test rRMSE plots of case two from the previous genetic algorithm run with 30 generations with 30 population is compared to this result with 60 generations and 60 populations, there is no significant difference between different runs for the genetic algorithm (Figure 3.9).

Table 3.6. RMSE values for MurD case two with 60 generations and 60 populations.

RMSE Type	**Case 2**	**Unit**
$rRMSE_{Avg}$ (Test)	1.37	Å
$rRMSE_{Avg}$ (Train)	0.80	Å
Coordinate $RMSE_{Avg}$ (Test)	0.98	Å
Coordinate $RMSE_{Avg}$ (Train)	0.51	Å
Opening Angle $RMSE_{Avg}$ from Latent Layer (Test)	1.16	Degree
Opening Angle $RMSE_{Avg}$ from EHL1 (Test)	1.78	Degree
Dihedral Angle $RMSE_{Avg}$ (Test)	0.23	Radian
Dihedral Angle $RMSE_{Avg}$ (Train)	0.16	Radian
cRMSE (GA Objective Function)	0.15	-

The genetic algorithm with more generation and population converged faster than the generation 30 run but overshoot the global optimum and trapped in the local minimum. This problem can be achieved by running genetic algorithm with more generations and populations. However, the computational time required for genetic

algorithm increases as the generation and population number increases. Therefore, trying other hyperparameter optimization methods would address this problem.

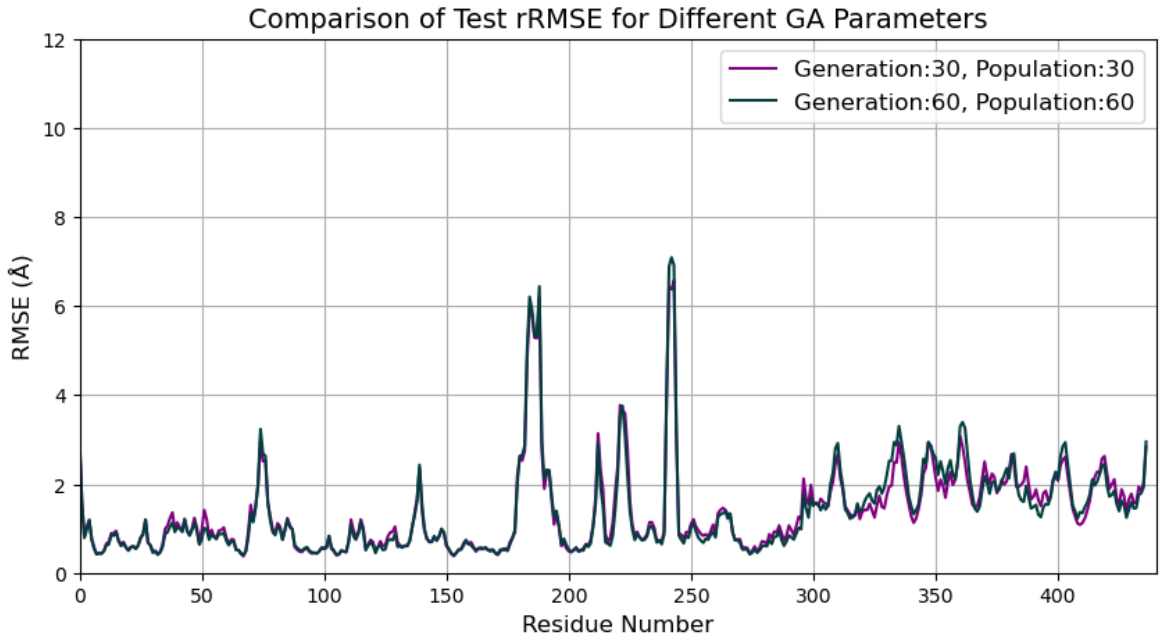

Figure 3.9. Comparison of rRMSE values for different runs of genetic algorithm for case two of MurD. Generation 30 and population 30 is shown with pink line while the generation 60 and population 60 is shown with green line.

Since the purpose of semi-supervised autoencoder is to extract the collective variables from the latent layer vectors, the resulting two-dimensional latent layer space from test data prediction for latent layer vector combinations ξ_1, ξ_2 and ξ_3 are scatter plotted and colored based on opening angle, RMSD, gyration and SASA in Figure 3.10, Figure 3.11 and Figure 3.12 for case one (generation: 30, population: 30), case two (generation: 30, population: 30) and case two (generation: 60, population: 60), respectively for test data (transition from closed-apo to open state). The last column of the figures shows the coloring variable values with respect to time in test data. Here, 100 ns simulation is the same as 1000 frame for the test data. For case one, latent layer vector combinations (ξ_1, ξ_2) and (ξ_2, ξ_3) resulted in two distinct clusters. When colored by the opening angle, these clusters are colored with the minimum (dark blue) and maximum (brown) of opening angle which are representing the closed-apo state ($\sim$70 degrees) and open state ($\sim$90 degrees) (Figure 3.10 (a)). This indicates that

these latent vector combinations can be considered as good collective variables that are showing two distinct states of the MurD. This alignment of distinct separation by latent space vectors and the opening angle indicates that the use of opening angle in the autoencoder with a semi-supervised manner successfully results in the extraction of good collective variables that can distinguish between the two separate states.

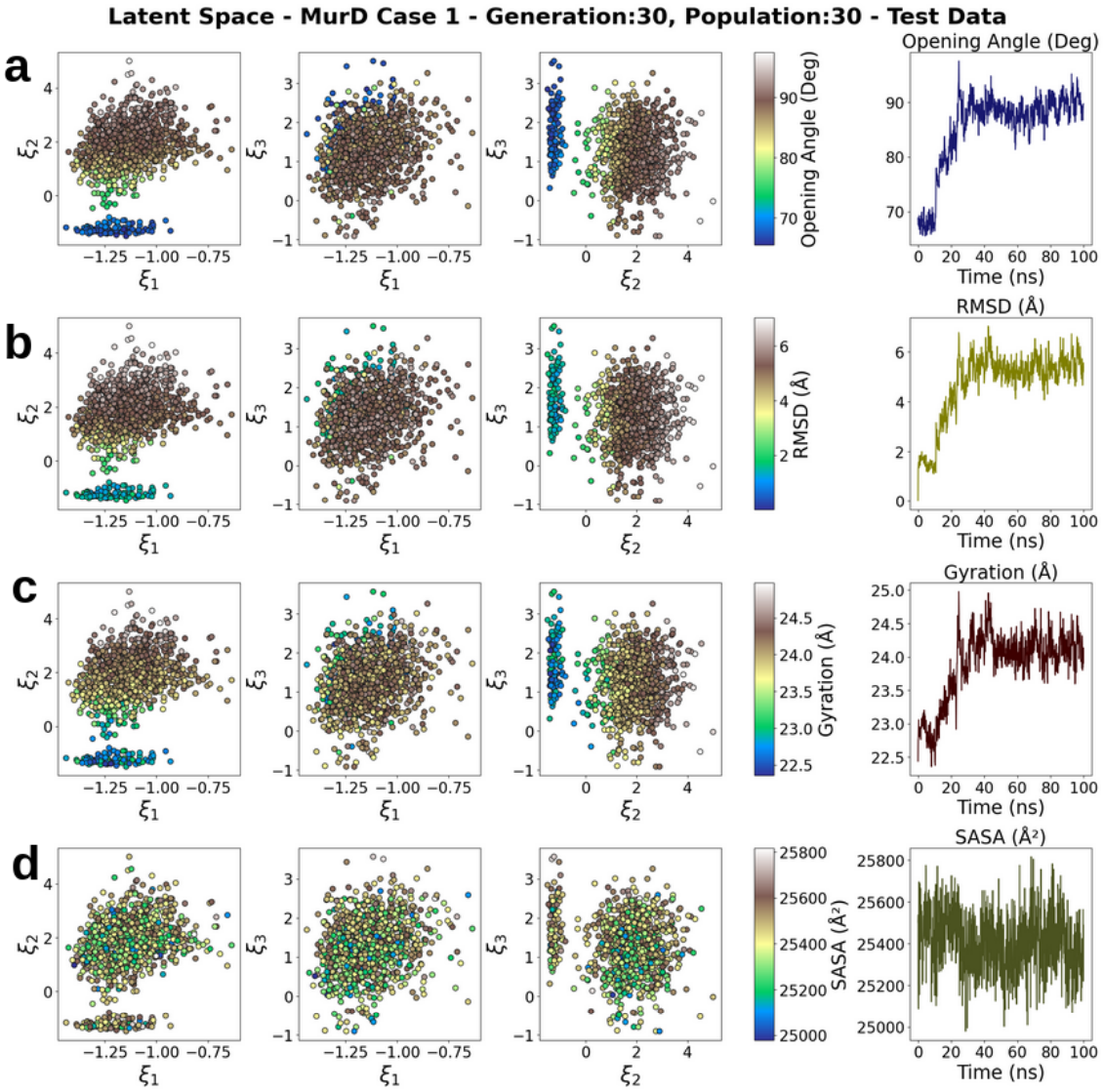

Figure 3.10. Two-dimensional latent space (first three columns) with combinations of latent vectors ξ_1, ξ_2 and ξ_3 colored by opening angle (a), RMSD (b), gyration (c), and SASA (d). Using best found hyperparameters from GA with 30 generations and 30 populations for MurD test data predictions of case one. Last column shows the coloring variable w.r.t. time for test data.

The RMSD of the test data (w.r.t. its starting structure) in the last column of Figure 3.10 (b), shows distinct states for closed-apo (∼1.5 Å) and open (∼5.5 Å). The coloring of the latent space is also compatible with these values. This indicates that the latent space vector combinations, (ξ_1, ξ_2) and (ξ_2, ξ_3), can be considered as a good CV candidate representing the same distinct states similar to RMSD. The same logic applies to the gyration in Figure 3.10 (c), as it is also showing two distinct states and compatible with the latent space clusters.

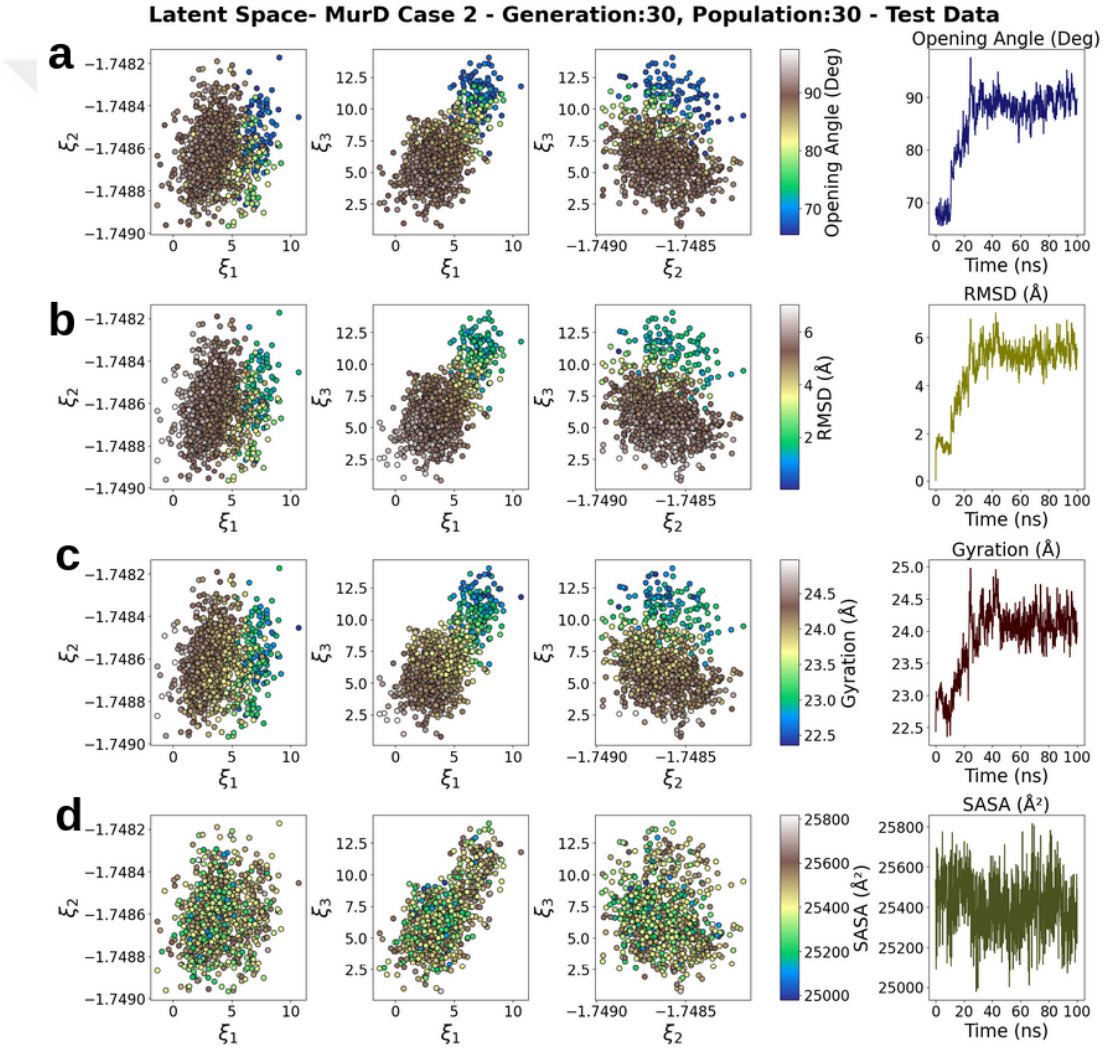

Figure 3.11. Two-dimensional latent space (first three columns) with combinations of latent vectors ξ_1, ξ_2 and ξ_3 colored by opening angle (a), RMSD (b), gyration (c), and SASA (d). Using best found hyperparameters from GA with 30 generations and 30 populations for MurD test data predictions of case two. Last column shows the coloring variable w.r.t. time for test data.

In Figure 3.10 (d), the last column shows the SASA plot of test data with respect to time. As it can be seen from this plot, SASA does not clearly represent distinct states. The coloring the scatter plots in the latent space by SASA does not show clear color separation. This suggests that the latent space vector combinations for (ξ_1, ξ_2) and (ξ_2, ξ_3) do not result in false positives.

