Development of automated computational methods for the redesign of protein dynamics using biomolecular simulations and machine learning

Dissertation

Submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy (Ph.D)

of the

Department of Computer Science,
Brunel University London

Submission date: 31 January 2025

Author:

Namir Oues

Supervisors:

Doctor Alessandro Pandini

Doctor Sarath Dantu

RDA:

Professor Yongmin Li

Declaration

I declare that this thesis is my original work and has been completed solely by me, Namir Oues. The research contained within has not been submitted for any other degree or qualification award. Some parts of the work have been published previously and are explicitly acknowledged in the text where relevant. All material sources have been appropriately cited, and full references are provided.

Publications

This is a list of publications lead-authored and co-authored by the author of this thesis during the PhD time frame:

Oues N., Dantu S.C., Patel R.J., Pandini A., MDSubSampler: *A posteriori* sampling of important protein conformations from biomolecular simulations, Bioinformatics, Volume 39, Issue 7, July 2023, btad427, https://doi.org/10.1093/bioinformatics/btad427

Oues N., Pandini A., MDAutoMut: A Toolkit for an automated workflow of redesigning protein dynamics through mutation engineering. *Manuscript in preparation.*

Hossein Nezhad F., **Oues N.**, Meli M., Pandini A., MDGraphEmb: A toolkit for encoding molecular dynamics simulations with graph embedding". *Manuscript under review with requested revisions.*

Abstract

Proteins are responsible for almost all biological mechanisms, and their three-dimensional structures and dynamics define their function. In recent years, outstanding advances have been made in protein design. However, redesigning protein dynamics to achieve desired properties or states remains a significant challenge in computational protein design. This thesis addresses this gap by introducing three novel toolkits—MDSubSampler, MDAutoMut, and MDAutoPredict—developed to integrate biomolecular simulations with machine learning for the automated redesign of protein dynamics.

MDSubSampler is designed to preprocess and *a posteriori* subsample molecular dynamics simulations, preserving critical dynamic information while reducing noise and data complexity. Its application demonstrates effective noise reduction and compatibility with machine learning workflows, validated using adenylate kinase as a model system. MDAutoMut automates mutation generation, simulation, and analysis, facilitating systematic identification of mutations that have a desired impact on protein dynamics. This toolkit successfully identifies mutations on adenylate kinase structure shifting dynamics towards a closed conformation, validated by literature benchmarks. MDAutoPredict extends the workflow by using machine learning models to predict conformational states from molecular dynamics data, offering an adaptable framework for dynamic state prediction.

These contributions represent an advance in computational protein design, providing scalable, automated solutions for mutation engineering and dynamic prediction. The toolkits are modular, extensible, and integrated with well-consolidated libraries, ensuring broad applicability across protein engineering challenges. This research highlights the potential of combining biomolecular simulations with machine learning to redesign protein dynamics and sets the stage for future innovations in computational biology.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Dr. Alessandro Pandini, for giving me the opportunity to work on this project. I am deeply thankful for his continuous support, patience, and guidance throughout my PhD. His feedback and advice at every stage of the project were invaluable, and his encouragement, even during challenging times, meant a great deal to me.

I would also like to thank Dr. Sarath Dantu for his help and thoughtful suggestions, as well as his input on the technical and theoretical aspects of this work, which played an important role in its completion.

My thanks extend to Dr. Ferdoos Hossein Nezhad for being both a supportive colleague and a great friend during difficult times, as well as for their feedback and advice on various parts of the project.

I am grateful to Dr. Arianna Fornili and Dr. Massimiliano Meli for their helpful feedback and advice during the preparation of my first publication. Their input significantly improved the work.

Last but not least, I would like to thank deeply my husband, Alexandros Peitsinis, for his love, patience, and support, which have been a source of strength throughout this journey, but also my family and friends for their unwavering support and encouragement.

This PhD is supported by a scholarship from Brunel University London EPSRC DTP (grant no. EP/T518116/1). This project made use of time on HPC granted via the UK High-End Computing Consortium for Biomolecular Simulation, HECBioSim (http://hecbiosim.ac.uk), supported by EPSRC (grant no. EP/R029407/1).

Table of Contents

Table of Figures

1 Introduction

Proteins are essential macromolecules responsible for nearly all biological processes, acting as catalysts, transporters, signalling molecules, and more. Their ability to perform such diverse functions derives from their three-dimensional, physico-chemical, and dynamic properties. Structural changes allow proteins to transition between functional states, enabling processes like enzymatic activity, molecular recognition, and signalling. Understanding protein dynamics is fundamental to explaining biological functions and critical for advancing fields such as drug design, synthetic biology, and disease prevention.

While experimental techniques such as X-ray crystallography and Nuclear Magnetic Resonance (NMR) spectroscopy are used to study protein structures, they face limitations in capturing the protein dynamic behaviour at the atomistic level [1]. Molecular dynamics (MD) simulations have become a widely used computational approach to overcome these limitations, offering atomistic-level details on protein motions over time [2], [3]. However, the large and complex MD trajectories present significant challenges for analysis. The data are often noisy, and due to their high dimensionality, it is challenging to identify functionally relevant conformations that are generally a subset of the phase space accessible to the protein [4]. Researchers currently rely on combinations of tools, approaches, and methods, as no unified analysis strategy exists for efficiently removing the noise, increasing the signal, and determining the important conformations of these large trajectories.

Most proteins fold into specific three-dimensional structures determined by their amino acid sequences, and these folded states are connected to proteins' biological functions. Even small changes in the sequence, such as single mutations, can dramatically impact protein dynamics, potentially disrupt their function and leading to disease. On the contrary, targeted mutations can enhance protein stability or improve function, offering strategies for therapeutic and engineering applications. Therefore, understanding the impact of mutations on protein dynamics is crucial in the computational protein design field.

The field of computational protein design has achieved remarkable success, as confirmed by the 2024 Nobel Prize in Chemistry award [5]. However, despite these advancements, the redesign of protein dynamics remains an unexplored area in protein design. Although several strategies have been developed to optimise active sites and sequences, these approaches mainly focus on static protein structures, often overlooking their dynamics. Multistate design has provided some insights into improving protein properties, but there is currently no unified

analysis strategy to automate the redesign of protein dynamics. This thesis aims to contribute to addressing this critical gap.

Despite advancements in machine learning (ML) and deep learning (DL), existing methods for predicting atomistic and molecular properties of protein systems lack integration with MD [6], [7]. They would benefit from the availability of automated workflows. Current techniques offer valuable insights but fail to provide a single, adaptable, and expandable tool that seamlessly combines data preprocessing, mutational analysis, and predictive modelling.

This research addresses some of these gaps by developing a new methodological approach based on three novel computational tools—MDSubSampler (MDSS), MDAutoMut (MDAM), and MDAutoPredict (MDAP). These tools are designed and implemented to address specific aspects of the challenge: MDSS focuses on efficient data preprocessing of MD data, MDAM enables systematic mutation analysis, and MDAP provides dynamic prediction capabilities. Together, these tools aim to establish a comprehensive and automated framework for advancing the computational protein design of MD.

1.1 Research questions

The research questions in this thesis aim to advance methods and tools in protein redesign through automated simulation and evaluation of the effect of mutations on protein dynamics. The first step is to preprocess the data used in this study. The second step is developing the method to achieve the study's goal. The last step is extending the method to a more advanced approach to help solve different research problems. The following research questions were considered when designing the solution approach for this thesis.

1. **Processing MD simulation data:**

   How can the volume of MD simulation data be effectively managed to enable its use in automated workflows without exceeding computational resource limitations? At the same time, how can the complexity of these data formats be addressed to ensure easy integration into ML/DL pipelines?

   To address this question, a robust and adaptable MD processing framework is required to systematically reduce noise while preserving critical simulation information and reformatting the data into ML/DL-compatible structures.

2. **Automating protein dynamics redesign:**

   How can computational strategies be developed to fine-tune protein dynamics through targeted mutations, and how can these workflows be automated, integrated with existing computational libraries, and scaled effectively using high-performance computing (HPC) resources?

   Redesigning protein dynamics through targeted mutations requires a unified, scalable computational strategy. This strategy involves developing an automated framework capable of generating, simulating, and analysing mutations to identify those that can achieve desired protein properties. Furthermore, scaling these workflows on HPC platforms is essential to efficiently process the extensive simulations and mutation datasets.

3. **Predictive modelling for protein dynamics**

   How can the automated workflow be extended to perform predictions in combination with ML/DL pipelines, and which ML models are most appropriate for predictive tasks in protein dynamics?

   ML provides transformative opportunities to analyse protein dynamics by making predictions from MD simulation data. Additionally, benchmarking several ML models for tasks like conformational state prediction will evaluate how ML can be applied within this context.