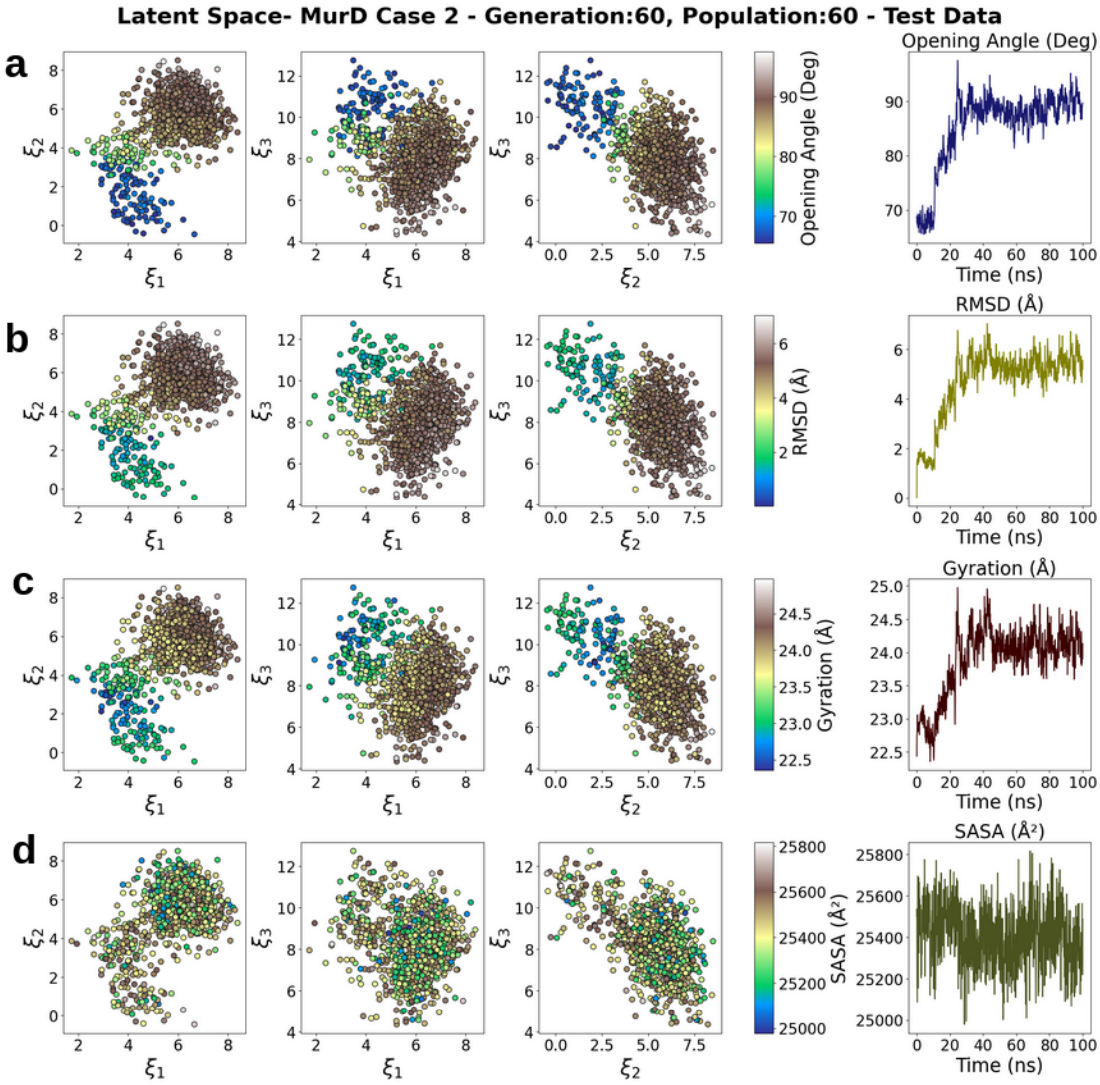

Figure 3.12. Two-dimensional latent space (first three columns) with combinations of latent vectors ξ_1, ξ_2 and ξ_3 colored by opening angle (a), RMSD (b), gyration (c), and SASA (d). Using best found hyperparameters from GA with 60 generations and 60 populations for MurD test data predictions of case two. Last column shows the coloring variable w.r.t. time for test data.

For case two (generation: 30, population: 30), the resulting latent space does not show distinct clusters when compared to the case one. There is still some groupings though in latent vector combinations of (ξ_1, ξ_3) and (ξ_2, ξ_3) and they are compatible with the coloring clusters by the opening angle, RMSD and the gyration. Again, distinct coloring the clusters does not happen in SASA.

The results for case two (generation: 60, population: 60), in all of the latent space vector combinations, show distinct clusters. And the clusters are colored according to their corresponding coloring data with the expected behaviour for opening angle, RMSD, gyration, and SASA. It is interesting that the semi-supervised autoencoder reconstruction error results, rRMSD, is better in case two than case one, while the latent space collective variable represents the intrinsic states of the simulation better in case one.

3.2. Hyperparameter Optimization with Genetic Algorithm: AdK

The genetic algorithm for two cases of adenosine kinase was run for 30 generations with 30 population similar to MurD. The same genetic algorithm parameters are used as in MurD (Table 3.1). Again the objective function (cRMSE) calculation for each hyperparameter combination is computed as the mean of 10 repetitions. The convergence plot of the genetic algorithm with AdK is given in Figure 3.13.

It can be seen that convergence for case two happens after 10 generations. For the case one, after 10 generations it remains the same but after 25 generations it starts to decrease again. That implies higher generation numbers are needed to achieve better minimization of objective function. In contrast to MurD, the early stopping criteria is not applied to genetic algorithm with AdK. This is because the size of the dataset is smaller than the MurD and there is no need for stopping early because genetic algorithm is faster in AdK. Therefore, early stopping criteria is opt out for this protein.

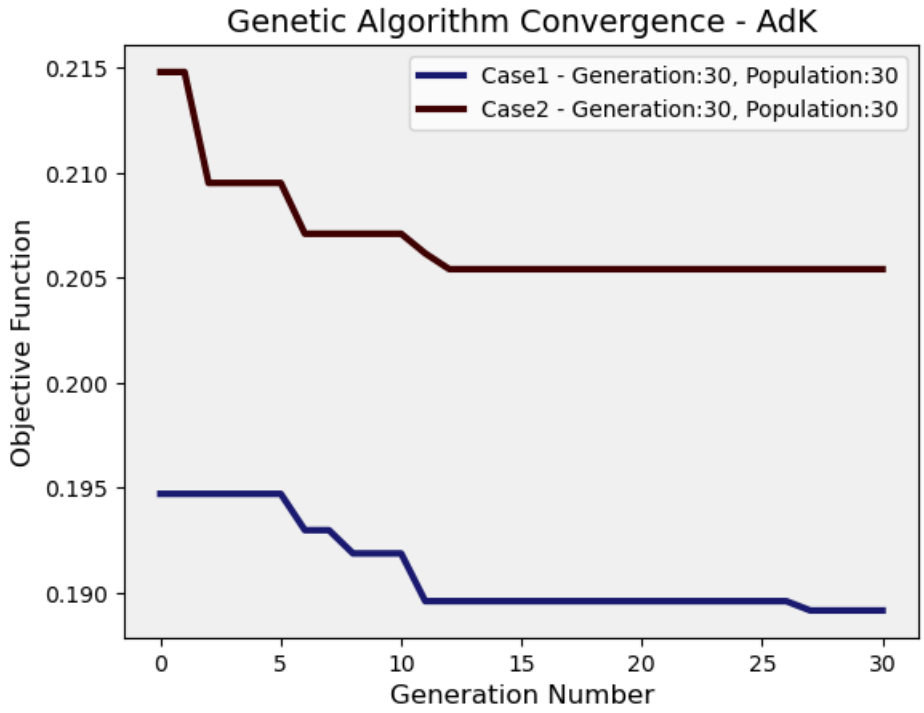

Figure 3.13. Convergence of genetic algorithm for AdK case one and two for 30 generations with 30 populations.

The hyperparameter space explored for the AdK is listed in Table 3.7. The same hyperparameters are tested for case one and case two. The resulting best hyperparameters are given in Table 3.8. The best hyperparameters for both of the cases resulted in same autoencoder architecture with same number of nodes and layers while different parameters for the loss function is observed.

Using the best hyperparameters from running genetic algorithm with 30 generations and 30 populations, the semi-supervised autoencoder is trained with the training data prepared for AdK. Then, the fitted model is used to predict the test and train data for AdK. The resulting rRMSE plots are given in Figure 3.14 for test and train data. The average rRMSE values are 0.56 Å and 0.35 Å for test and train data predictions, respectively. Training data performs better than the test data as anticipated. Overall, the prediction for this protein yielded better results than the MurD.

Table 3.7. Hyperparameters explored for AdK

Hyperparameters	**Case 1 and Case 2**
Activation Function	Sigmoid ReLU SELU
Regularizer (λ)	1e-3, 1e-4, 1e-5
Learning Rate (η)	1e-3, 1e-4, 1e-5
Momentum (β_1)	0.7, 0.75, 0.80 0.85, 0.90, 0.95
Epoch	100, 200, 300 400, 500
Batch Size	16, 32, 64 256, 512
Hidden Layer Number	1, 2, 3
Dimension of Hidden Layer 1	50, 75, 90, 110, 125 150, 175, 205, 250, 325
Dimension of Hidden Layer 2	10, 17, 30, 45, 55 65, 80, 100, 125, 150
Dimension of Hidden Layer 3	7, 9, 10, 12, 15 20, 25, 35, 40, 60
Latent Layer Dimension	2, 3, 4, 5, 6

It should be noted that the dataset for AdK is very small compared to MurD. The test data shows some peaks around residue 50 which is part of the NMP domain. In the semi-supervised autoencoder, the NMP-CORE and LID-CORE angles are used in a manner similar to opening angle in the MurD where the model semi-supervised with these angle values. The RMSF of AdK is given in Figure 3.15. The peaks in the rRMSE plot is in the similar residues as in the NMP domain. Although there is also fluctuations in the LID domain, the rRMSE values are low there.

Table 3.8. Best hyperparameters for AdK obtained from GA.

Hyperparameters	**Case 1**	**Case 2**
Activation Function	SELU	SELU
Regularizer (λ)	1e-3	1e-4
Learning Rate (η)	1e-3	1e-3
Momentum (β_1)	0.7	0.85
Epoch	500	300
Batch Size	32	16
Hidden Layer Number	1	1
Dimension of Hidden Layer 1	110	110
Dimension of Hidden Layer 2	0	0
Dimension of Hidden Layer 3	0	0
Latent Layer Dimension	6	6

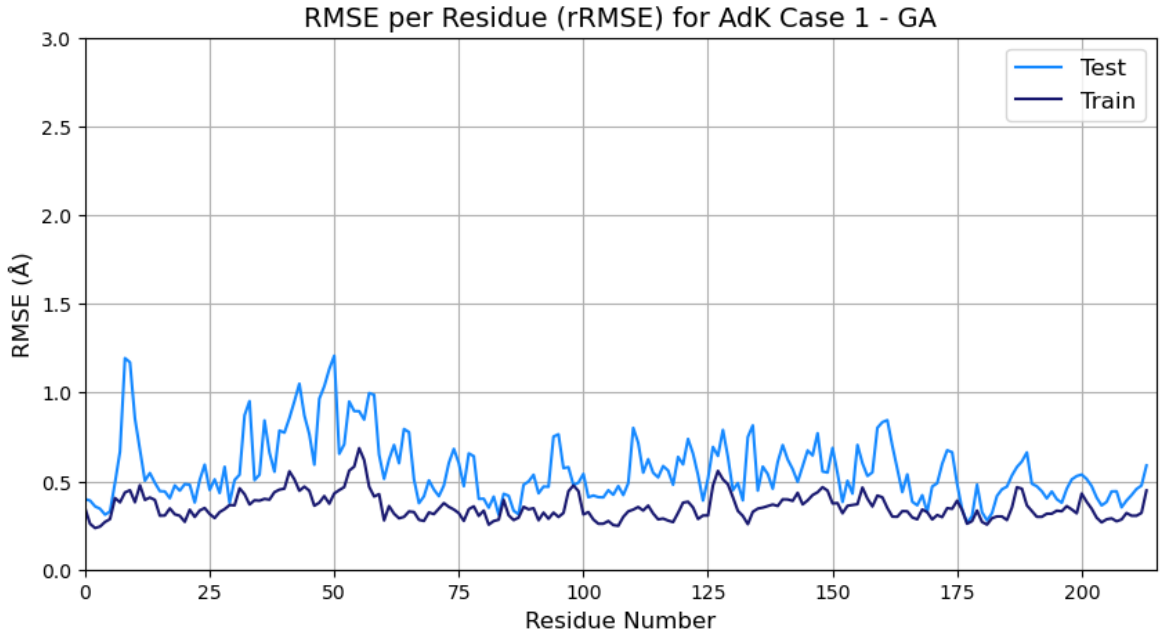

Figure 3.14. RMSE per residue for the AdK case one showing the prediction results for test (light blue line) and train (dark blue line) data. The best hyperparameter set is found by the genetic algorithm with 30 generations and 30 populations.

To follow a similar pipeline with the MurD, in case two of AdK, the sine and cosine pairs of dihedral angles for the residues are also given to the semi-supervised autoencoder along with the Cα coordinates. The hyperparameter space explored for the case two of AdK is the same as the case one (Table 3.7). The best hyperparameter combination for the case two is given in Table 3.8. Using these hyperparameters, the semi-supervised trained and then the test and train data are predicted.

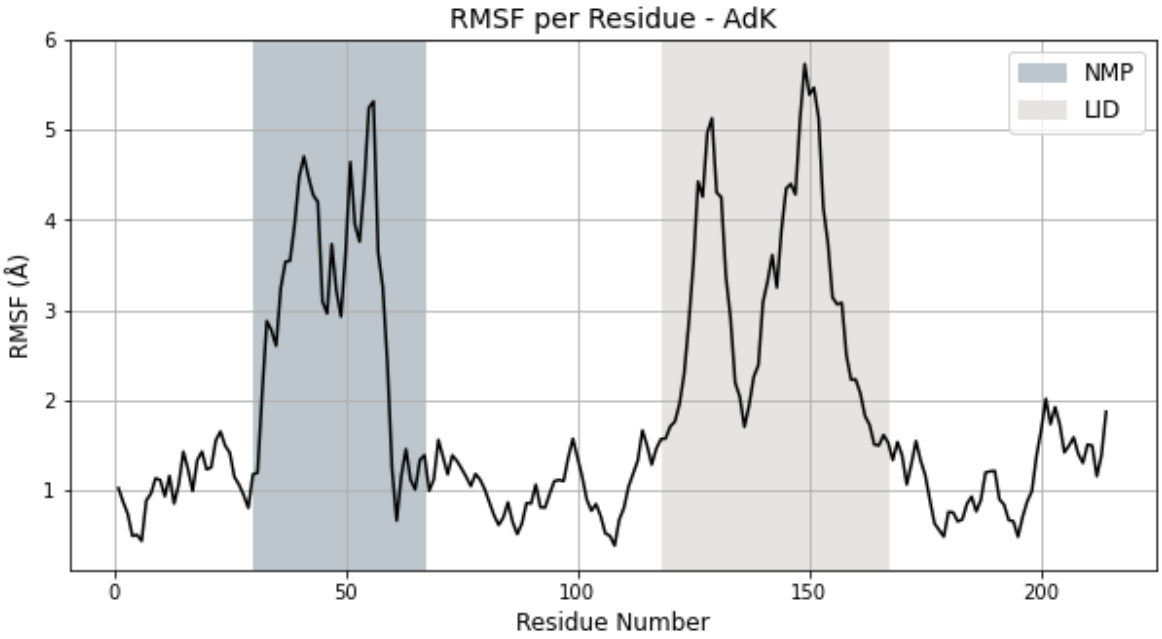

Figure 3.15. RMSF per residue for the AdK.

The rRMSE plot for the test and train data is given in Figure 3.16. The average rRMSE values are 0.63 Å and 0.35 Å for test and train data, respectively. In the rRMSE plot, the test data prediction is poor in the NMP region and in the residues around 10. However, overall prediction performance is good as in case one. A comparison of rRMSE values for case one and case two are shown in Figure 3.17. Although they are almost the same, case one is performing better in predicting the NMP region. The other RMSE values for reconstructed coordinates, LID-CORE and NMP-CORE angles, and others are compared for the two cases in Table 3.9.

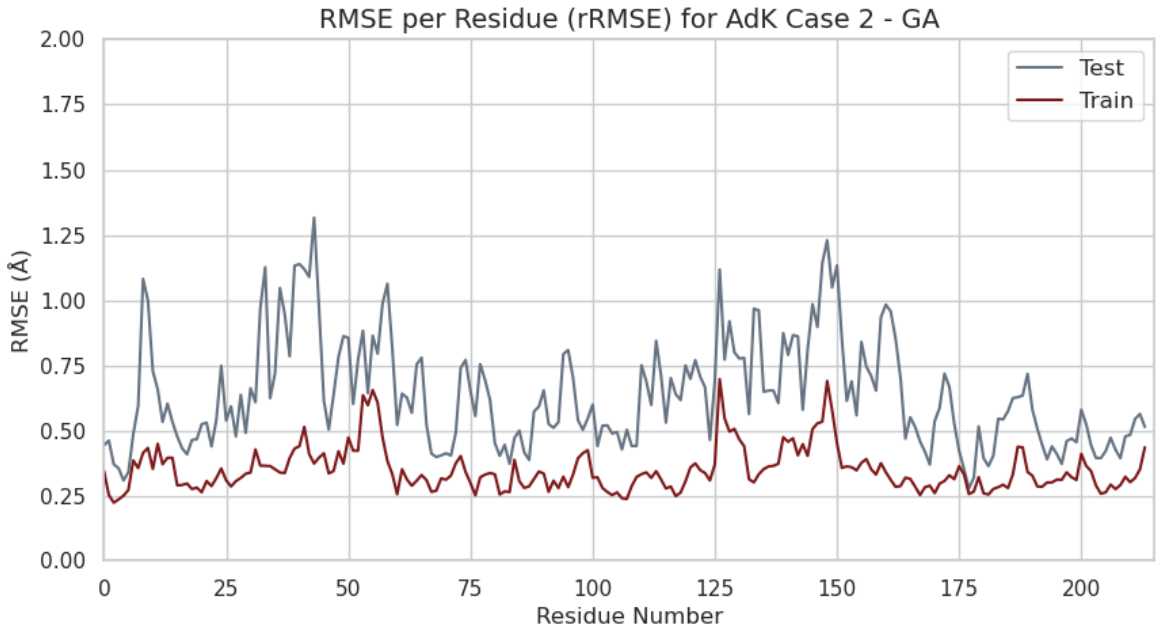

Figure 3.16. RMSE per residue for the AdK case two showing the prediction results for test (gray line) and train (red line) data. The best hyperparameter set is found by the genetic algorithm with 30 generations and 30 populations.

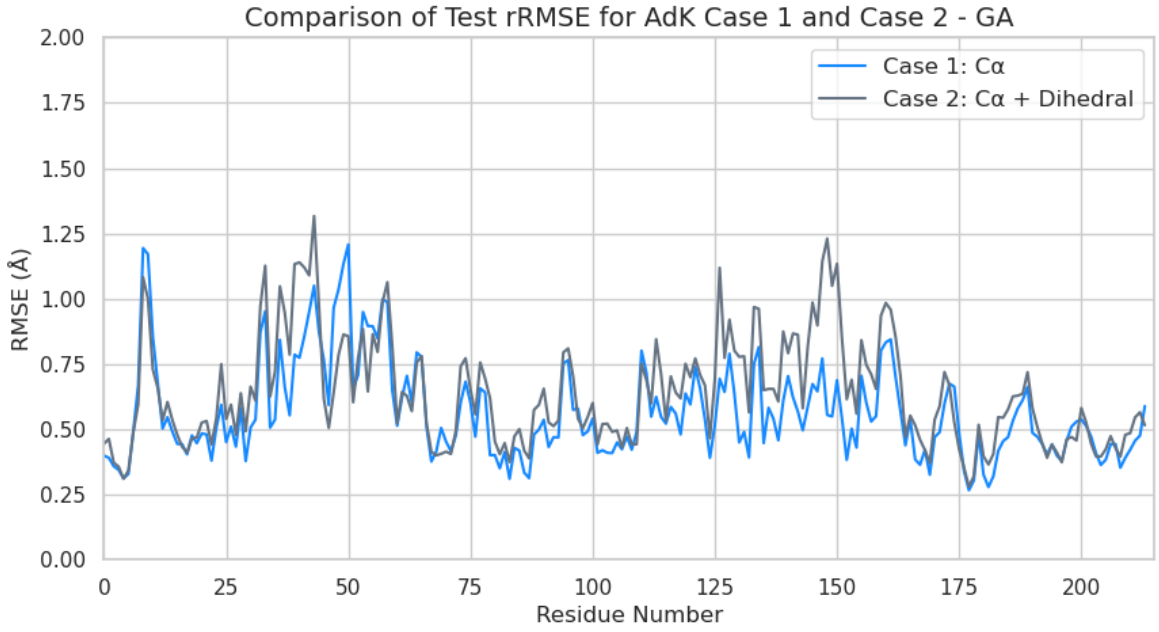

Figure 3.17. Test data prediction rRMSE comparison for case one (light blue line) and case two (gray line) of AdK using the hyperparameters obtained from running genetic algorithm for 30 generations with 30 populations.

Although both of the cases performed good, to understand the effect of more generations and populations, another genetic algorithm was run for case two for 60 generations with 60 populations. The convergence plot of the genetic algorithm is given in Figure 3.18. It can be seen that the genetic algorithm converged after 40 generations. Compared to the previous genetic algorithm runs with AdK, it took three days to complete for 60 generations and 60 populations. The hyperparameter space explored is the same as the previous runs (Table 3.7). The resulting best hyperparameter combination is listed in Table 3.10.

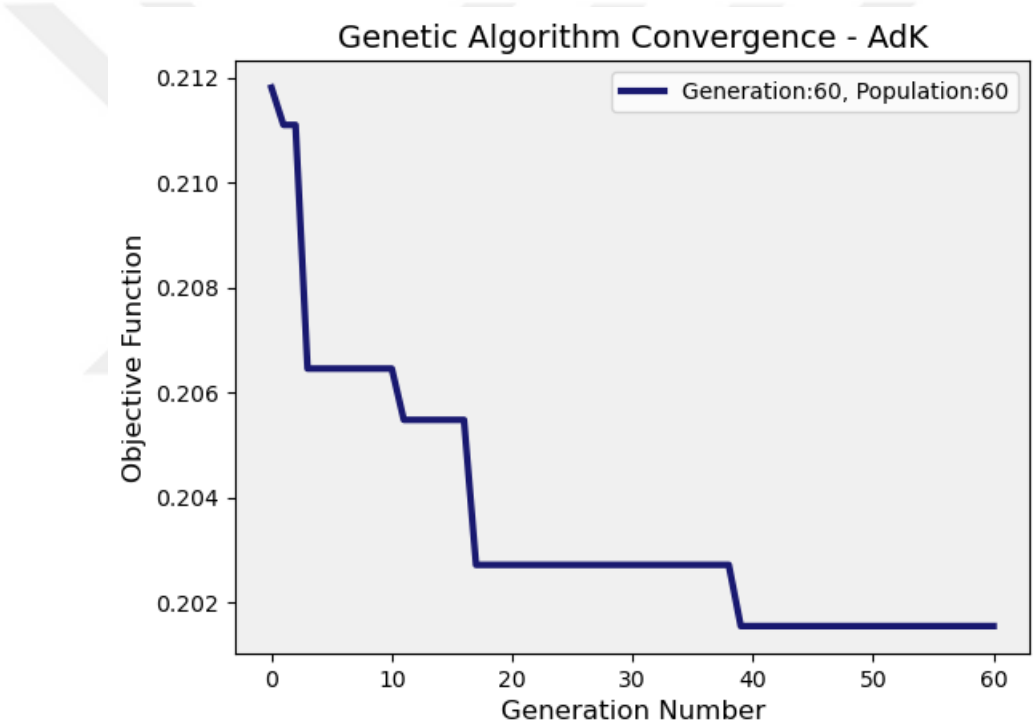

Figure 3.18. Convergence of genetic algorithm for AdK case two for 60 generations with 60 populations.

After obtaining the best hyperparameters from genetic algorithm with 60 generations and 60 populations, the next step is to run the semi-supervised autoencoder for AdK case two. The resulting rRMSE plot for test and train is given in Figure 3.19. The average rRMSE values are 0.7 Å and 0.43 Å for test and train data, respectively. The poor performance on NMP domain increased and it also got worse than the previous runs for predicting the LID domain around residue 150. A comparison with the different runs of case two is given in Figure 3.20. The other RMSE calculations for

the LID-CORE and NMP-CORE angle predictions from latent layer and first encoder hidden layer is listed in Table 3.11. Interestingly, the performance of the bet hyper-parameters found by longer genetic algorithm is worse than running the algorithm with lower generations and populations. The reason for that the genetic algorithm is got trapped in a local minima point. To overcome this, increasing the generation and population number would help. However, running 60 generations takes three days increasing the generation number will lead to longer computational calculations and therefore it is not feasible.

Table 3.9. RMSE values for AdK case one and two with 30 generations and 30 populations.

RMSE Type	**Case 1**	**Case 2**	**Unit**
$rRMSE_{Avg}$ (Test)	0.56	0.63	Å
$rRMSE_{Avg}$ (Train)	0.35	0.35	Å
Coordinate $RMSE_{Avg}$ (Test)	0.34	0.38	Å
Coordinate $RMSE_{Avg}$ (Train)	0.20	0.20	Å
LID-CORE Angle $RMSE_{Avg}$ from Latent Layer (Test)	1.18	1.84	Degree
NMP-CORE Angle $RMSE_{Avg}$ from Latent Layer (Test)	0.8	1.91	Degree
LID-CORE Angle $RMSE_{Avg}$ from EHL1 (Test)	1.05	1.43	Degree
NMP-CORE Angle $RMSE_{Avg}$ from EHL1 (Test)	0.98	1.64	Degree
cRMSE (GA Objective Function)	0.19	0.21	-

Since the aim of the semi-supervised autoencoders is to derive collective variables from the latent layer vectors, the resulting latent space from case one, case two (generation: 30, population: 30) and case two (generation: 60 and population: 60) are presented in Figure 3.21, Figure 3.22, and Figure 3.23, respectively. Each two-dimensional combinations of latent space vectors (ξ_1 and ξ_2, ξ_1 and ξ_3, ξ_2 and ξ_3) are scatter plotted and colored based on the corresponding LID-CORE angle, NMP-CORE angle, RMSD, gyration and SASA. In the last column, these coloring variables are shown with respect to time (20 ps). Due to the small amount of test data sample, the latent space vectors are consists of 20 points which makes it difficult to show significantly distinct clusters.

Table 3.10. Best hyperparameters for AdK obtained from genetic algorithm for 60 generations with 60 populations.

Hyperparameters	**Case 2**
Activation Function	SELU
Regularizer (λ)	1e-3
Learning Rate (η)	1e-3
Momentum (β_1)	0.7
Epoch	400
Batch Size	32
Hidden Layer Number	1
Dimension of Hidden Layer 1	150
Dimension of Hidden Layer 2	0
Dimension of Hidden Layer 3	0
Latent Layer Dimension	6

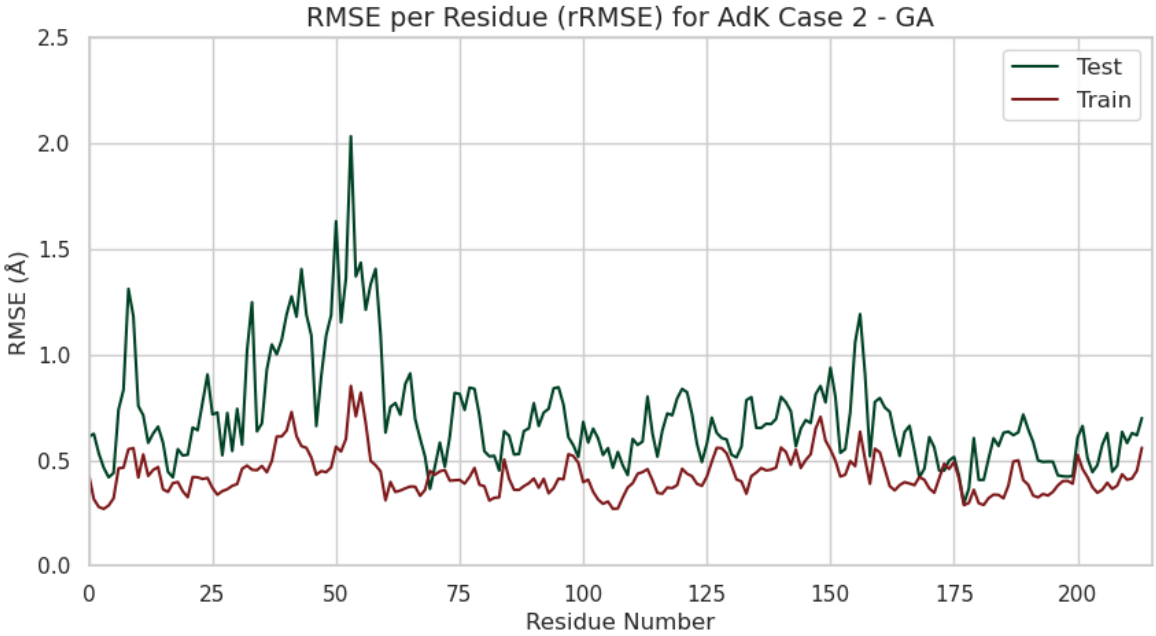

Figure 3.19. RMSE per residue for the AdK case two showing the prediction results for test (green line) and train (red line) data. The best hyperparameter set is found by the genetic algorithm with 60 generations and 60 populations.

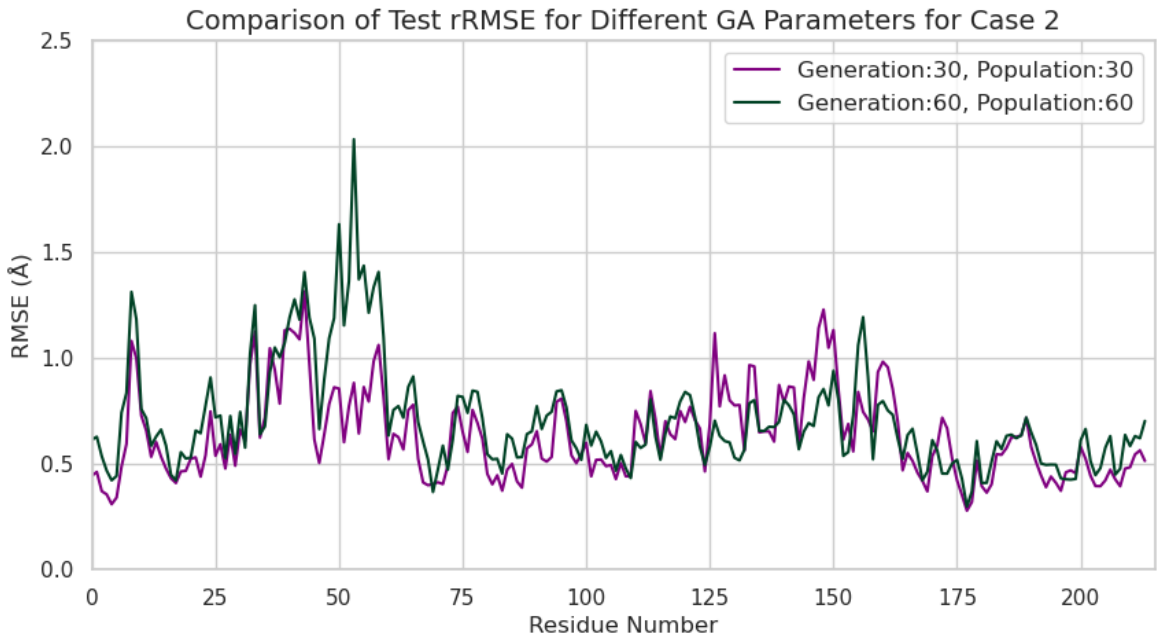

Figure 3.20. Comparison of rRMSE values for different runs of genetic algorithm for case two of AdK. Generation 30 and population 30 is shown with pink line while the generation 60 and population 60 is shown with green line

Table 3.11. RMSE values for AdK case two with 60 generations and 60 populations.