1.2 Aims and Objectives

Aim

This research aims to deliver an innovative framework for integrating biomolecular simulations with predictive computational methods, redesigning protein dynamics, and advancing protein engineering and design.

Objectives

1. **Develop and validate a toolkit for preprocessing and subsampling MD simulation data:**

Design, implement, and test the MDSS toolkit to efficiently process MD simulation data, reducing noise while preserving the distribution of key geometric properties. This tool will ensure that MD data can be transformed into ML-compatible formats, enabling advanced predictive analyses.

2. **Automate the analysis of mutations' impact on protein dynamics:**

   Design, implement, and test MDAM, a toolkit to automatically generate, simulate, and compare wild-type and mutant protein dynamics. This tool will facilitate evaluating how specific mutations can impact protein dynamics. MDAM will implement automated workflows to identify optimal solutions to the problem of redesigning protein dynamics.

3. **Explore and integrate machine learning techniques for predictive modelling in protein design:**

   Design, implement, and test MDAP, an ML framework for predicting protein states of a protein system using MD data processed with MDSS. The objective includes benchmarking several ML models to identify the best-suited approaches for conformational state prediction. MDAP offers an extendable and adaptable framework that can use MD simulation data for supervised learning predictions with a target variable describing conformational or state properties.

1.3 Novel contribution to science

This thesis presents a transformative framework for redesigning protein dynamics and addresses some of the critical limitations in MD simulations and computational protein design. The heart of this contribution lies in developing three novel, integrated toolkits – MDSS, MDAM, and MDAP. These toolkits provide a cohesive solution to the challenges associated with large-scale MD data preprocessing, mutation analysis, and predictive modelling of protein dynamics. Together, they form a unified workflow that is customisable and scalable and designed to advance protein redesign methods through the seamless integration of computational and ML approaches.

The MDSS toolkit introduces *a posteriori* efficient subsampling of large MD simulation trajectories, ensuring important information is preserved while reducing data complexity and noise. MDAM automates the generation and analysis of mutations, allowing researchers to

systematically and heuristically identify their specific effects on protein dynamics. Finally, MDAP leverages advanced ML techniques to predict dynamic behaviour, providing an extensible framework for data-driven exploration of protein functions. A foreseeable extension of this approach includes state-prediction capabilities, where the most promising mutations identified by MDAM are tested for their ability to generate desired states, as annotated by MDAP.

These contributions fill long-standing gaps in computational biology by providing easy-to-use, open-source tools validated through rigorous adenylate kinase (ADK) enzyme testing. Integrating these toolkits into established libraries and software (e.g. GROMACS [2], *MDAnalysis* [8], *PyRosetta* [9]) ensures their accessibility and applicability to various research problems.

1.4 Novel toolkits

MDSubSampler

MDSS is an object-oriented Python library designed to preprocess MD simulation data. The tool can perform *a posteriori* subsampling of MD trajectories while ensuring that critical information of the data is preserved. MDSS uses statistical sampling methods and dissimilarity measures to evaluate the sampling. Specifically, the sampling is evaluated by assessing the distance between distributions of original and sample trajectories for relevant geometric properties. The tool efficiently identifies the most important conformations in large trajectories by reducing the noise and increasing the signal.

MDSS is validated using the ADK example system, successfully identifying critical open and closed conformational states of the protein's LID domain. Root Mean Square Deviation (RMSD) distribution analysis demonstrated its ability to capture essential dynamics while significantly reducing data size, ensuring compatibility with ML workflows.

MDAutoMut

MDAM automates the evaluation of mutation impacts on protein dynamics. This toolkit runs in an automated workflow for generating simulations, mutation engineering, and evaluating protein dynamics in MD simulation data. The simulation engine can prepare and simulate the wild type and mutants, and it is performed by integration with GROMACS via the gmxapi Python interface. Mutation engineering is done for single, double, or multiple mutations by

integrating with *PyRosetta*. Finally, the evaluation of dynamics is done by assessing the distribution of properties that capture the system's dynamics, and it is performed by integration with MDSS. The goal is to find mutations that have the desired impact on the system's dynamics.

MDAM is validated with a proof-of-concept study on the ADK system. The protein provides a clear example of protein dynamics since it has two distinct states (open and closed). Given two positions in the ADK structure, the goal is to identify which mutations can increase sampling of close ADK state when scanning across all 20 different amino acids. The change in ADK dynamics is evaluated by assessing relevant properties that describe ADK's closure. The proof-of-concept is done on known mutations from the literature that can achieve ADK's closure. MDAM provides a framework for systematic and heuristic scanning for the space search. Its modular design allows seamless integration with MDSS and MDAutoPredict tools.

MDAutoPredict

MDAP is a predictive modelling framework that uses ML to analyse MD data generated by MDAM and processed by MDSS. The tool can predict protein conformational states using MD simulation data as inputs. The toolkit provides a framework with several ML techniques to identify which is more appropriate for specific research problems involving MD trajectories.

MDAP is tested on the ADK as an example system. It performs ML classification prediction of the system's states (e.g., open, closed). While this thesis focuses on validating the toolkit with a simple example scenario, the tool can expand into more complex workflows.

1.5 Thesis overview

This work is divided into six chapters, including **Chapter 1**. Below is a description of the remaining five chapters.

Chapter 2 provides a literature review and establishes the theoretical foundations of protein structure, dynamics, and function. It explores key experimental and computational methods essential to computational protein design, including MD simulations and ML approaches. The limitations of existing tools in redesigning protein dynamics are acknowledged.

Chapter 3 features the methods used in this research. It describes ADK as a model protein system used in this study. The protocols for generating MD simulations for testing the three

tools are also described. Additionally, the chapter discusses the tools' use of high-performance computing resources for scalability and their implementation in Docker environments for reproducibility.

Chapter 4 contains the design, implementation, and testing of the three novel toolkits developed in this research: MDSS, MDAM and MDAP. It also highlights the modularity and user-friendly structure, ensuring accessibility to researchers with different expertise.

Chapter 5 shows the results by demonstrating the toolkit's application on the example system (ADK). MDSS demonstrates its effectiveness in subsampling large trajectories *a posteriori* without information loss. MDAM's capabilities are validated through a proof-of-concept mutation study that shifts the sampling of the ADK state from open to closed. MDAP is evaluated for its prediction accuracy and uses MD simulation data to classify conformational states.

Finally, **Chapter 6** concludes the research work and discusses the results. Future directions are suggested, especially future integrations of these tools with more complex packages and workflows, setting the stage for further advances in computational protein design.

2 Literature Review

This chapter provides a theoretical background on protein structure, dynamics, function, and design. It examines how conformational flexibility enables proteins to perform complex biological functions, beginning with protein structure and folding principles. The chapter introduces MD simulations as a key computational tool for exploring protein dynamics at the atomistic level, alongside enhanced sampling techniques and ML approaches for analysing conformational changes.

The discussion then progresses to protein design methods, examining traditional approaches and recent advances in multistate design and computational tools for predicting mutational impact on protein properties. The chapter concludes by addressing the current challenges in redesigning protein dynamics, highlighting the limitations of existing methods and the need for new strategies to better account for protein dynamics.

The following diagram (Figure 2.1) illustrates the flow of topics covered in this chapter, moving from the foundational aspects of protein structure to redesigning protein dynamics. Short summaries of each subsection are reported below.

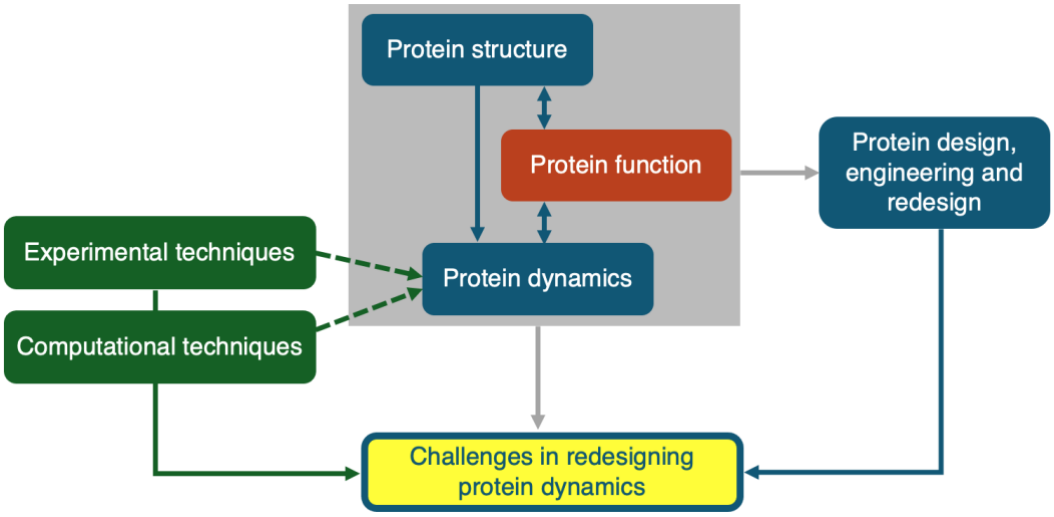


Figure 2.1 Hierarchy of concepts in chapter two. The interplay between protein structure, dynamics, and function and its implications for protein design and engineering. Protein structure forms the basis for biological function, as shown by the unidirectional arrow, while protein dynamics serve as a bridge between structure and function, represented by the bidirectional arrows. Experimental and computational approaches (dashed arrows) provide insights into protein dynamics, which inform protein design, engineering, and redesign. The challenges in redesigning protein dynamics, highlighted in the yellow box, stem from the complexities of modelling conformational flexibility and its impact on protein function.

2.1 Protein structures. This section provides an overview of protein structures, discussing how proteins fold from linear amino acid sequences into complex, three-dimensional conformations. It explores the traditional classification in structural levels—primary, secondary, tertiary, and quaternary—and how folding and stability are critical for protein function.

2.2. Protein function. This section describes how proteins utilise their structure and flexibility to perform specific biological functions. It explains how conformational changes enable proteins to interact with other molecules and to sample different functional states.

2.3 Protein dynamics. This section examines the conformational changes and their roles in biological functions. Both experimental and computational techniques are introduced, with emphasis on MD simulations. An overview of enhanced sampling techniques and an explanation of MD integration with ML approaches are presented.

2.4 Protein design, engineering, and redesign. This section introduces protein design and engineering, covering both template-based and *de novo* approaches and focusing on recent advancements in multistate design. It provides an overview of the computational tools developed for protein design and structural prediction, mutation impact assessment, and stability optimisation. The section sets the scene for this research work by identifying the critical gap in the current methodologies: redesigning protein dynamics.

2.5 Challenges in redesigning protein dynamics. The section provides an overview of the challenges and limitations when redesigning protein dynamics due to current limitations in modelling the impact of mutations on the conformational flexibility of biomolecules.

2.1 Protein structure

Proteins are biomolecules, often referred to as the workhorses of the cell. They are involved in various cellular functions, from providing structural support and catalysing metabolic reactions to facilitating immune responses and transmitting signals between cells [10], [11]. A protein's ability to perform complex functions efficiently is closely linked to its three-dimensional structure. Specifically, to perform their functions, proteins typically fold into specific three-dimensional shapes and undergo structural changes [12], [13].

Changes in protein structure, whether caused by mutations, environmental factors or other perturbations, can impact their stability, dynamics, and function [14]. These changes may result in subtle shifts that affect functional dynamics, more pronounced changes that

compromise protein stability, or, in severe cases, lead to misfolding and aggregation [15], [16]. Understanding the dynamic behaviour of proteins, especially mutational impact, is crucial to advance drug design, biotechnology, and the development of treatments for these diseases [17], [18].

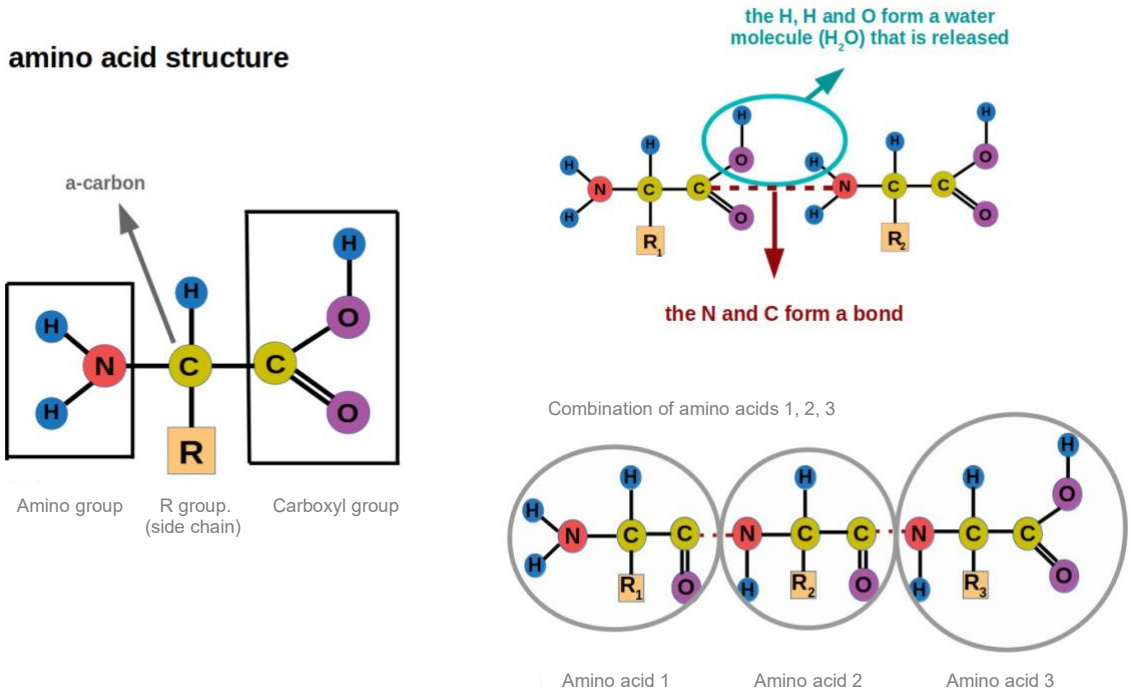


Figure 2.2 Structure of an amino acid. Left: highlighting the amino group, carboxyl group, and variable R group attached to the α-carbon. Top-Right: Formation of a peptide bond between two amino acids, where a molecule of water (H_2O) is released. Bottom-Right: A sequential combination of three amino acids forms a polypeptide chain with a repeating backbone structure.

Chemically, proteins are linear polymers made of amino acids connected by peptide bonds (Figure 2.2). The order in which amino acids are arranged defines the resulting three-dimensional structure, which determines the protein's function [13]. However, it is known that functional adaptation and modulation can also be influenced by factors beyond the isolated protein structure, such as environmental signals and molecular interactions [19]. While many proteins adopt well-defined three-dimensional structures, others, such as intrinsically disordered proteins, lack a stable structure under physiological conditions but still perform important biological functions [20], [21]. In addition, a rare class of proteins, known as metamorphic proteins, can adopt multiple stable conformations, enabling them to perform their functions depending on their environment or interaction with other molecules [22]. These proteins represent a small but significant exception to the general principle of one stable structure per protein.

Before being chemically connected to form the polypeptide chain, amino acids are composed of an alpha carbon atom attached to an amino group (which is typically protonated as $-NH_3^+$ under physiological conditions), a carboxyl group (which is typically deprotonated as $-COO^-$ under physiological conditions), and a unique side chain referred to as the R group (Figure 2.2). Once integrated into the polypeptide chain through peptide bonds, the amino acid becomes known as an amino acid residue. In this process, the amino and carboxyl groups join to form the peptide bond as a byproduct through a condensation reaction (Figure 2.2), while the R group remains free. Each amino acid is characterised by a distinct side chain called R group. The specific properties of the R group, such as its size, shape, and physicochemical properties, including polarity, charge, and hydrophobicity, are fundamentally important for the protein structure.

Four levels are commonly used to interpret protein structure—primary, secondary, tertiary, and quaternary—which helps in understanding their behaviour at the atomic level [10] (Figure 2.3). The amino acid sequence, the primary structure, contains much information required for a protein to attain its final shape and function [23]. In some cases, however, interactions with partner molecules or cellular components are also necessary for the protein to reach its native state [13].

Secondary structures arise from local hydrogen bonds between amino acids. The most common structures are alpha helices and beta sheets [10]. The alpha helix is a right-handed structure, stabilised by hydrogen bonds formed between the backbone atoms of every fourth amino acid, with side chains extending outward and completing one turn approximately every 3.6 residues [23]. In contrast, beta sheets consist of extended beta strands linked by hydrogen bonds between adjacent strands, forming a sheet-like structure where side chains alternate above and below the plane [10].

The tertiary structure represents the complete three-dimensional arrangement of a protein, stabilised by a combination of interactions, including hydrophobic interactions, hydrogen bonds, ionic interactions, and van der Waals forces [13], [24]. A complex interplay of these forces drives the folding process leading to this structure: while hydrophobic interactions drive initial chain collapse, the specific native structure emerges through the progressive formation of hydrogen bonds, optimal packing achieved by van der Waals interactions, and establishment of ionic bonds, with their relative contributions varying by local protein environment [10].

Finally, Proteins composed of multiple polypeptide chains adopt a quaternary structure, where individual subunits interact to function as a single, cohesive complex [23].