RMSE Type	**Case 2**	**Unit**
$rRMSE_{Avg}$ (Test)	0.70	Å
$rRMSE_{Avg}$ (Train)	0.43	Å
Coordinate $RMSE_{Avg}$ (Test)	0.43	Å
Coordinate $RMSE_{Avg}$ (Train)	0.25	Å
LID-CORE Angle $RMSE_{Avg}$ from Latent Layer (Test)	1.07	Degree
NMP-CORE Angle $RMSE_{Avg}$ from Latent Layer (Test)	0.87	Degree
LID-CORE Angle $RMSE_{Avg}$ from EHL1 (Test)	1.01	Degree
NMP-CORE Angle $RMSE_{Avg}$ from EHL1 (Test)	3.07	Degree
cRMSE (GA Objective Function)	0.20	-

In case one, the latent space vector combination, ξ_1 and ξ_2, showing a diagonal distribution of points where the coloring transitions are visible for NMP-CORE angle, RMSD and gyration. There is also some coloring transition in LID-CORE angle

coloring, but not as strong as others. This is expected since the LID-CORE angle with respect to time is not as dramatically changing as the other three. SASA is not differentiating between the two states as well. And the coloring of the transition is not as clearly visible in SASA. For the other vector combinations, it can be also seen that there are some sort of groupings. Especially in the ξ_1 and ξ_3 combination. Coloring transition behaviour is consistent in this vector combination as in the previous one.

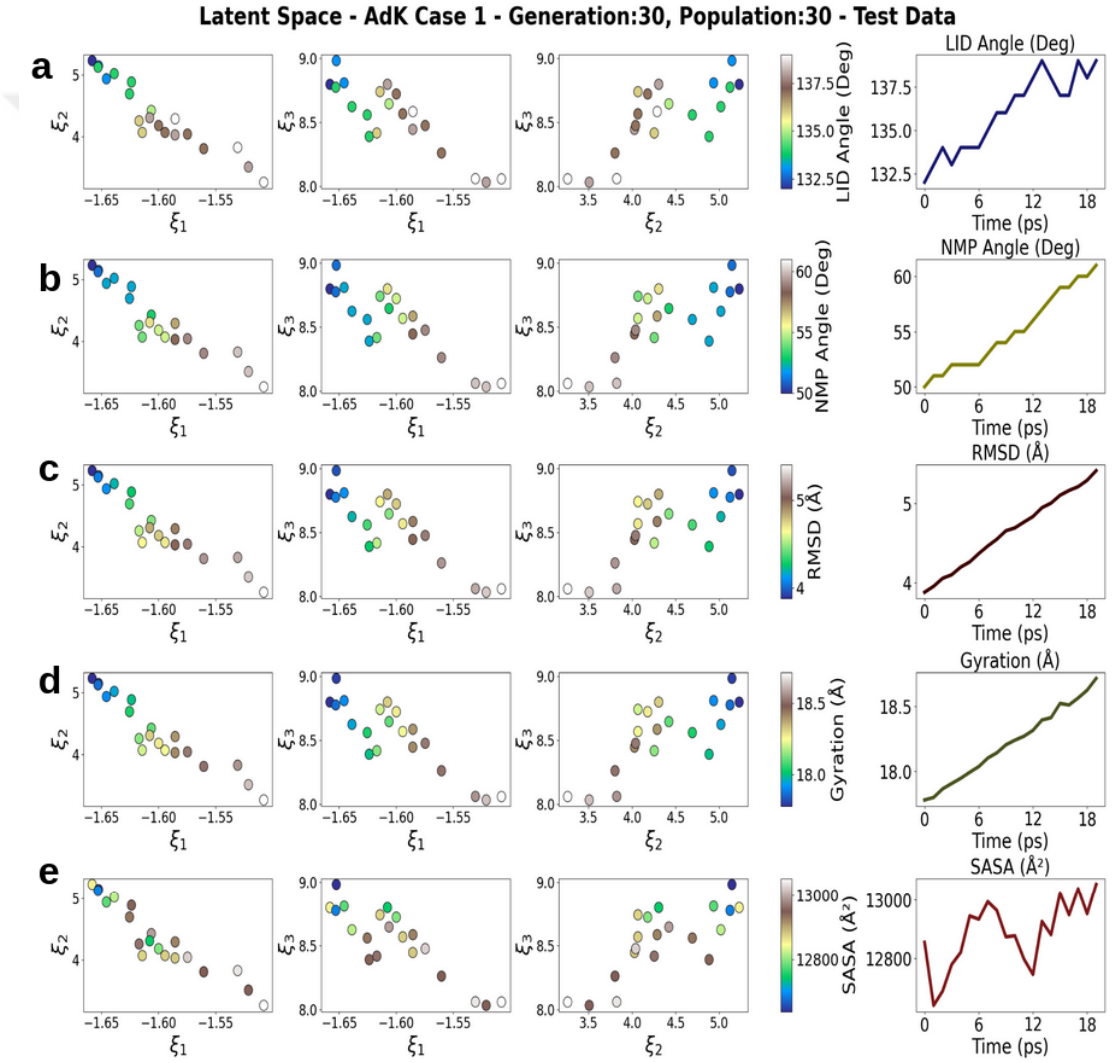

Figure 3.21. Two-dimensional latent space (first three columns) with combinations of latent vectors 1, 2 and 3 colored by LID-CORE angle (a), NMP-CORE angle (b), RMSD (c), gyration (d), and SASA (e). Using best hyperparameters from genetic algorithm with 30 generations and 30 populations for AdK test data predictions of case one. Last column shows the coloring variable w.r.t. time for test data.

In case two from genetic algorithm with 30 generations and 30 populations, the vector combinations does not hold clearly separated points (Figure 3.22). Only in the last combination, ξ_2 and ξ_3, the transition behaviour is observed. And the coloring by NMP-CORE, RMSD and gyration is very similar and they are followed by the LID-CORE angle in terms of performance quality. This can imply that since the case one RMSE values are better than case two, the collective variable quality is also better in case one.

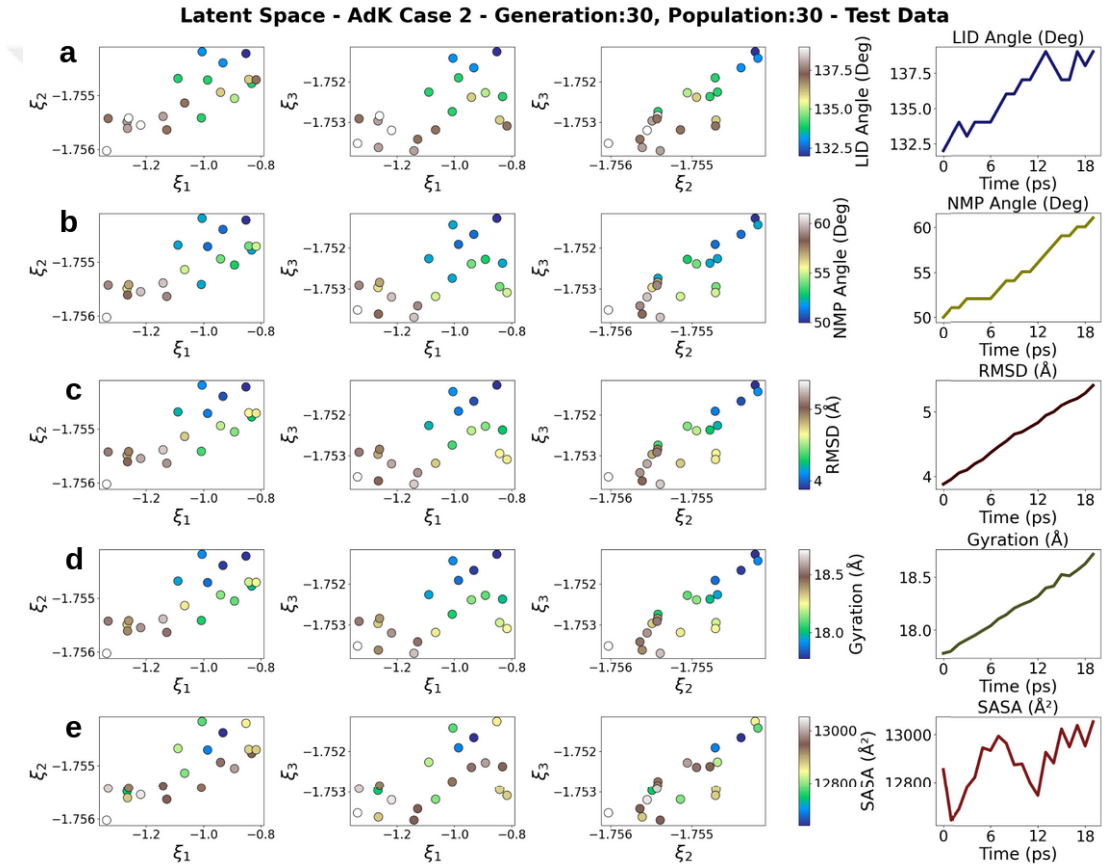

Figure 3.22. Two-dimensional latent space (first three columns) with combinations of latent vectors 1, 2 and 3 colored by LID-CORE angle (A), NMP-CORE angle (B), RMSD (C), gyration (D), and SASA (E). Using best hyperparameters from genetic algorithm with 30 generations and 30 populations for AdK test data predictions of case two. Last column shows the coloring variable w.r.t. time for test data.

Although the case two with 60 generations and 60 populations has the worst RMSE, the diagonal transition behaviour of the points are better in this last one

(Figure 3.23). In all of the two-dimensional latent space combinations, the diagonal distribution of points observed with two distinct state representations at the ends of the diagonal. Again, the NMP-CORE angle, RMSD and gyration coloring are clearly seen on the points followed by LID-CORE angle coloring. And the SASA is not performing good as expected. Even if the size of the data is too small, it is clearly representing a transition from closed state to the open state. It is curious to observe that in both MurD and AdK, where the worst performing cases in terms of rRMSE (case one in MurD and case two in AdK), are the best performing ones in terms of CVs for representing the distinct states.

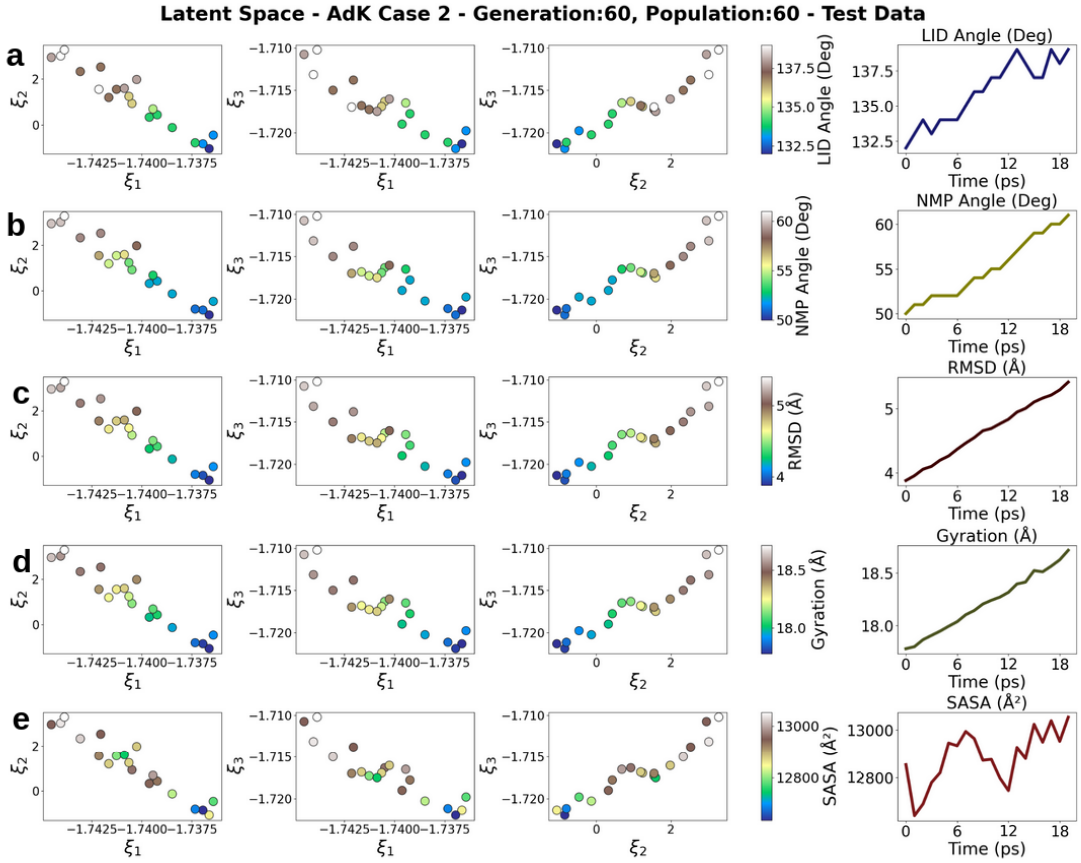

Figure 3.23. Two-dimensional latent space (first three columns) with combinations of latent vectors 1, 2 and 3 colored by LID-CORE angle (A), NMP-CORE angle (B), RMSD (C), gyration (D), and SASA (E). Using best hyperparameters from genetic algorithm with 60 generations and 60 populations for AdK test data predictions of case two. Last column shows the coloring variable w.r.t. time for test data.

3.3. Semi-supervised Autoencoder vs Vanilla Autoencoder

The proposed semi-supervised autoencoder model is compared with a baseline model, vanilla autoencoder, where there is only encoder and decoder for MD trajectory coordinates. Semi-supervised autoencoder differs from vanilla autoencoder with extra output nodes coming from latent layer and encoder hidden layers by providing extra information of opening angle. The performance two models are compared for their RMSE values per residue (rRMSE) and time (fRMSE) in Figure 3.24 using MurD test data prediction for case one and for case two in Figure 3.25.

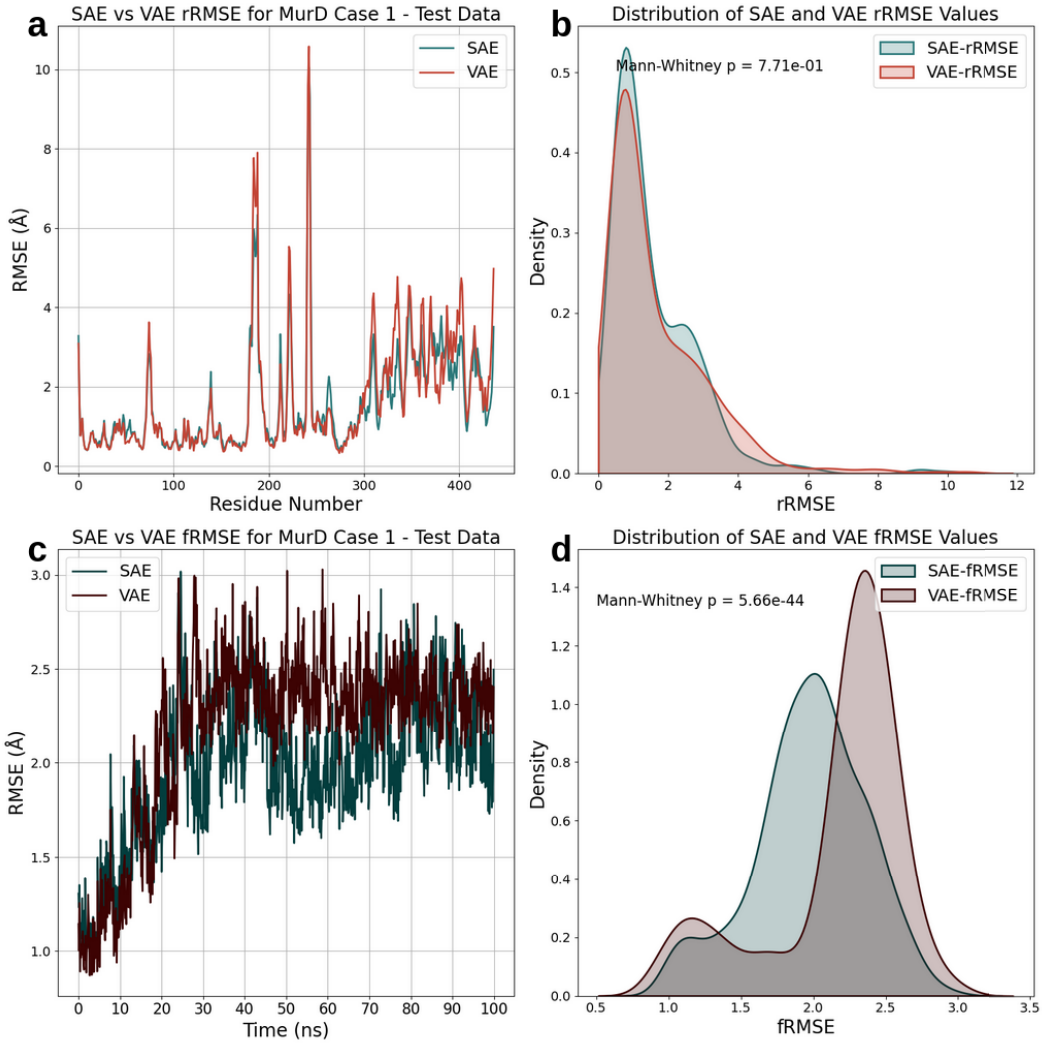

Figure 3.24. Comparison of SAE and VAE model performances with MurD case one. (a) rRMSE for test data, (b) distribution of rRMSE values, (c) fRMSE across simulation time, and (d) distribution of fRMSE values.

For case one in Figure 3.24, the rRMSE profiles demonstrate comparable accuracies between the SAE and VAE models, as indicated by a non-significant p-value of 0.771. Conversely, a marked distinction is observed in the fRMSE values over the 100 ns trajectory, with the SAE model demonstrating a statistically significant improvement in performance ($p < 0.001$).

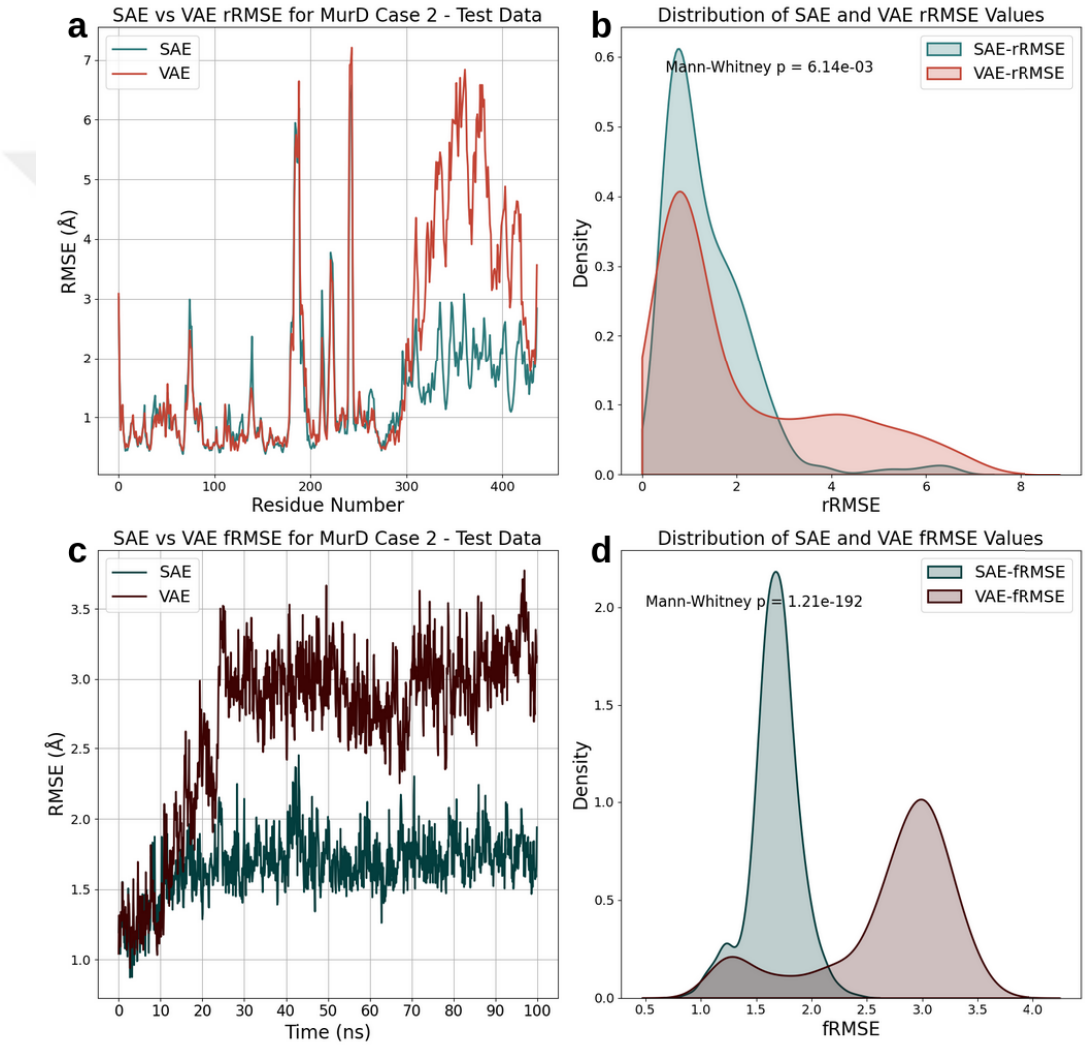

Figure 3.25. Comparison of SAE and VAE model performances with MurD case two. (a) rRMSE for test data, (b) rRMSE distribution, (c) fRMSE across simulation time, and (d) distribution of fRMSE values.

For case two, presented in Figure 3.25, the SAE model exhibits marginally lower rRMSE values than the VAE model, suggesting a subtle yet statistically significant edge ($p = 0.00614$). This advantage is magnified in the fRMSE distribution over the

simulation duration, with the SAE model showing a highly significant lead in accuracy ($p < 0.001$), as confirmed by the Mann-Whitney U test. This suggests that the addition of dihedral angle information is significantly improves the performance of the semi-supervised autoencoder. The heatmap visualizations in Figure 3.26 present a detailed comparison of the absolute prediction errors across different MurD cases for both SAE and VAE models. In panels (a) and (b), representing case one, the SAE model displays a largely uniform error distribution over time, with spikes in errors at residues 242 and 185. The VAE model, on the other hand, exhibits a similar error pattern but with a slightly higher frequency of error peaks, as denoted by the red streaks across the timeline.

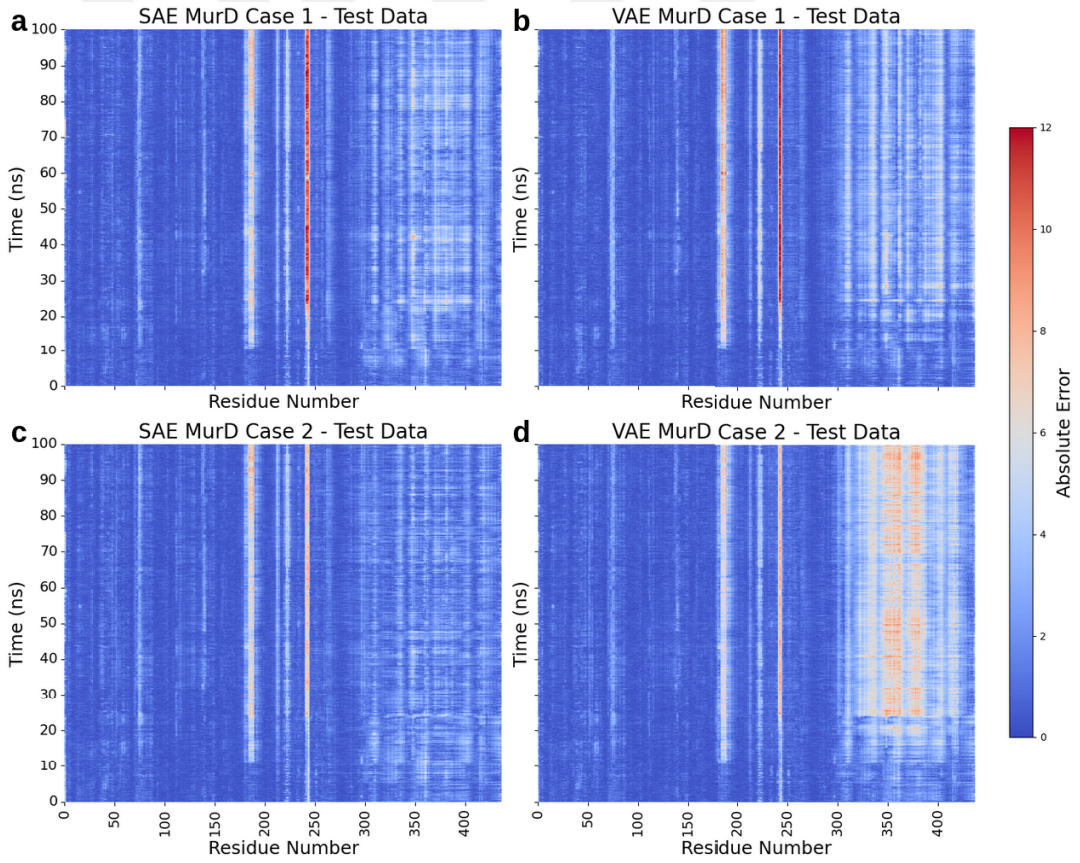

Figure 3.26. Heatmap visualization of absolute errors in test data predictions for SAE and VAE models across different MurD cases. (a) SAE model errors for case one over time, (b) VAE model errors for case one, (c) SAE model errors for case two, and (d) VAE model errors for case two. The color scale indicates the magnitude of error, with red representing higher error.

For case two, as shown in panels (c) and (d), the error landscape changes markedly. The SAE model maintains a relatively consistent error profile with fewer instances of high error, while the VAE model reveals a broader spread of significant errors, particularly in specific regions that appear as vertical red lines. This implies that for case two, the SAE model predicts certain residues with greater accuracy than the VAE model. The color intensity in these heatmaps, ranging from blue to red, quantitatively underscores the regions where the models struggle to accurately predict. Red zones are indicative of areas with the highest errors and are particularly revealing of the temporal and spatial inaccuracies in the models' predictions. From the temporal information, it can be seen that the errors increase in the C-terminal region and the peaks after around 10 ns, which corresponds to the ending of the closed state. This suggests that VAE poorly predicting the transition and open structures of the MurD because of the flexibility in these regions. However, SAE is still performing better in C-terminal even though it is flexible.

3.3.1. Secondary Structure Analysis

In addition to evaluating the overall performance of the semi-supervised autoencoder and vanilla autoencoder in generating conformations, the precision of these models in predicting secondary structure changes over time is explored through secondary structure analysis. Specifically, the accuracy of secondary structure predictions is assessed for MurD enzyme in the APO state (test data), focusing on case two.

Figure 3.27 offers a side-by-side representation of the actual versus predicted secondary structures as rendered by the SAE model. The left panel validates the actual secondary structures, depicting the expected persistence and fluctuations of α-helices (blue), β-sheets (red), and loops (gray) across the timeline. The right panel of Figure 3.27 showcases the SAE's secondary structure predictions. Notably, the SAE model maintains a strong agreement with the actual data, particularly in the consistent portrayal of α-helices and β-sheets, though slight deviations can be observed in the prediction of loops over time. The percentage of overall prediction of secondary

structures by SAE case two is 86.86 % (Table 3.12). This similarity in the patterns between the actual and SAE-predicted data underscores the model's capability to capture and replicate the secondary structure dynamics.

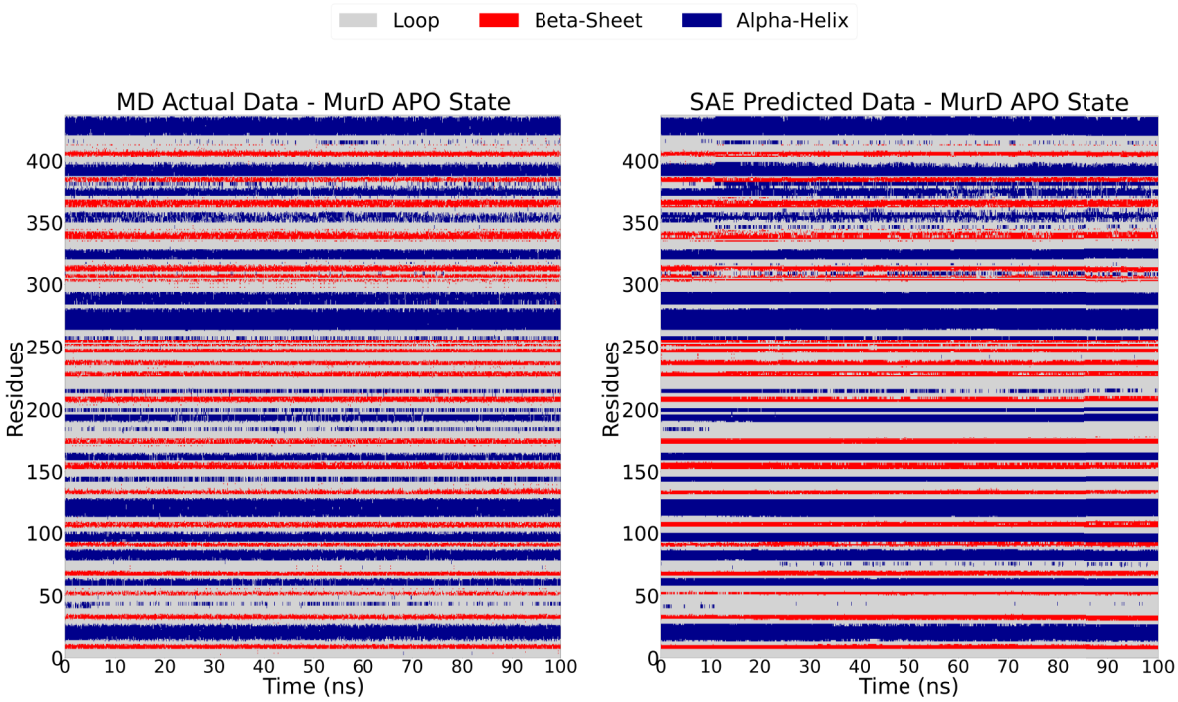

Figure 3.27. Comparison of secondary structure evolution in MurD APO state for case two over 100 ns between (a) actual molecular dynamics data and (b) SAE model predictions, highlighting α-helices (blue), β-sheets (red), and loops (gray).

Extending the analysis to the VAE model, Figure 3.28 shows the comparison against the actual MD data. In left, the secondary structure as determined by direct MD simulations observed, which provides a reference for evaluating the VAE model's performance. Right panel reveals the predictions made by the VAE model. There is a notable agreement with the actual MD data, particularly with the α-helices and β-sheets. Interestingly, the VAE model seems to offer a marginally better prediction of loop structures compared to the SAE model, suggesting an enhanced capability in capturing the flexible aspects of the protein. The percentage of overall prediction performance of VAE in case two is 85.18 % (Table 3.12).