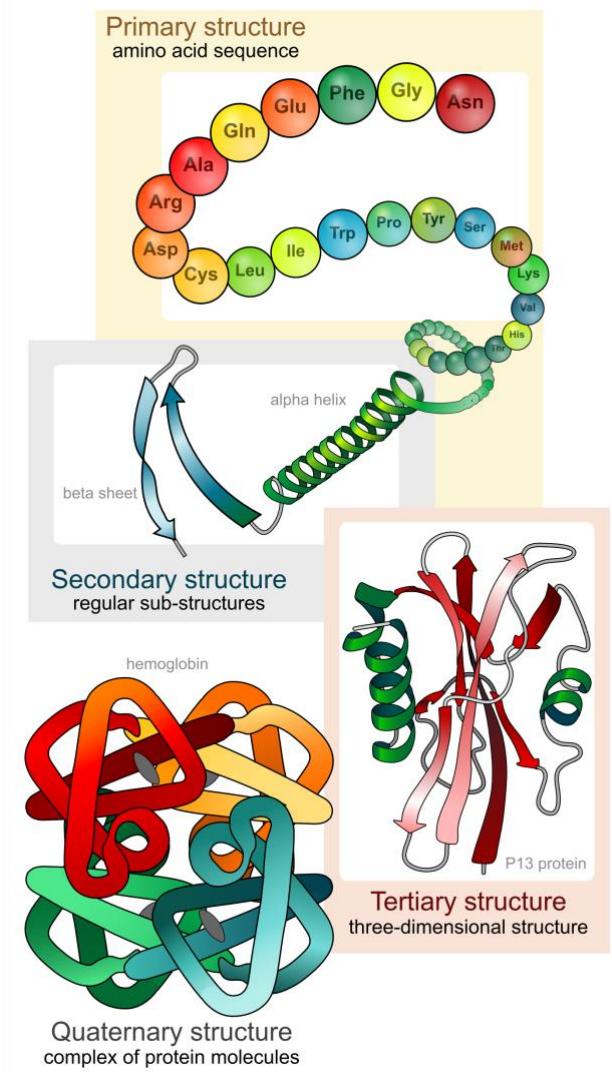


Figure 2.3 Four levels of protein structure—primary, secondary, tertiary, and quaternary. This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License (CC BY-SA 3.0) [25].

Proteins made of a single polypeptide chain fold to form their primary, secondary, and tertiary structures. In contrast, those composed of multiple polypeptide chains form their structures before assembling into a quaternary structure [13] (Figure 2.3). The folding process typically progresses through a series of intermediate states represented as a folding funnel, where from a large available set of unfolded conformations, the protein progresses to a limited number of intermediate states to finally reach a very small subset of native folded conformations [13]. In this model, proteins move from a high-energy, unfolded state toward a more stable, low-energy

conformation. Proper folding is essential for protein function, as misfolding can lead to dysfunctional proteins and is associated with diseases [15], [16].

While significant progress has been made in protein folding, accurately predicting how a protein will fold has long been a major challenge in structural biology, often referred to as the “protein folding problem” [24]. However, recent breakthroughs have significantly advanced the field. In 2024, David Baker was honoured with the Nobel Prize in Chemistry [5] for his innovative work in computational protein design. He shared the award with Demis Hassabis and John Jumper, who were recognised for developing *AlphaFold* [11], an AI model that has revolutionised the prediction of protein structures. These achievements represent a paradigm shift, marking a new era in which computational approaches and AI-driven technologies rapidly advance our ability to understand and manipulate protein structure and function.

2.2 Protein function

The function of a protein depends not only on its static three-dimensional structure but also on its dynamic behaviour, which plays a critical role in modulation. These dynamic changes, ranging from subtle local fluctuations to large conformational changes, enable proteins to interact with other molecules, respond to environmental changes, and transition between functional states [12].

For example, enzymes undergo conformational changes during catalysis, essential for substrate binding, chemical transformation, and product release [26], [27], [28]. These structural changes optimise the active site geometry for each step of the catalytic cycle [26], [27], [28]. Similarly, membrane receptors undergo structural adaptations upon ligand binding, triggering intracellular signalling cascades that facilitate cellular communication [26], [27], [28].

Various cellular processes exemplify the connection between dynamics and function. Adenylate kinase, a key enzyme in cellular energy homeostasis, alternates between open and closed conformations to facilitate substrate binding and product release [29]. Another classic example is haemoglobin, which switches between tense and relaxed states to modulate oxygen affinity, enabling efficient oxygen transport under different physiological conditions [30].

Understanding protein dynamics is, therefore, fundamental to elucidating protein function in living systems, as these movements directly connect structure to biological activity.

2.3 Protein dynamics

Studying protein dynamics is crucial for understanding the range of conformational changes proteins undergo as they perform their biological functions [31]. As proteins carry out their roles, they adopt distinct structural conformations known as functional states. These functional states are associated with specific biological roles or activities [32]. Dynamic movements between these states are directly linked to key functional events, such as substrate binding, signalling, and allosteric regulation [33], [34].

Unveiling proteins' dynamical properties is essential for identifying important functional states, the pathways connecting them, and the energetic barriers they need to overcome for transition between states [35].

Protein movements occur across different timescales [36]. Rapid, local fluctuations in atomic positions—such as bond vibrations—occur on the scale of femtoseconds (10^{-15} seconds) to picoseconds (10^{-12} seconds) [37]. Bond vibrations occur in femtoseconds, while side-chain rotations occur in picoseconds [37]. These small, fast movements are critical for maintaining structural flexibility at the atomic level and contribute to protein entropy [38]. On a slower timescale, nanosecond (10^{-9} seconds) to microsecond (10^{-6} seconds) conformational changes, such as loop motions and collective domain movements, play key roles in protein function, including substrate binding and enzyme catalysis [39], [40]. Surface loop motions typically occur in nanoseconds, while domain movements usually take microseconds [40]. Even larger, slower structural transitions—such as those involved in protein folding, allosteric regulation, or complex ligand binding events—occur over milliseconds (10^{-3} seconds) to seconds or longer [41], [42]. Protein folding can take anywhere from microseconds to seconds, depending on the protein size and complexity [43].

Both experimental techniques and computational methods have been developed to study protein dynamics across these timescales [44]. Classical MD simulations, in particular, can capture these motions at the atomic level, providing insight into how proteins function over time [45]. Modern MD simulations can now span the pico- to millisecond range, allowing researchers to simulate fast local fluctuations and slower, functionally relevant conformational changes [45]. These approaches offer a detailed view of protein behaviour that is difficult to capture solely through experimental methods.

2.3.1 Experimental techniques

Experimental techniques are key in studying protein dynamics, offering valuable structural information about proteins [46]. One of the most widely used methods is X-ray crystallography, which provides high-resolution snapshots of protein structures in their crystallised form [47]. This technique allows researchers to determine the arrangement of atoms within a protein, yielding detailed insights into its three-dimensional structure [47]. However, X-ray crystallography is limited in capturing the scale of dynamic movements that proteins undergo in their natural, solution-based environment, as crystallisation often locks proteins in a single, static conformation [47].

In contrast, nuclear magnetic resonance (NMR) spectroscopy provides insights into the flexibility and dynamics of proteins in solution [48]. NMR is beneficial for studying conformational changes and molecular motions over various timescales (picoseconds to seconds), offering a more dynamic view of protein behaviour than crystallography [48]. Various NMR experiments (relaxation measurements, residual dipolar couplings, chemical shift analysis) can probe different aspects of protein dynamics [49]. However, NMR has limitations, particularly when studying larger systems, as the signals become more challenging to resolve [50].

Cryo-electron microscopy (cryo-EM) has revolutionised structural biology, enabling the visualisation of large protein complexes in various conformational states at near-atomic resolution [1]. Cryo-EM can capture proteins in multiple conformational states by imaging many individual molecules, providing insights into conformational heterogeneity [1]. However, while cryo-EM excels at revealing different conformational states, it cannot directly observe the transitions between states or capture fast atomic-level dynamics [1].

Despite the strengths of these techniques, a critical limitation is that none of them can provide detailed information on protein dynamics at the atomistic level, nor can they fully capture the energetics of these dynamic processes [1]. While techniques like NMR or cryo-EM can offer glimpses of protein movements, they cannot reconstruct the complete free energy landscape required to fully understand transitions between functional states at the atomic level [46]. Therefore, experimental techniques often combine with computational methods, such as molecular dynamics simulations, to provide a more comprehensive view of protein dynamics and energetics [51].

2.3.2 Computational techniques

Computational techniques have been developed to study protein dynamics at various levels of resolution and timescales. Quantum Mechanics (QM) and Molecular Mechanics (MM) methods combine quantum mechanical calculations for specific regions (typically active sites) with classical molecular mechanics for the rest of the system [52]. This hybrid approach enables the study of chemical reactions and electronic interactions while maintaining computational feasibility [52]. QM/MM is particularly valuable for investigating enzyme mechanisms, chemical bond breaking/formation, and electronic effects in catalysis [52]. However, its high computational cost limits QM regions to typically 50-200 atoms, with simulation timescales rarely exceeding picoseconds [52].