The overall matched structures and unmatched structures for the actual and predicted data can be seen better in Figure 3.29, where black dots represent the unmatched secondary structure while gray regions shows the prediction's agreement with the actual structures. The two models are compared in terms of the errors in predicting the secondary structure. According to these, the SAE model performs slightly better in overall prediction, especially in the C-terminal region of the MurD enzyme. A snapshot of structures at 50 ns is also shown below their corresponding plots. The predicted and actual structures are aligned and colored as cyan, red and blue for actual, SAE-predicted and VAE-predicted, respectively. From these alignments, some of the deviations in the predictions can be observed. For example, the SAE-predicted structure mistakenly shows β-sheet instead of loop (shown by arrows on the figure) in the C-terminal region of the protein. Likewise, the VAE-predicted structure shows the same mistake as well with the addition of more mismatch such as predicting loops as α-helix.

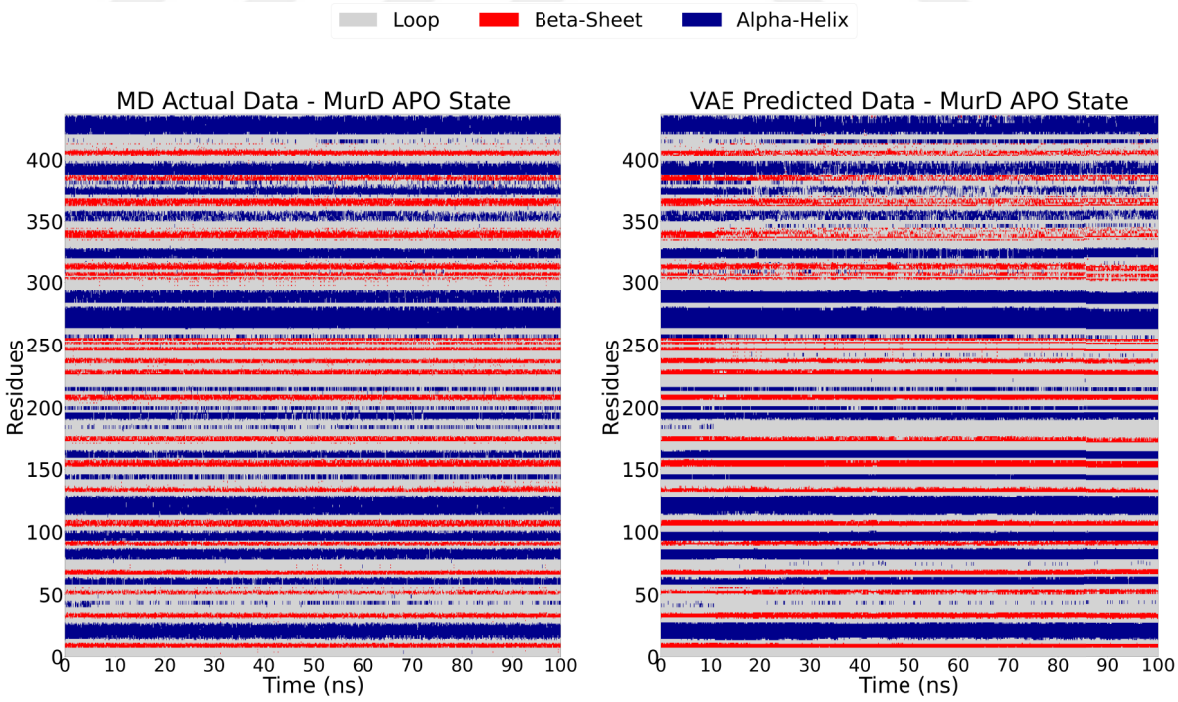

Figure 3.28. Comparison of secondary structure evolution in MurD APO state for case two over 100 ns between (a) actual molecular dynamics data and (b) VAE model predictions, highlighting α-helices (blue), β-sheets (red), and loops (gray).

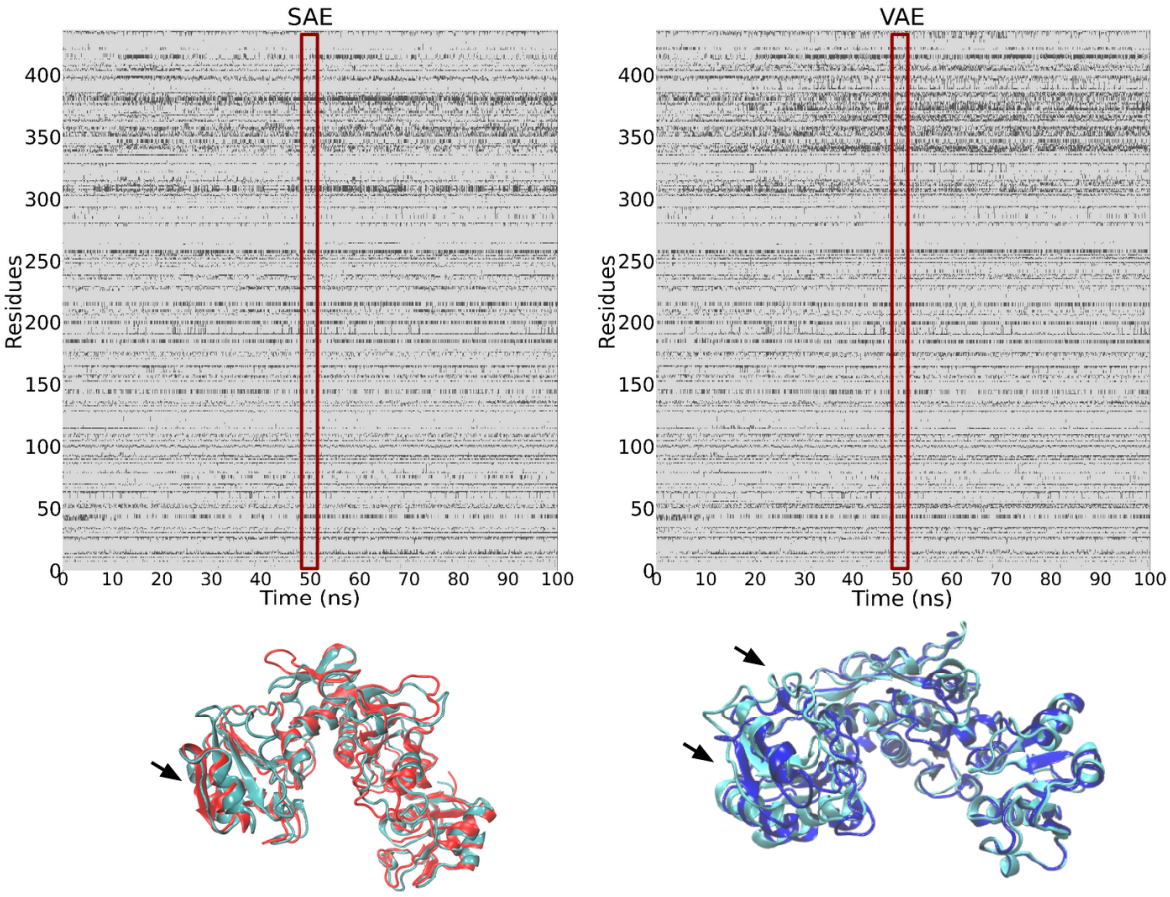

Figure 3.29. Secondary structure prediction errors for MurD case two with black dots indicating errors and gray areas showing correct predictions by SAE (left) and VAE (right) models. The lower images compare the actual protein conformation (cyan) against SAE (red) and VAE (dark blue) predictions at 50 ns, highlighting regions with significant prediction deviations.

Table 3.12. Percentage of matching secondary structure elements between test structures and their reconstructed equivalents.

Autoencoder Architecture Type	**Case 1**	**Case 2**
Semi-supervised Autoencoder	85.67 %	86.86 %
Vanilla Autoencoder	86.39 %	85.18 %

This visual comparison is crucial in assessing the models' abilities to accurately capture the complex folding and unfolding patterns that occur in protein secondary structures over time. The figures reveals that both models have a tendency to mispre-

dict certain regions, which could correspond to the protein's more dynamic segments, such as loop regions or turns that are inherently difficult to predict due to their variable nature. It is also observed that both of the models poorly predict the transition regions where two of the secondary structures are next to each other. For example transitioning from α-helix to loop presents challenge for the autoencoders to predict which one is which. Nevertheless, the overall shape and folding patterns are remarkably well captured by both models, with the SAE model showing a slightly better congruence with the actual structure.

In all of the analysis so far in MurD APO state, it can be clearly observed that predicting the C-terminal region of MurD enzyme is challenging. This might be due to the fact that the opening and closing movements of the protein is directed by its C-terminal domain. The flexibility in this domain makes it challenging for the semi-supervised autoencoder to predict. The APO state starts with the closed structure and after around 10 ns it starts to transition to the open state and after 30 ns it is completely open. The C-terminal prediction performance of the autoencoders in general also decreases after 10 ns as it can be seen in the heat maps in Figure 3.26. However, when SAE is compared to VAE, prediction of C-terminal regions are significantly better. The C-terminal is more stable in close state than in the transition and open state. Therefore, it is hard to correctly predict the unstable regions of the MurD with the vanilla autoencoder. It can also be seen from the secondary structure analysis that predicting the loop regions also problematic since they are also highly flexible regions. It can be concluded that the overall prediction performance of the semi-supervised autoencoder is significantly better than the vanilla autoencoder.

4. CONCLUSION

Understanding the conformational dynamics of proteins is the first step to uncover their mechanism of action. Through molecular dynamics simulations, realistic representation of these conformational changes of the proteins at the atomic level can be investigated. Given the large number of atoms present in a protein, computing atomic level interactions for simulation trajectory requires excessive computational time and resources. To analyse and understand the large amount of MD trajectory data of proteins, there are several methods to reduce the dimensionality of the data and obtain the reduced representation of the protein movement. These reduced representations are also called collective variables. These variables should be chosen carefully to represent the large scaled movements in a protein. Geometric collective variables such as RMSD, radius of gyration, and opening angles are used extensively. However, not all the conformational changes of proteins can be represented by these collective variables. In addition, choosing the correct collective variable is intuition driven specific to each protein under investigation. More automated approaches are developed for the discovery of collective variables. Data-driven approaches combined with the advanced machine learning techniques are becoming popular in recent years. Neural network based deep learning methods are offering interesting approaches to collective variable discovery. Autoencoders, a neural network type, are being extensively used as a dimensionality reduction technique. And in the last decade, major amount of research is going on for collective variable extraction from MD simulation data using autoencoders.

Autoencoders consists of an encoder, decoder, and a latent layer. The MD simulation data of proteins are given to the encoder layers by gradually reducing the dimensions up to latent layer, i.e. bottleneck layer. The vectors from the latent layer space can be used as collective variables to understand their function and movement. Obtaining the best collective variables that can represent the dynamics of the proteins dependent on the fine-tuned autoencoder architecture and features given to the au-

toencoder. For the purpose of obtaining correct collective variables in an automatic manner through employing autoencoders, a semi-supervised autoencoder architecture is used in this thesis. The performance of semi-supervised autoencoder is tested on two different proteins MurD and adenosine kinase, using their MD simulation data. These proteins are selected because of the nature of their dynamics involves distinct conformational states that can be represented by the geometric collective variables for testing the validity of the proposed semi-supervised autoencoder model. So that the latent space extracted collective variables from semi-supervised autoencoder can be compared to these geometric collective variables to measure the performance. To test the effect of using different features for these proteins, two different cases applied for both of the proteins. In case one, only the coordinates of Cα atoms used from the MD trajectory to prepare test and train datasets. In case two, along with the coordinates, the sine and cosine pairs of dihedral angles for each residue is also given to the semi-supervised autoencoder as a feature. To achieve better performance of the autoencoder, hyperparameters of semi-supervised autoencoder are optimized using genetic algorithm. The optimized hyperparameters include activation function, regularizer, learning rate, momentum, epoch number, batch size, hidden layer number, and dimensions of each hidden layer.

The optimization by genetic algorithm yielded good results for MurD and AdK. The best collective variables for MurD are obtained through using the Cα coordinates of simulation trajectory with running genetic algorithm for 30 generations with 30 populations. The addition of dihedral angle information as a feature did not yield good CVs when compared to the former case. However, increasing the generation and population number of the genetic algorithm improved the collective variable extraction in the second case. Interestingly, when the reconstruction errors of encoded and decoded outputs are compared, the addition of dihedral angles in case two of MurD yielded best results. For AdK, the same behaviour observed. The collective variables from Cα coordinates yielded better results than addition of dihedral angles. The performance of the second case increased when the generation and population number increased in genetic algorithm run. These findings suggests that, increasing the generation and

population number of the genetic algorithm improves the latent space representation hence collective variables. However, as the number of generations and populations increase the computational time also increases taking days and even weeks to complete even for relatively small number of 60 generations. Considering these, it can be noted that optimized autoencoder architecture can lead to better reduced representation. However, choosing a fast and accurate optimization method is crucial.

For testing the performance of the proposed semi-supervised autoencoder, it was compared with a baseline autoencoder, vanilla autoencoder for MurD enzyme data. It was observed that semi-supervised autoencoder architecture performed significantly better than the vanilla autoencoder in case when the Cα atoms and dihedral angles are used in the input data. This shows that the addition of extra output layers coming from the latent layer and encoder hidden layer decreases the reconstruction error in protein MD simulation trajectory predictions. It was also observed that in all of the analysis of MurD enzyme, the C-terminal region is challenging to predict in the transition and open states. C-terminal region is highly flexible due to its role in directing the opening and closing of the enzyme during its catalytic process.

This study adds to our knowledge of hyperparameter optimization of autoencoders as well as the collective variable extraction methods. The collective variables obtained from the latent layer of the semi-supervised autoencoder can be used in the downstream enhanced sampling algorithms. This research has also thrown up many questions in need of further investigation. Different optimization methods other than genetic algorithm can be tested for semi-supervised autoencoder. Moreover, to prevent the trapping in the local optimum in genetic algorithm, more generations and more populations can be tried if the computational time issue is solved. For the reconstruction error, the highly flexible regions presents a challenge for both the semi-supervised and the baseline autoencoder. This issue can be solved by improving the semi-supervised autoencoder architecture by addition of extra information other than the opening angle or training the autoencoder with more simulation trajectory data of the transition and open states. Further research is needed to provide solutions to these problems.

REFERENCES

1. Borhani, D. W. and D. E. Shaw, "The Future of Molecular Dynamics Simulations in Drug Discovery", *Journal of Computer-Aided Molecular Design*, Vol. 26, No. 1, pp. 15–26, 2012.

2. Salsbury Jr, F. R., "Molecular Dynamics Simulations of Protein Dynamics and Their Relevance to Drug Discovery", *Current Opinion in Pharmacology*, Vol. 10, No. 6, pp. 738–744, 2010.

3. McCammon, J. A., B. R. Gelin and M. Karplus, "Dynamics of Folded Proteins", *Nature*, Vol. 267, No. 5612, pp. 585–590, 1977.

4. Karplus, M. and J. A. McCammon, "Molecular Dynamics Simulations of Biomolecules", *Nature Structural Biology*, Vol. 9, No. 9, pp. 646–652, 2002.

5. Bernardi, R., D. Gomes, R. Gobato, C. Taft, A. Ota and P. Pascutti, "Molecular Dynamics Study of Biomembrane/Local Anesthetics Interactions", *Molecular Physics*, Vol. 107, No. 14, pp. 1437–1443, 2009.

6. Liu, Y., J. Strümpfer, P. L. Freddolino, M. Gruebele and K. Schulten, "Structural Characterization of λ-Repressor Folding From All-Atom Molecular Dynamics Simulations", *The Journal of Physical Chemistry Letters*, Vol. 3, No. 9, pp. 1117–1123, 2012.

7. Zhao, G., J. R. Perilla, E. L. Yufenyuy, X. Meng, B. Chen, J. Ning, J. Ahn, A. M. Gronenborn, K. Schulten, C. Aiken *et al.*, "Mature HIV-1 Capsid Structure by Cryo-Electron Microscopy and All-Atom Molecular Dynamics", *Nature*, Vol. 497, No. 7451, pp. 643–646, 2013.

8. Maveyraud, L. and L. Mourey, "Protein X-Ray Crystallography and Drug Discovery", *Molecules*, Vol. 25, No. 5, p. 1030, 2020.

9. Alderson, T. R. and L. E. Kay, "Nuclear Magnetic Resonance Spectroscopy Captures the Essential Role of Dynamics in Regulating Biomolecular Function", *Cell*, Vol. 184, No. 3, pp. 577–595, 2021.

10. Ponder, J. W. and D. A. Case, "Force Fields for Protein Simulations", *Advances in Protein Chemistry*, Vol. 66, pp. 27–85, 2003.

11. Hollingsworth, S. A. and R. O. Dror, "Molecular Dynamics Simulation for All", *Neuron*, Vol. 99, No. 6, pp. 1129–1143, 2018.

12. Henzler-Wildman, K. A., M. Lei, V. Thai, S. J. Kerns, M. Karplus and D. Kern, "A Hierarchy of Timescales in Protein Dynamics is Linked to Enzyme Catalysis", *Nature*, Vol. 450, No. 7171, pp. 913–916, 2007.

13. Harvey, M. J., G. Giupponi and G. D. Fabritiis, "Accelerating Biomolecular Dynamics in the Microsecond Time Scale", *Journal of Chemical Theory and Computation*, Vol. 5, No. 6, pp. 1632–1639, 2009.

14. Noid, W. G., J.-W. Chu, G. S. Ayton, V. Krishna, S. Izvekov, G. A. Voth, A. Das and H. C. Andersen, "The Multiscale Coarse-Graining Method. I. A Rigorous Bridge Between Atomistic and Coarse-Grained Models", *The Journal of Chemical Physics*, Vol. 128, No. 24, 2008.

15. Dobson, C. M., "Protein Folding and Misfolding", *Nature*, Vol. 426, No. 6968, pp. 884–890, 2003.

16. Onuchic, J. N., Z. Luthey-Schulten and P. G. Wolynes, "Theory of Protein Folding: The Energy Landscape Perspective", *Annual Review of Physical Chemistry*, Vol. 48, No. 1, pp. 545–600, 1997.

17. Bernardi, R. C., M. C. Melo and K. Schulten, "Enhanced Sampling Techniques in Molecular Dynamics Simulations of Biological Systems", *Biochimica et Biophysica Acta - General Subjects*, Vol. 1850, No. 5, pp. 872–877, 2015.

18. Laio, A. and M. Parrinello, "Escaping Free-Energy Minima", *Proceedings of the National Academy of Sciences*, Vol. 99, No. 20, pp. 12562–12566, 2002.

19. Laio, A. and F. L. Gervasio, "Metadynamics: A Method to Simulate Rare Events and Reconstruct the Free Energy in Biophysics, Chemistry and Material Science", *Reports on Progress in Physics*, Vol. 71, No. 12, p. 126601, 2008.

20. Bussi, G., F. L. Gervasio, A. Laio and M. Parrinello, "Free-Energy Landscape for β Hairpin Folding From Combined Parallel Tempering and Metadynamics", *Journal of the American Chemical Society*, Vol. 128, No. 41, pp. 13435–13441, 2006.

21. Gervasio, F. L., A. Laio and M. Parrinello, "Flexible Docking in Solution Using Metadynamics", *Journal of the American Chemical Society*, Vol. 127, No. 8, pp. 2600–2607, 2005.

22. Bussi, G. and A. Laio, "Using Metadynamics to Explore Complex Free-Energy Landscapes", *Nature Reviews Physics*, Vol. 2, No. 4, pp. 200–212, 2020.

23. Torrie, G. M. and J. P. Valleau, "Nonphysical Sampling Distributions in Monte Carlo Free-Energy Estimation: Umbrella Sampling", *Journal of Computational Physics*, Vol. 23, No. 2, pp. 187–199, 1977.

24. Kästner, J., "Umbrella Sampling", *Wiley Interdisciplinary Reviews: Computational Molecular Science*, Vol. 1, No. 6, pp. 932–942, 2011.

25. Darve, E., D. Rodríguez-Gómez and A. Pohorille, "Adaptive Biasing Force Method for Scalar and Vector Free Energy Calculations", *The Journal of Chemical Physics*, Vol. 128, No. 14, 2008.

26. Comer, J., J. C. Gumbart, J. Hénin, T. Lelièvre, A. Pohorille and C. Chipot, "The Adaptive Biasing Force Method: Everything You Always Wanted to Know But Were Afraid to Ask", *The Journal of Physical Chemistry B*, Vol. 119, No. 3,

pp. 1129–1151, 2015.

27. Sugita, Y. and Y. Okamoto, "Replica-Exchange Molecular Dynamics Method for Protein Folding", *Chemical Physics Letters*, Vol. 314, No. 1-2, pp. 141–151, 1999.

28. Yu, T.-Q., J. Lu, C. F. Abrams and E. Vanden-Eijnden, "Multiscale Implementation of Infinite-Swap Replica Exchange Molecular Dynamics", *Proceedings of the National Academy of Sciences*, Vol. 113, No. 42, pp. 11744–11749, 2016.

29. Hashemian, B., D. Millán and M. Arroyo, "Modeling and Enhanced Sampling of Molecular Systems With Smooth and Nonlinear Data-Driven Collective Variables", *The Journal of Chemical Physics*, Vol. 139, No. 21, 2013.

30. Fiorin, G., M. L. Klein and J. Hénin, "Using Collective Variables to Drive Molecular Dynamics Simulations", *Molecular Physics*, Vol. 111, No. 22-23, pp. 3345–3362, 2013.

31. Bhakat, S., "Collective Variable Discovery in the Age of Machine Learning: Reality, Hype and Everything in Between", *Royal Society of Chemistry Advances*, Vol. 12, No. 38, pp. 25010–25024, 2022.

32. Bhakat, S. and P. Soderhjelm, "Flap Dynamics in Pepsin-Like Aspartic Proteases: A Computational Perspective Using Plasmepsin-II and Bace-1 as Model Systems", *Journal of Chemical Information and Modeling*, Vol. 62, No. 4, pp. 914–926, 2022.

33. Dodda, L. S., J. Tirado-Rives and W. L. Jorgensen, "Unbinding Dynamics of Non-Nucleoside Inhibitors From Hiv-1 Reverse Transcriptase", *The Journal of Physical Chemistry B*, Vol. 123, No. 8, pp. 1741–1748, 2018.

34. Tribello, G. A., M. Bonomi, D. Branduardi, C. Camilloni and G. Bussi, "Plumed 2: New Feathers for an Old Bird", *Computer Physics Communications*, Vol. 185, No. 2, pp. 604–613, 2014.

35. Rohrdanz, M. A., W. Zheng and C. Clementi, "Discovering Mountain Passes Via Torchlight: Methods for the Definition of Reaction Coordinates and Pathways in Complex Macromolecular Reactions", *Annual Review of Physical Chemistry*, Vol. 64, pp. 295–316, 2013.

36. Valsson, O., P. Tiwary and M. Parrinello, "Enhancing Important Fluctuations: Rare Events and Metadynamics From a Conceptual Viewpoint", *Annual Review of Physical Chemistry*, Vol. 67, pp. 159–184, 2016.

37. Bernetti, M., M. Bertazzo and M. Masetti, "Data-Driven Molecular Dynamics: A Multifaceted Challenge", *Pharmaceuticals*, Vol. 13, No. 9, p. 253, 2020.

38. Chen, W., A. R. Tan and A. L. Ferguson, "Collective Variable Discovery and Enhanced Sampling Using Autoencoders: Innovations in Network Architecture and Error Function Design", *The Journal of Chemical Physics*, Vol. 149, No. 7, 2018.

39. Chen, W. and A. L. Ferguson, "Molecular Enhanced Sampling With Autoencoders: On-the-Fly Collective Variable Discovery and Accelerated Free Energy Landscape Exploration", *Journal of Computational Chemistry*, Vol. 39, No. 25, pp. 2079–2102, 2018.

40. Lewis-Beck, C. and M. Lewis-Beck, *Applied Regression: An Introduction*, Vol. 22, Sage Publications, Thousand Oaks, CA, 2015.

41. Tan, P.-N., M. Steinbach and V. Kumar, *Introduction to Data Mining*, Vol. 1, Addison-Wesley, Boston, 2006.

42. Sultan, M. M. and V. S. Pande, "Automated Design of Collective Variables Using Supervised Machine Learning", *The Journal of Chemical Physics*, Vol. 149, No. 9, 2018.

43. Mendels, D., G. Piccini and M. Parrinello, "Collective Variables From Local Fluc-

tuations", *The Journal of Physical Chemistry Letters*, Vol. 9, No. 11, pp. 2776–2781, 2018.

44. Peng, J.-H., W. Wang, Y.-Q. Yu, H.-L. Gu and X. Huang, "Clustering Algorithms to Analyze Molecular Dynamics Simulation Trajectories for Complex Chemical and Biological Systems", *Chinese Journal of Chemical Physics*, Vol. 31, No. 4, pp. 404–420, 2018.

45. Tribello, G. A., M. Ceriotti and M. Parrinello, "A Self-Learning Algorithm for Biased Molecular Dynamics", *Proceedings of the National Academy of Sciences*, Vol. 107, No. 41, pp. 17509–17514, 2010.

46. Tribello, G. A. and P. Gasparotto, "Using Dimensionality Reduction to Analyze Protein Trajectories", *Frontiers in Molecular Biosciences*, Vol. 6, p. 46, 2019.

47. Bellman, R., *Adaptive Control Processes: A Guided Tour (Reprint from 1961)*, Vol. 2045 of *Princeton Legacy Library*, Princeton University Press, Princeton, NJ, 2015.

48. Wold, S., K. Esbensen and P. Geladi, "Principal Component Analysis", *Chemometrics and Intelligent Laboratory Systems*, Vol. 2, No. 1-3, pp. 37–52, 1987.

49. Spiwok, V., P. Lipovová and B. Králová, "Metadynamics in Essential Coordinates: Free Energy Simulation of Conformational Changes", *The Journal of Physical Chemistry B*, Vol. 111, No. 12, pp. 3073–3076, 2007, pMID: 17388445.

50. White, E. K., "The Nonlinear Dynamics of Protein Folding", *American Institute of Physics Conference Proceedings*, Vol. 676, pp. 372–372, Melville, NY, 2003.

51. Tenenbaum, J. B., V. d. Silva and J. C. Langford, "A Global Geometric Framework for Nonlinear Dimensionality Reduction", *Science*, Vol. 290, No. 5500, pp. 2319–2323, 2000.

52. Das, P., M. Moll, H. Stamati, L. E. Kavraki and C. Clementi, "Low-Dimensional, Free-Energy Landscapes of Protein-Folding Reactions by Nonlinear Dimensionality Reduction", *Proceedings of the National Academy of Sciences*, Vol. 103, No. 26, pp. 9885–9890, 2006.

53. Ferguson, A. L., A. Z. Panagiotopoulos, P. G. Debenedetti and I. G. Kevrekidis, "Systematic Determination of Order Parameters for Chain Dynamics Using Diffusion Maps", *Proceedings of the National Academy of Sciences*, Vol. 107, No. 31, pp. 13597–13602, 2010.

54. Ferguson, A. L., A. Z. Panagiotopoulos, I. G. Kevrekidis and P. G. Debenedetti, "Nonlinear Dimensionality Reduction in Molecular Simulation: The Diffusion Map Approach", *Chemical Physics Letters*, Vol. 509, No. 1-3, pp. 1–11, 2011.

55. Preto, J. and C. Clementi, "Fast Recovery of Free Energy Landscapes Via Diffusion-Map-Directed Molecular Dynamics", *Physical Chemistry Chemical Physics*, Vol. 16, No. 36, pp. 19181–19191, 2014.

56. Hinton, G. E. and R. R. Salakhutdinov, "Reducing the Dimensionality of Data With Neural Networks", *Science*, Vol. 313, No. 5786, pp. 504–507, 2006.

57. Chen, H. and C. Chipot, "Chasing Collective Variables Using Temporal Data-Driven Strategies", *Quantitative Reviews of Biophysics Discovery*, Vol. 4, p. e2, 2023.

58. Lemke, T. and C. Peter, "Encodermap: Dimensionality Reduction and Generation of Molecule Conformations", *Journal of Chemical Theory and Computation*, Vol. 15, No. 2, pp. 1209–1215, 2019.

59. Trapl, D., I. Horvacanin, V. Mareska, F. Ozcelik, G. Unal and V. Spiwok, "Ancolvar: Approximation of Complex Collective Variables by Artificial Neural Networks for Analysis and Biasing of Molecular Simulations", *Frontiers in Molecular*

Biosciences, Vol. 6, p. 25, 2019.

60. Belkacemi, Z., P. Gkeka, T. Lelièvre and G. Stoltz, "Chasing Collective Variables Using Autoencoders and Biased Trajectories", *Journal of Chemical Theory and Computation*, Vol. 18, No. 1, pp. 59–78, 2021.

61. Eastman, P., J. Swails, J. D. Chodera, R. T. McGibbon, Y. Zhao, K. A. Beauchamp, L.-P. Wang, A. C. Simmonett, M. P. Harrigan, C. D. Stern *et al.*, "OpenMM 7: Rapid Development of High Performance Algorithms for Molecular Dynamics", *PLoS Computational Biology*, Vol. 13, No. 7, p. e1005659, 2017.

62. Bandyopadhyay, S. and J. Mondal, "A Deep Autoencoder Framework for Discovery of Metastable Ensembles in Biomacromolecules", *The Journal of Chemical Physics*, Vol. 155, No. 11, 2021.

63. Ghorbani, M., S. Prasad, J. B. Klauda and B. R. Brooks, "Variational Embedding of Protein Folding Simulations Using Gaussian Mixture Variational Autoencoders", *The Journal of Chemical Physics*, Vol. 155, No. 19, 2021.

64. Bischl, B., M. Binder, M. Lang, T. Pielok, J. Richter, S. Coors, J. Thomas, T. Ullmann, M. Becker, A.-L. Boulesteix *et al.*, "Hyperparameter Optimization: Foundations, Algorithms, Best Practices, and Open Challenges", *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, Vol. 13, No. 2, p. e1484, 2023.

65. Ackley, D. H., G. E. Hinton and T. J. Sejnowski, "A Learning Algorithm for Boltzmann Machines", *Cognitive Science*, Vol. 9, No. 1, pp. 147–169, 1985.

66. Chollet, F. *et al.*, "Keras", `https://keras.io`, 2015, accessed on Jan 15, 2024.

67. Abadi, M., P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "{TensorFlow}: A System for {Large-Scale} Machine Learning", *12th Advanced Computing Systems Association Symposium*

on Operating Systems Design and Implementation, pp. 265–283, Savannah, GA, 2016.