Coarse-grained (CG) simulations reduce computational complexity by grouping atoms into larger units, such as representing amino acid residues with one or a few interaction sites [53]. Popular CG models include MARTINI, SIRAH, and elastic network models (ENMs) [54], [55], [56]. This simplification enables the simulation of larger systems (hundreds of proteins) and longer timescales (microseconds to milliseconds), allowing the study of large-scale conformational changes [53], [56]. However, CG models sacrifice atomic detail and may miss important local interactions [53]. Additionally, they tend to smooth the free energy landscape, which can obscure finer details of energy barriers and local minima important for specific biological processes [53].

Through stochastic sampling, Monte Carlo (MC) simulations explore protein conformational space [57]. Unlike MD, MC methods generate new configurations through random moves and accept or reject moves based on the Metropolis criterion [57]. These methods can easily overcome energy barriers and are particularly useful for equilibrium properties [58]. However, MC methods do not provide direct dynamic information and may struggle with complex collective motions [59].

MD simulations provide atomistic detail and temporal evolution by numerically solving Newton's equations of motion [60]. Modern MD simulations cover timescales from femtoseconds to microseconds (specialised hardware can reach milliseconds) and provide atomic-resolution trajectories [60]. They use physics-based force fields (e.g., AMBER [61], [62], CHARMM [63], OPLS [64], GROMOS [65], can incorporate various environmental conditions, and enable calculation of thermodynamic and kinetic properties [66]. A visual summary of this typical MD simulation pipeline is shown in Figure 2.4.

Enhanced sampling techniques such as Replica Exchange, Metadynamics, and Umbrella Sampling complement traditional MD by improving the exploration of conformational space and calculating free energy landscapes [67]. These methods help overcome the limitations of standard MD in sampling rare events and crossing high energy barriers, providing access to longer timescale phenomena and thermodynamic properties that might be otherwise inaccessible [67]. However, the price to pay for these methods is that it is generally challenging to define appropriate collective variables or temperature ranges for effective exchange, and reconstructing the free energy landscape *a posteriori* requires careful compensation for biases and potential artefacts introduced by these techniques [67].

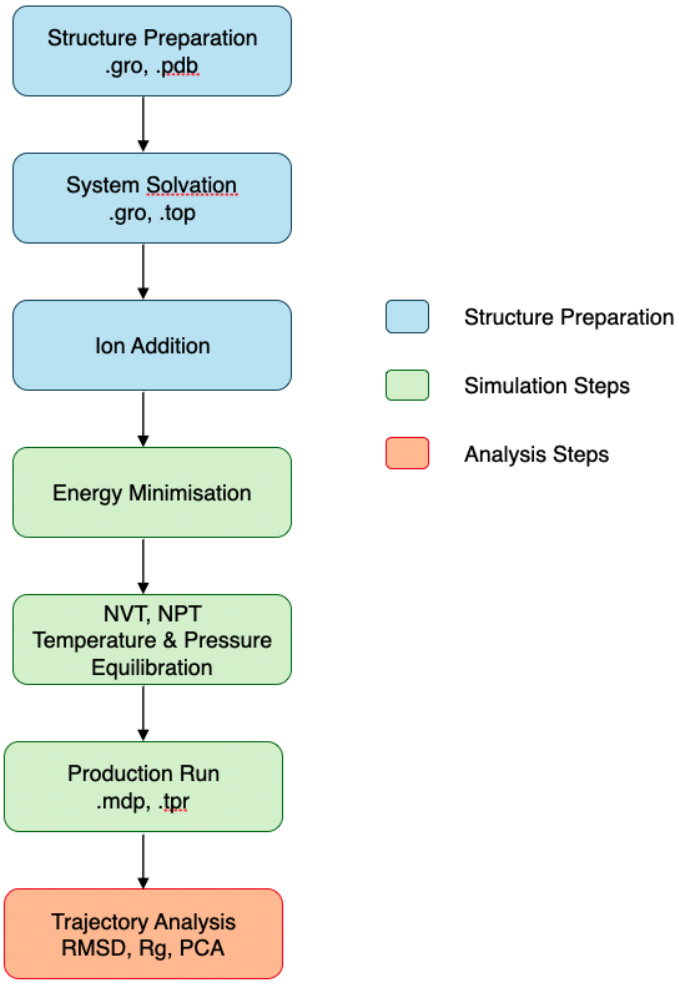


Figure 2.4 General workflow of a molecular dynamics simulation. The pipeline begins with system and topology preparation, solvation and ion addition, followed by energy minimisation, thermal (NVT) and pressure (NPT) equilibration steps, and a production run. The resulting trajectory is then analysed to extract dynamic and structural properties. The workflow shown is general but is presented here with reference to the file formats used by GROMACS, which is the selected simulation engine for this research.

2.3.3 Molecular Dynamic Simulations (MD)

MD simulations are a widely used computational technique for studying the time-dependent behaviour of molecular systems. They simulate the interactions between particles, typically atoms or molecules, and provide detailed insight into proteins' dynamics and conformational changes over time.

In MD simulations, the motion of particles is determined by solving Newton's second law of motion, which states that the force F_i acting on a particle i is equal to the product of its mass m_i and its acceleration α_i [3]:

$$F_i = m_i \times \alpha_i = m_i \times \frac{d^2 r_i(t)}{dt^2} \tag{2.1}$$

Where r_i represents the position of the particle at the time t. The force F_i is defined as the negative gradient of the potential energy function U in relation to the position of the particle [3]:

$$F_i = -\nabla_i U(r_1, r_2, \dots r_N) \tag{2.2}$$

This potential energy function defines all interactions between atoms in the system [3]. The classical mechanics approximation enables the simulation of large molecular systems over biologically relevant timescales by solving these equations for all particles in the system [68].

MD captures both the local, fast movements (such as bond vibrations) and larger-scale motions (such as protein folding or conformational transitions) by progressively adjusting the positions and velocities of atoms over very short timesteps (typically on the order of femtoseconds, 10-15 seconds) [68]. The accuracy of an MD simulation depends on the underlying force field used to define these potential energy functions and their parameters, the numerical integration scheme used to solve Newton's equations of motion, and the applied thermodynamic conditions such as temperature, pressure, and boundary conditions [68].

2.3.3.1 Force fields

Force fields provide a mathematical framework to describe the potential energy of a molecular system [69]. The potential energy function $U(r)$ includes contributions from bonded and non-bonded interactions [69].

$$U(r) = u_{bonded} + u_{non-bonded} \tag{2.3}$$

This can be expanded as:

$$U(r) = \sum_{bonds} k_b\,(r - r_i)^2 + \sum_{angles} k_\theta\,(\theta - \theta_0)^2 + \sum_{torsions} V_n\,(1 + \cos\,(n\varphi - \gamma)] + \sum_{pairs} 4\varepsilon\left[\left(\frac{\sigma}{r_{ij}}\right)^{12} - \left(\frac{\sigma}{r_{ij}}\right)^{6}\right] + \sum_{pairs} \frac{q_i q_j}{4\pi\varepsilon_0 r_{ij}} \tag{2.4}$$

Bonded interactions: In a molecular system, they describe how atoms connected by chemical bonds interact. These include bond stretching, angle bending, and torsional (dihedral) rotations. Bond stretching occurs when the distance between two bonded atoms changes from its equilibrium value. This is typically modelled using Hooke's Law as a harmonic potential:

$$U_{bond}\,(r) = \sum_{bonds} k_b\,(r - r_0)^2 \tag{2.5}$$

The bond length is represented by r, with r_0 denoting the equilibrium bond length and k_b being the force constant that describes the stiffness of the bond. Angle bending, which involves changes in the bond angles between three atoms, is modelled similarly to bond stretching.

$$U_{angle}\,(\theta) = \sum_{angles} k_\theta\,(\theta - \theta_0)^2 \tag{2.6}$$

Where θ is the bond angle, θ_0 is the equilibrium bond angle, and k_θ is the corresponding force constant. Torsional (dihedral) interactions arise when atoms rotate around a bond. The torsional potential energy is periodic, accounting for rotations around bonds:

$$U_{torsion}\,(\varphi) = \sum_{torsions} V_n\,(1 + \cos\,(n\varphi - \gamma)] \tag{2.7}$$

Where φ is the torsional angle, V_n is the barrier height, n is the periodicity, and γ is the phase shift.

Non-bonded interactions: In a molecular system, these refer to forces between atoms that are not directly bonded but still influence each other through spatial proximity. Non-bonded interactions comprise van der Waals forces and electrostatic interactions.

Van der Waals forces are short-range attractive or repulsive forces between atoms. These are represented by the Lennard-Jones potential, which includes a repulsive term to prevent atoms from collapsing and an attractive term at longer distances:

$$U_{vdW}\ (r_{ij}) = 4\varepsilon\left[\left(\frac{\sigma}{r_{ij}}\right)^{12} - \left(\frac{\sigma}{r_{ij}}\right)^{6}\right] \tag{2.8}$$

Here, r_{ij} represents the distance between atoms i and j, ε is the depth of the potential (defining the strength of the interaction), and σ is the distance at which the potential is zero.

Electrostatic interactions describe the forces between charged particles and are calculated using Coulomb's law:

$$U_{elec}\ (r_{ij}) = \frac{q_i q_j}{4\pi\varepsilon_0 r_{ij}} \tag{2.9}$$

Here, q_i and q_j represent the charges on atoms i and j, respectively, while r_{ij} denotes the distance between them. This term is significant in defining how molecules interact, especially in biological systems where long-range electrostatic interactions are important.

Force field in use: AMBER ff99SB*-ILDN

Several force fields have been developed for MD simulations, each optimised for different molecular systems. Among the most widely used are AMBER [61], CHARMM [63] and GROMOS [65]. The AMBER force field used for this research is the AMBER ff99SB*-ILDN [70] variant.

The original AMBER ff99 [71] force field was designed to accurately describe the behaviour of proteins, nucleic acids and other biomolecules by carefully parameterising the bonded and non-bonded interactions. However, as computational techniques and experimental data evolved, it became clear that specific dihedral parameters required refinement, particularly in describing the flexibility of the protein backbone [71].

To address these issues, the AMBER ff99SB force field [71] was introduced, improving the accuracy of protein backbone dihedrals by modifying the torsion potentials for the crucial φ (phi) and ψ (psi) angles. Further modifications led to the development of AMBER ff99SB*-ILDN [70], which incorporates both improved backbone torsion refinements and ILDN

corrections for side chain rotamer distributions of isoleucine, leucine, aspartate, and asparagine residues.

The AMBER ff99SB*-ILDN force field was selected for this research because it provides high accuracy in modelling backbone dynamics and side chain behaviour, which is particularly important for protein folding and protein-ligand interactions studies [70]. Its wide adoption and extensive examples of published applications were additional reasons for adopting it.

When selecting force field parameters, several key factors must be considered: the chemical environment (aqueous solution, membrane interface, or crystalline state), specific interaction requirements (such as metal ions or post-translational modifications), and simulation conditions (temperature, pressure, pH, salt concentration) [72]. Parameters should be validated against experimental data, including structural properties from X-ray crystallography or NMR and dynamic properties from spectroscopic measurements. For protein simulations, ff99SB*-ILDN parameters have been extensively validated across these conditions, particularly for standard amino acids in physiological environments [73].

2.3.3.2 Integration methods

Newton's equation of motion in MD simulations determines how the system evolves [74]. Particle positions and velocities are updated incrementally, with the selected integration method affecting both the accuracy and stability of the results [74].

Several integration methods are commonly applied in MD simulations, such as the Verlet algorithm [75], the Velocity Verlet algorithm [76], and the Leap-frog method [77], [78]. The Verlet algorithm updates positions based on previous positions and forces, though it does not calculate velocities directly, which can sometimes be a limitation. By contrast, the Velocity Verlet method updates positions and velocities at each timestep. In this study, the Leap-frog integration method, a variation of the Verlet algorithm, was selected for its combination of accuracy and computational efficiency.

The Leap-frog method updates velocities and positions at alternating timesteps, effectively "leaping" over itself. This approach offers a time-reversible solution that remains computationally stable, even when using larger timesteps, such as 1-2 femtoseconds. Specifically, the method performs the following:

1. Updates velocities first at half-time steps:

$$u_i\left(t+\frac{\Delta t}{2}\right)=u(t)+\alpha_i(t)\frac{\Delta t}{2} \tag{2.10}$$

Where u_i is the velocity of particle i, α_i is the acceleration derived by the forces acting on the particle, and Δt is the timestep.

2. Positions are updated using the velocities from step 1:

$$r_i(\Delta+\Delta t)=r_i(t)+u_i\left(t+\frac{\Delta t}{2}\right)\Delta t \tag{2.11}$$

Where r_i represents the position of particle i.

These equations are applied iteratively throughout the simulation, with the choice of timestep being crucial for accuracy.

In MD simulations, the rapid vibrations of hydrogen bonds typically occur on the scale of 10 femtoseconds. A very small timestep, around 0.5 femtoseconds, is often required to capture these motions accurately. However, constraining these bond lengths to their equilibrium values can increase the timestep without losing accuracy. Algorithms like SHAKE [79], RESPA [80] and LINCS [81] are commonly used to apply these constraints. In this thesis, the LINCS algorithm was chosen to manage bond constraints because of its effectiveness in large biomolecular systems, as well as its speed and reliability. Compared to SHAKE, LINCS performs faster, especially when multiple bonds need to be constrained at once, and it is the default algorithm for constraints in GROMACS [2] simulations.

The LINCS algorithm determines the positions of atoms in constrained bonds following each integration step. It then iteratively adjusts these positions using a matrix-based method to ensure that bond length constraints are satisfied throughout the simulation.

2.3.3.3 Thermodynamics conditions

In MD simulations, thermodynamic conditions regulate the system's temperature, pressure, and volume, influencing its time evolution. Thermodynamic ensembles are used to simulate specific physical boundary conditions. The canonical ensemble (NVT) maintains a fixed number of particles (N), constant volume (V), and steady temperature (T), with thermostats employed to control the system's temperature. This ensemble is typically applied when precise temperature control is necessary, but changes in volume are not required. Common

thermostats include the Langevin thermostat [82], Berendsen [83], and V-rescale [84] algorithms.

The isothermal-isobaric ensemble (NPT) maintains a fixed number of particles (N), constant pressure (P), and steady temperature (T) by using a thermostat and a barostat. The NPT ensemble is often used in biological simulations as it reflects physiological conditions, providing a more realistic environment for studying protein dynamics and interactions. Pressure control is achieved through barostats such as Berendsen [83] or Parrinello-Rahman [85]. In this research, both NVT and NPT ensembles were employed sequentially. Initial system equilibration used the Berendsen thermostat to achieve target temperature efficiency rapidly. The production simulations utilised the V-rescale thermostat [84] for accurate sampling while maintaining system stability. The pressure was controlled using the Parrinello-Rahman [85] barostat.

2.3.3.4 Solvation

In MD simulations, selecting an appropriate solvation model is essential for accurately capturing the interactions between biomolecules and their solvent. Describing intermolecular interactions when mediated or shielded by the solvent is also critical. Two main approaches have been proposed over the years: explicit and implicit water models.

Explicit solvation models represent each water molecule individually with its coordinates and degrees of freedom, providing detailed and realistic solvent-solute interactions. Standard explicit water models include TIP3P [86], a three-site model with a rigid structure; TIP4P [86], a four-site model with improved electrostatic representation; and SPC [87], the Simple Point Charge model.

This research employed the TIP3P water model due to its computational efficiency while accurately reproducing water's physical properties. The model was also selected because it is the reference model used for AMBER force field [88] parametrisation. It is a well-established and extensively validated model for simulating globular proteins.

Implicit solvation models treat water as a continuous medium characterised by a dielectric constant, offering reduced computational demands and faster simulation times. These models enable efficient free energy calculations using Generalized Born [89] and Poisson-Boltzmann [90]. However, implicit models cannot capture detailed solute-solvent interactions, such as specific hydrogen bonding patterns or local solvent structure effects.

2.3.3.5 Periodic boundary conditions

In MD simulations, the behaviour of biomolecules is typically studied in a finite simulation box [91]. However, real biological systems exist in infinite environments where molecules can move freely without encountering artificial boundaries [91]. To mimic this infinite behaviour, periodic boundary conditions (PBC) are employed [91]. PBC eliminate artificial boundaries by creating a system where molecules leaving one side of the box re-enter from the opposite side, ensuring particles exist in a continuous environment [91].

The shape of the simulation box significantly impacts PBC implementation. The box must be large enough to prevent interactions between a molecule and its periodic image, which can create artificial surface effects [91]. A minimum distance of approximately 10 Å is typically maintained between the solute and box boundary to prevent these self-interactions [86]. While cubic boxes offer simplicity in implementation, truncated octahedral geometries can be more efficient for spherical solutes by reducing the required solvent volume and, thus, computational cost [92]. For this research, a cubic box was selected based on system requirements and setup compatibility.

To ensure realistic simulations under PBC, accurate treatment of non-bonded interactions is critical. Efficient calculation of these interactions is essential for simulating large systems over biologically relevant timescales.