68. García, S., J. Luengo and F. Herrera, *Data Preprocessing in Data Mining*, Vol. 72, Springer, Cham, 2015.

69. Han, J., M. Kamber and J. Pei, *Data Mining Concepts and Techniques Third Edition*, Morgan Kaufmann, Waltham, MA, 2012.

70. Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-Learn: Machine Learning in Python", *The Journal of Machine Learning Research*, Vol. 12, pp. 2825–2830, 2011.

71. Altis, A., M. Otten, P. H. Nguyen, R. Hegger and G. Stock, "Construction of the Free Energy Landscape of Biomolecules Via Dihedral Angle Principal Component Analysis", *The Journal of Chemical Physics*, Vol. 128, No. 24, 2008.

72. Varolgüneş, Y. B., T. Bereau and J. F. Rudzinski, "Interpretable Embeddings From Molecular Simulations Using Gaussian Mixture Variational Autoencoders", *Machine Learning: Science and Technology*, Vol. 1, No. 1, p. 015012, 2020.

73. Smith, C. A., "Structure, Function and Dynamics in the Mur Family of Bacterial Cell Wall Ligases", *Journal of Molecular Biology*, Vol. 362, No. 4, pp. 640–655, 2006.

74. Azam, M. A. and S. Jupudi, "MurD Inhibitors as Antibacterial Agents: A Review", *Chemical Papers*, Vol. 74, No. 6, pp. 1697–1708, 2020.

75. Hervin, V., V. Roy and L. A. Agrofoglio, "Antibiotics and Antibiotic Resistance—Mur Ligases as an Antibacterial Target", *Molecules*, Vol. 28, No. 24, p. 8076, 2023.

76. Bertrand, J. A., E. Fanchon, L. Martin, L. Chantalat, G. Auger, D. Blanot, J. van Heijenoort and O. Dideberg, ""Open" Structures of MurD: Domain Movements and Structural Similarities With Folylpolyglutamate Synthetase", *Journal of Molecular Biology*, Vol. 301, No. 5, pp. 1257–1266, 2000.

77. Bertrand, J. A., G. Auger, L. Martin, E. Fanchon, D. Blanot, D. Le Beller, J. van Heijenoort and O. Dideberg, "Determination of the MurD Mechanism Through Crystallographic Analysis of Enzyme Complexes", *Journal of Molecular Biology*, Vol. 289, No. 3, pp. 579–590, 1999.

78. Degiacomi, M. T., "Coupling Molecular Dynamics and Deep Learning to Mine Protein Conformational Space", *Structure*, Vol. 27, No. 6, pp. 1034–1040, 2019.

79. Webb, B. and A. Sali, "Comparative Protein Structure Modeling Using Modeller", *Current Protocols in Bioinformatics*, Vol. 54, No. 1, pp. 5–6, 2016.

80. Lindahl, E., B. Hess and D. van der Spoel, "Gromacs 3.0: A Package for Molecular Simulation and Trajectory Analysis", *Journal of Molecular Modeling*, Vol. 7, No. 8, pp. 306–317, 2001.

81. Dzeja, P. and A. Terzic, "Adenylate Kinase and Amp Signaling Networks: Metabolic Monitoring, Signal Communication and Body Energy Sensing", *International Journal of Molecular Sciences*, Vol. 10, No. 4, pp. 1729–1772, 2009.

82. Boison, D. and M. F. Jarvis, "Adenosine Kinase: A Key Regulator of Purinergic Physiology", *Biochemical Pharmacology*, Vol. 187, p. 114321, 2021.

83. Kowaluk, E. A. and M. F. Jarvis, "Therapeutic Potential of Adenosine Kinase Inhibitors", *Expert Opinion on Investigational Drugs*, Vol. 9, No. 3, pp. 551–564, 2000.

84. Li, D., M. S. Liu and B. Ji, "Mapping the Dynamics Landscape of Conformational Transitions in Enzyme: The Adenylate Kinase Case", *Biophysical Journal*, Vol.

109, No. 3, pp. 647–660, 2015.

85. Müller, C., G. Schlauderer, J. Reinstein and G. E. Schulz, "Adenylate Kinase Motions During Catalysis: An Energetic Counterweight Balancing Substrate Binding", *Structure*, Vol. 4, No. 2, pp. 147–156, 1996.

86. Müller, C. W. and G. E. Schulz, "Structure of the Complex Between Adenylate Kinase From Escherichia Coli and the Inhibitor Ap5A Refined at 1.9 Å Resolution: A Model for a Catalytic Transition State", *Journal of Molecular Biology*, Vol. 224, No. 1, pp. 159–177, 1992.

87. Peng, C., J. Wang, Y. Shi, Z. Xu and W. Zhu, "Increasing the Sampling Efficiency of Protein Conformational Change by Combining a Modified Replica Exchange Molecular Dynamics and Normal Mode Analysis", *Journal of Chemical Theory and Computation*, Vol. 17, No. 1, pp. 13–28, 2020.

88. Gowers, R. J., M. Linke, J. Barnoud, T. J. Reddy, M. N. Melo, S. L. Seyler, J. Domanski, D. L. Dotson, S. Buchoux, I. M. Kenney *et al.*, "Mdanalysis: A Python Package for the Rapid Analysis of Molecular Dynamics Simulations", *Proceedings of the 15th Python in Science Conference*, Vol. 98, p. 105, SciPy Austin, TX, 2016.

89. Michaud-Agrawal, N., E. J. Denning, T. B. Woolf and O. Beckstein, "Mdanalysis: A Toolkit for the Analysis of Molecular Dynamics Simulations", *Journal of Computational Chemistry*, Vol. 32, No. 10, pp. 2319–2327, 2011.

90. Beckstein, O., E. J. Denning, J. R. Perilla and T. B. Woolf, "Zipping and Unzipping of Adenylate Kinase: Atomistic Insights Into the Ensemble of Open Closed Transitions", *Journal of Molecular Biology*, Vol. 394, No. 1, pp. 160–176, 2009.

91. Mishra, C. and D. Gupta, "Deep Machine Learning and Neural Networks: An Overview", *Institute of Advanced Engineering and Science International Journal*

of Artificial Intelligence, Vol. 6, No. 2, p. 66, 2017.

92. Sharma, S., S. Sharma and A. Athaiya, "Activation Functions in Neural Networks", *Towards Data Science*, Vol. 6, No. 12, pp. 310–316, 2017.

93. Waoo, A. A. and B. K. Soni, "Performance Analysis of Sigmoid and Relu Activation Functions in Deep Neural Network", *Intelligent Systems: Proceedings of Symposium on Computational Intelligence and Systems*, pp. 39–52, Springer, Berlin, 2021.

94. Hahnloser, R. H., R. Sarpeshkar, M. A. Mahowald, R. J. Douglas and H. S. Seung, "Digital Selection and Analogue Amplification Coexist in a Cortex-Inspired Silicon Circuit", *Nature*, Vol. 405, No. 6789, pp. 947–951, 2000.

95. Nair, V. and G. E. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines", *Proceedings of the 27th International Conference on Machine Learning*, pp. 807–814, Haifa, Palestine, 2010.

96. Ramachandran, P., B. Zoph and Q. V. Le, "Searching for Activation Functions", *arXiv preprint arXiv:1710.05941* , 2017.

97. Goodfellow, I., Y. Bengio and A. Courville, *Deep Learning*, Massachusetts Institute of Technology Press, Cambridge, MA, 2016.

98. Maas, A. L., A. Y. Hannun, A. Y. Ng *et al.*, "Rectifier Nonlinearities Improve Neural Network Acoustic Models", *Proceedings of International Conference on Machine Learning*, Vol. 30, p. 3, Atlanta, GA, 2013.

99. Klambauer, G., T. Unterthiner, A. Mayr and S. Hochreiter, "Self-Normalizing Neural Networks", *Advances in Neural Information Processing Systems*, Vol. 30, Long Beach, CA, 2017.

100. Rojas, R., "The Backpropagation Algorithm", *Neural Networks: A Systematic*

Introduction, Springer, Berlin, 1996.

101. Amari, S.-i., "Backpropagation and Stochastic Gradient Descent Method", *Neurocomputing*, Vol. 5, No. 4-5, pp. 185–196, 1993.

102. Kingma, D. P. and J. Ba, "Adam: A Method for Stochastic Optimization", *arXiv preprint arXiv:1412.6980*, 2014.

103. Ruder, S., "An Overview of Gradient Descent Optimization Algorithms", *arXiv preprint arXiv:1609.04747*, 2016.

104. Hawkins, D. M., "The Problem of Overfitting", *Journal of Chemical Information and Computer Sciences*, Vol. 44, No. 1, pp. 1–12, 2004.

105. Tibshirani, R., "Regression Shrinkage and Selection Via the Lasso", *Journal of the Royal Statistical Society Series B: Statistical Methodology*, Vol. 58, No. 1, pp. 267–288, 1996.

106. Hoerl, A. E. and R. W. Kennard, "Ridge Regression: Biased Estimation for Nonorthogonal Problems", *Technometrics*, Vol. 42, No. 1, pp. 80–86, 2000.

107. Bishop, C. M. and N. M. Nasrabadi, *Pattern Recognition and Machine Learning*, Vol. 4, Springer, New York, NY, 2006.

108. Bishop, C. M., *Neural Networks for Pattern Recognition*, Oxford University Press, Oxford, 1995.

109. Burke, L. I., "Introduction to Artificial Neural Systems for Pattern Recognition", *Computers & Operations Research*, Vol. 18, No. 2, pp. 211–220, 1991.

110. Wang, Y., H. Yao and S. Zhao, "Auto-Encoder Based Dimensionality Reduction", *Neurocomputing*, Vol. 184, pp. 232–242, 2016.

111. Fan, Y. J., "Autoencoder Node Saliency: Selecting Relevant Latent Representations", *Pattern Recognition*, Vol. 88, pp. 643–653, 2019.

112. Brownlee, J., "What Is the Difference Between a Batch and an Epoch in a Neural Network", `https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/`, 2018, accessed: 2024-05-17.

113. Keskar, N. S., D. Mudigere, J. Nocedal, M. Smelyanskiy and P. T. P. Tang, "On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima", *arXiv preprint arXiv:1609.04836*, 2016.

114. Wang, H., K. Ren and J. Song, "A Closer Look at Batch Size in Mini-Batch Training of Deep Auto-Encoders", *3rd Institute of Electrical and Electronics Engineers International Conference on Computer and Communications*, pp. 2756–2761, Chengdu, 2017.

115. Feurer, M. and F. Hutter, "Hyperparameter Optimization", *Automated Machine Learning: Methods, Systems, Challenges*, pp. 3–33, Springer, Cham, 2019.

116. Kohavi, R. and G. H. John, "Automatic Parameter Selection by Minimizing Estimated Error", *Machine Learning Proceedings 1995*, pp. 304–312, Elsevier, San Francisco, CA, 1995.

117. Mantovani, R. G., T. Horváth, R. Cerri, J. Vanschoren and A. C. De Carvalho, "Hyper-Parameter Tuning of a Decision Tree Induction Algorithm", *5th Brazilian Conference on Intelligent Systems*, pp. 37–42, Institute of Electrical and Electronics Engineers, Recife, 2016.

118. Thornton, C., F. Hutter, H. H. Hoos and K. Leyton-Brown, "Auto-Weka: Combined Selection and Hyperparameter Optimization of Classification Algorithms", *Proceedings of the 19th International Conference on Knowledge Discovery and Data Mining*, pp. 847–855, Association for Computing Machinery, Chicago, IL,

2013.

119. Sanders, S. and C. Giraud-Carrier, "Informing the Use of Hyperparameter Optimization Through Metalearning", *International Conference on Data Mining*, pp. 1051–1056, Institute of Electrical and Electronics Engineers, New Orleans, LA, 2017.

120. Bergstra, J. and Y. Bengio, "Random Search for Hyper-Parameter Optimization", *Journal of Machine Learning Research*, Vol. 13, No. 2, 2012.

121. Young, S. R., D. C. Rose, T. P. Karnowski, S.-H. Lim and R. M. Patton, "Optimizing Deep Learning Hyper-Parameters Through an Evolutionary Algorithm", *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments*, pp. 1–5, Austin, TX, 2015.

122. Darwin, C., *On the Origin of Species by Means of Natural Selection; or, the Preservation of Favored Races in the Struggle for Life*, John Murray, London, 1876.

123. Wallace, A. R., "On the Tendency of Varieties to Depart Indefinitely From the Original Type", *Journal of Linnean Society of London, Zoology*, Vol. 3, No. 9, pp. 53–62, 1858.

124. Holland, J. H., "Genetic Algorithms", *Scientific American*, Vol. 267, No. 1, pp. 66–73, 1992.

125. Raji, I. D., H. Bello-Salau, I. J. Umoh, A. J. Onumanyi, M. A. Adegboye and A. T. Salawudeen, "Simple Deterministic Selection-Based Genetic Algorithm for Hyperparameter Tuning of Machine Learning Models", *Applied Sciences*, Vol. 12, No. 3, p. 1186, 2022.

126. Xiao, X., M. Yan, S. Basodi, C. Ji and Y. Pan, "Efficient Hyperparameter Optimization in Deep Learning Using a Variable Length Genetic Algorithm", *arXiv*

preprint arXiv:2006.12703, 2020.

127. Lee, S., J. Kim, H. Kang, D.-Y. Kang and J. Park, "Genetic Algorithm Based Deep Learning Neural Network Structure and Hyperparameter Optimization", *Applied Sciences*, Vol. 11, No. 2, p. 744, 2021.

128. Mitchell, M., *An Introduction to Genetic Algorithms*, Massachusetts Institute of Technology Press, Cambridge, MA, 1998.

129. "Python Package Index", `https://pypi.org/`, accessed on Jan 15, 2024.

130. Razali, N. M., Y. B. Wah *et al.*, "Power Comparisons of Shapiro-Wilk, Kolmogorov-Smirnov, Lilliefors and Anderson-Darling Tests", *Journal of Statistical Modeling and Analytics*, Vol. 2, No. 1, pp. 21–33, 2011.

131. McKnight, P. E. and J. Najab, "Mann-Whitney U Test", *The Corsini Encyclopedia of Psychology*, pp. 1–1, Wiley Online Library, Hoboken, NJ, 2010.

132. Virtanen, P., R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python", *Nature Methods*, Vol. 17, pp. 261–272, 2020.

133. Rotkiewicz, P. and J. Skolnick, "Fast Procedure for Reconstruction of Full-Atom Protein Models From Reduced Representations", *Journal of Computational Chemistry*, Vol. 29, No. 9, pp. 1460–1465, 2008.

134. Kabsch, W. and C. Sander, "Dictionary of Protein Secondary Structure: Pattern Recognition of Hydrogen-Bonded and Geometrical Features", *Biopolymers:*

Original Research on Biomolecules, Vol. 22, No. 12, pp. 2577–2637, 1983.

135. Prechelt, L., "Early Stopping-But When?", *Neural Networks: Tricks of the Trade*, pp. 55–69, Springer, Berlin, 2002.

APPENDIX A: SOFTWARE

The Python scripts and data used for this thesis can be reached from this link: `https://drive.google.com/drive/folders/1l--cpiGh4QWismiPBhp5Lb_LBJ_2MOJw?usp=sharing`. Below is a detailed explanation of the organization of the folders.

/Semi-Supervised Autoencoder: This folder contains the data and Python scripts for running a semi-supervised autoencoder using MurD or AdK protein simulation trajectories for their two different cases.

/Genetic Algorithm: This folder contains the data and Python scripts for running a genetic algorithm for hyperparameter optimization of semi-supervised autoencoder using MurD or AdK protein simulation trajectories for their two different cases.