In MD simulations, non-bonded interactions, including van der Waals and electrostatic forces, are typically evaluated within a defined cut-off distance [93]. Interactions within this cut-off (typically around 10–12 Å in biomolecular simulations) are computed explicitly using the Lennard-Jones potential for van der Waals interactions and Coulomb's law for electrostatics [93]. However, long-range electrostatic interactions beyond the cut-off are not ignored; instead, they are efficiently approximated using the PME method [94], which decomposes electrostatics into short-range (real space) and long-range (reciprocal space) components. PME ensures that periodic boundary effects are handled accurately and that electrostatic interactions are correctly reproduced across the infinite lattice of periodic images.

In this study, a cut-off distance of 10 Å was used for both van der Waals and short-range Coulombic interactions, in accordance with the default settings recommended for the AMBER ff99SB*-ILDN force field [70]. This cut-off value balances computational efficiency with

accuracy, ensuring the reliability of the interactions simulated while leveraging PME to account for long-range electrostatics.

2.3.3.6 Particle Mesh Ewald (PME)

Long-range electrostatic interactions in MD simulation systems with PBC are computed using the PME method [94]. PME can decompose the electrostatic interactions into short-range and long-range components [95]. The short-range interactions are calculated directly in real space within a cutoff distance, while the long-range interactions are processed in reciprocal space using Fourier transforms [95].

The electrostatic potential $\varphi(r_i)$ for a particle i is influenced by the positions and charges of all other particles j in the system and their periodic images [96]. This can be expressed as:

$$\varphi(r_i) = \sum_{k} \sum_{j=1}^{N} q_i \left(\frac{1}{|r_i - r_j + kL|} \right) \tag{2.12}$$

Here, k represents the vectors corresponding to the periodic images, and $r_i - r_j$ denotes the distance between particles i and j [96].

This thesis used PME to ensure accurate and efficient computation of electrostatic interactions, enabling realistic simulations of biomolecular dynamics in periodic boundary conditions.

2.3.4 Computational analysis of protein dynamics through MD

Understanding protein dynamics is essential for studying biomolecular function, which often relies on the flexibility and motion of atomic structures. MD simulations offer a robust computational approach to examining these dynamics at the atomistic level, providing detailed insights into the conformational landscape that proteins explore over time [45]. MD simulations capture protein dynamics as an ensemble of atomistic configurations, or "conformations", where each conformation represents a snapshot of the protein structure, collectively providing a probabilistic view of its dynamic states [97].

These conformational ensembles can calculate several geometric properties to interpret protein dynamics [98]. The root-mean-square deviation (RMSD) measures structural similarity to a reference structure, while the radius of gyration (Rg) indicates the protein’s spatial

distribution and compactness [93]. The centre of mass (COM) tracks overall protein position and movement, and the root-mean-square fluctuation (RMSF) quantifies the local flexibility of specific residues [93]. Interatomic distances provide a means to monitor specific interactions and conformational changes [93].

Statistical measures can be employed to analyse distributions from different simulation ensembles to determine the significance of changes in these properties, particularly when comparing different conditions or mutations [99]. In this work, the following statistical methods were used: Kullback-Leibler divergence [100], Bhattacharyya distance [101], and Pearson correlation [102]. They enable quantitative comparisons between property distributions and a practical way to estimate distances between distributions and, in turn, to estimate distances in dynamical behaviour between simulations [99].

2.3.5 Enhanced sampling techniques

Sampling the conformational space when using MD simulations is challenging due to the high dimensionality of protein systems and the substantial energy barriers in the energy landscape [103]. Proteins often become trapped in local minima, making it difficult to efficiently sample all relevant conformational states. Standard MD simulations may take a long time to explore the entire conformational landscape, especially for transitions that occur on slow timescales, such as protein folding, binding, or conformational changes [103]. To address these limitations, several enhanced sampling techniques were developed over the years to accelerate the exploration of conformational space and allow simulations to overcome these energy barriers [103].

Among the most widely used enhanced sampling techniques are Umbrella Sampling [103], Replica Exchange Molecular Dynamics (REMD) [103], and Metadynamics [103]. These methods bias the system to promote exploration of less frequently sampled regions of the energy landscape.

Umbrella sampling employs a series of biasing potentials, known as "windows", along a reaction coordinate to improve sampling over energy barriers [104]. REMD runs multiple system replicas at different temperatures, enabling exchanges that improve sampling efficiency by allowing higher-temperature replicas to overcome energy barriers [105]. Metadynamics introduces a time-dependent bias to a set of chosen degrees of freedom, known as collective variables (CVs) [106]. These CVs represent key structural changes within

the system and act as geometrical measures describing important protein structure changes. Common CVs include RMSD, Rg or distances between specific atoms [107].

2.3.6 MD simulations data in machine learning

The integration of MD data into ML and DL pipelines offers significant opportunities but presents several challenges [4], [108]. MD simulations generate vast amounts of high-dimensional, complex, and noisy data that capture the detailed temporal evolution of molecular systems [4]. This data often includes trajectories of atomic positions and velocities, which are difficult to directly utilise in ML models as they are not in the tabular or tensor format commonly accepted by ML software [108]. As a result, the first obstacle is transforming this raw data into a form that can be processed by computational models, especially when considering the scale of simulations, which can produce terabytes of data over relatively short periods.

ML/DL models can “learn” complex patterns in data [109], [110]. Identifying patterns within MD datasets makes it possible to understand biomolecular dynamics in greater detail, predict molecular interactions, or even discover new conformational states that might be difficult to detect through traditional methods [108]. Additionally, applying ML and DL to MD data has the potential to accelerate the analysis of protein folding, ligand-binding events, and other critical molecular processes, making the simulations more predictive and informative [6], [7], [111]. However, due to the scale and complexity of MD outputs, there remains a significant gap in the field: efficient methods to preprocess, simplify, and integrate this data into ML models [4]. Addressing this gap will significantly enhance our ability to leverage MD simulations in computational biology.

2.4 Protein design, engineering, and redesign

Protein design is a well-established and rapidly advancing field, focusing on generating new proteins or modifying existing ones to achieve desired structural, functional, or dynamic properties. The transformative contributions of computational methods in this field were recognised with the 2024 Nobel Prize in Chemistry [5]. By engineering protein sequences at the molecular level, researchers can modify a protein’s function, stability and interactions within biological systems. Generally, these modifications have primarily focused on optimising static properties, such as stability and binding affinity, while overlooking dynamic behaviours, which are equally essential for biological function [112]. Dynamic transitions between different structural states are crucial for protein function, such as ligand binding, enzyme catalysis, and allostery [113], [114].

Protein design is generally classified into template-based and *de novo* approaches [115]. The template-based design uses known protein structures as scaffolds to guide targeted modifications, introducing mutations at specific residues to improve stability, binding affinity, or catalytic activity [115]. In contrast, *de novo* design generates new protein sequences capable of novel folds and functions [115].

While both template-based and *de novo* design have facilitated significant advancements, they traditionally rely on single-state models that often do not account for the flexibility required for complex biological tasks, including ligand binding and allosteric regulation. Recent developments in multistate design (MSD) address this by incorporating structural flexibility, enabling the design of proteins that transition between functional states, thus allowing for more accurate predictions of binding specificity and conformational changes [116], [117], [118]. MSD is especially beneficial for designing proteins intended for conformational shifts, such as molecular switches or regulatory enzymes, where transitions between states are crucial for function [119].

Furthermore, MSD methods such as those implemented in *iCFN* [116] and *POMPd* [117] allow for optimisation across multiple conformational states, demonstrating improved stability and functionality for complex systems where single-state models fall short. To enhance flexibility, MSD with backbone ensembles utilises multiple conformational templates, which not only refines the accuracy of stability predictions but also allows proteins to occupy stable positions across energy landscapes, lowering the barriers for transitions [112], [120]. Despite this, MSD approaches focus on managing multiple stable conformations rather than fully exploring dynamic transitions across all functional states [121]. However, recent advancements, such as ML-guided sequence-structure prediction in tools like *RoseTTAFold*, are beginning to bridge these gaps, enhancing the potential for accurate, flexible protein design [118].

2.4.1 Protein engineering, redesign, and mutation engineering

Related to protein design, protein engineering aims to enhance specific protein attributes—such as stability, activity, or specificity—by systematically modifying natural sequences, often targeted mutations [122]. One prominent approach in protein engineering is mutation engineering, where selected mutations are introduced to fine-tune properties like stability, binding affinity, or catalytic efficiency [122]. These modifications optimise protein performance in specific conditions, making protein engineering invaluable in therapeutic development and industrial biotechnology [122]. However, while mutation engineering is effective for stabilising

proteins or enhancing their interactions, it frequently relies on computational models that focus on static structures [123]. While robust in assessing stability changes, many of these tools do not fully capture dynamic interactions within the protein, limiting their ability to address the conformational flexibility required for complex biological functions [124].

Protein redesign builds on these principles by targeting specific attributes—such as binding affinity or thermal stability—while maintaining the protein's core structure [125]. This approach is beneficial for applications where proteins need to perform specific functions while maintaining their structural integrity [125]. Although established methods for redesign have led to significant gains in optimising static properties, they often overlook dynamic behaviours, which are essential for proteins that operate through conformational changes across multiple states [125]. Consequently, there remains a need for more comprehensive frameworks that can account for dynamic flexibility in protein redesign, particularly for applications that require a range of functional states [125].

2.4.2 Computational tools for protein design

Several tools to model sequence-structure relationships, predict mutation effects, and optimise protein properties have been developed in computational protein design (CPD) [126]. These tools vary in approaches and capabilities, from energy-based calculations to ML methods [127]. At the core of CPD is the *Rosetta* software [128] suite and its Python interface, *PyRosetta* [9], which provide comprehensive platforms for protein structure prediction, design, and analysis. *Rosetta* employs energy functions and sampling methods to explore conformational space, evaluate mutation impacts, and perform *de novo* design [128]. *PyRosetta* extends these capabilities through accessible Python scripting, enabling researchers to implement custom protocols for mutation engineering, stability assessment, and backbone flexibility optimisation [9].

FoldX [129] and *AlphaMissense* [130] represent two complementary approaches for mutation impact analysis. *AlphaFold*, a revolutionary DL-based tool that predicts high-resolution protein structures from amino acid sequences, has transformed structural biology [11]. Building on similar deep learning advances, *AlphaMissense* predicts the functional consequences of mutations by analysing sequence conservation patterns and structural context. *FoldX* employs empirical force fields to calculate changes in free energy (ΔΔG) upon mutation, providing quantitative predictions of protein stability changes [131], [10]. While *FoldX* focuses on thermodynamic stability [131], *AlphaMissense* provides broader insights into potential

functional impacts, including effects on protein-protein interactions and catalytic activity [130]. Together, these tools enable a comprehensive assessment of stability and functional changes in protein design.

Several tools specialise in predicting mutation-induced changes in protein flexibility. *DynaMut* [14] and *DUET* [132] employ ML approaches combined with molecular modelling to assess how mutations affect local and global protein effects. While these tools provide rapid assessments of flexibility changes, they primarily focus on equilibrium stability rather than tracking conformational transitions between states [14], [132]. Complementary to these approaches, elastic network models like the Elastic Network Contact Model [133] (ENCoM) use normal mode analysis (NMA) to simulate protein motions. ENCoM specifically examines how mutations influence protein vibrational modes and flexibility patterns, offering insights into potential changes in protein dynamics and stability at a more mechanistic level [133].

ML has transformed CPD by enabling predictions of protein dynamics and mutation effects [134]. Early ML approaches, including neural networks (NN) and support vector machines (SVMs), successfully classified mutation impacts using sequence and structural features [135], [136]. More advanced deep learning techniques, particularly convolutional neural networks (CNNs) for spatial pattern recognition and recurrent neural networks (RNNs) for analysing and learning time-dependent information from simulations, have expanded the scope of analysis [137], [138], [139]. These methods can identify subtle patterns in protein conformational dynamics that traditional physics-based methods might miss [139]. Despite their potential, ML methods remain constrained by the limited availability of high-quality, experimentally validated training data, particularly for proteins with multiple conformational states [140]. This scarcity of comprehensive dynamic data can affect prediction accuracy, particularly when identifying and classifying distinct conformational states in proteins that undergo significant structural changes [140].

ProteinMPNN [141] represents a significant advance in MPD through its message-passing neural network architecture. The tool specialises in designing protein sequences that maintain stability across multiple conformational states, a crucial requirement for proteins that undergo functional transitions [141]. Unlike traditional single-state design approaches, *ProteinMPNN* can simultaneously optimise sequences for multiple backbone conformations, considering the complex energetic landscape required for conformational flexibility [141]. This capability has enabled the design of proteins with engineered dynamic properties, including allosteric regulators and molecular switches [119].

Two significant advances have revolutionised structure prediction: *AlphaFold3* [142] and *RoseTTAFold* [143]. *AlphaFold3*, building on its predecessor *AlphaFold2* [11], achieves high accuracy in protein structure prediction through its advanced attention-based DL architecture [142]. *RoseTTAFold* introduced a complementary approach with its three-track neural network that simultaneously processes sequence, structure, and interface information, offering insights into protein flexibility that complement *AlphaFold3*'s predictions [143].

These tools have become foundational for protein design, with recent adaptations employing advanced techniques. Activation maximisation has been used for sequence optimisation, enabling the identification of sequences that fold into specific target structures [144]. Inpainting approaches have been applied for structural refinement, allowing for the completion of partial protein structures or the design of sequences for predefined motifs [143]. Most recently, denoising diffusion models, as implemented in *RFdiffusion*, have generated novel protein structures and sequences with high success rates in designing functional proteins, including protein-protein interfaces, catalytic sites, and complex assemblies [145]. Together, these advancements represent a significant leap forward in the automated design of functional proteins.

Despite their remarkable success in structure prediction, both tools face limitations in capturing protein dynamics [146], [147]. They primarily generate static snapshots and cannot fully account for conformational changes induced by mutations or environmental factors [147]. Understanding protein dynamics still requires integration with other computational tools or experimental validation methods to capture the full range of protein flexibility [148].

2.4.3 PyRosetta for mutation engineering

PyRosetta represents a powerful platform for mutation engineering, providing a Python-based interface to the comprehensive *Rosetta* molecular modelling suite [9]. It combines *Rosetta*'s energy functions and sampling algorithms with the flexibility of Python scripting, enabling systematic exploration of mutation effects on proteins [9]. The tool employs a hierarchical approach to mutation analysis: first, it utilises MC sampling, which incorporates the concept of fragments to efficiently explore conformational space, followed by energy minimisation to refine local geometry [9]. *PyRosetta*'s MC exploration minimises the search space dimensionality by utilising predefined structural fragments, ensuring computational efficiency while maintaining biologically relevant conformations [9]. Additionally, *PyRosetta*'s implementation of the *Rosetta* energy function, which includes terms for various physical

interactions, allows for a detailed assessment of how mutations influence local and global protein properties [128].

The Rosetta Energy Function (REF) is fundamental to *PyRosetta*'s mutation engineering capabilities. It combines physics-based and knowledge-based terms to evaluate protein energetics [149]. This function integrates multiple components: van der Waals interactions, hydrogen bonding, solvation effects, and torsional potentials [149]. *PyRosetta* calculates a total energy score for each proposed mutation, where lower values indicate more favourable conformations [149]. This scoring system enables quantitative assessment of mutation-induced stability changes [149].

PyRosetta implements MC sampling to explore the conformational landscape of mutated proteins. The algorithm systematically evaluates amino acid substitutions at targeted positions, generating new accepted or rejected conformations based on the Metropolis criterion [150]. This probabilistic approach favours energetically favourable states while maintaining the ability to escape local energy minima, ensuring thorough exploration of possible conformations [9] [128].

Side-chain optimisation utilises rotamer libraries to sample energetically favourable conformations for mutated residues [128]. *PyRosetta* evaluates each potential rotamer based on its interactions with neighbouring residues, optimising local geometry and overall structural stability [128]. This systematic sampling ensures accurate modelling of side-chain positions, which is critical for predicting mutation effects on protein structure and function [128].

The final refinement stage employs gradient-based energy minimisation to optimise the mutated structure [128]. This process fine-tunes bond angles, torsions, and atomic positions to resolve steric conflicts and optimise interactions around the mutation site [128]. Detailed refinement is crucial for mutations affecting functional sites, such as binding interfaces or catalytic centres [149]. Together, these computational steps provide a comprehensive framework for predicting and analysing protein mutations' structural and functional consequences.

2.5 Challenges in redesigning protein dynamics

The redesign of protein dynamics presents unique challenges due to the complexity of protein motion and conformational flexibility [151]. While traditional protein engineering has successfully optimised static properties such as thermal stability or binding affinity, engineering

dynamic behaviours requires understanding and controlling transitions between multiple conformational states [151]. These transitions are essential for protein functions, including allosteric regulation, signal transduction, and molecular recognition [151]. The challenge is amplified by the need to effectively explore and capture the information in the conformational landscape [152]. Significant computational resources are required to capture local fluctuations and large-scale conformational changes occurring over biologically relevant timescales [153].

Despite significant advances in computational protein design, current tools remain predominantly focused on static structure prediction rather than dynamic behaviour [154]. This limitation stems from technical and practical constraints: simulating protein dynamics requires extensive computational resources to capture motions across biologically relevant timescales [153]. Recent studies highlight the critical need for new computational frameworks to integrate dynamic behaviour into the protein design process, balancing structural stability and functional flexibility [112].

2.6 Summary

This chapter has presented an integrated view of proteins, from structural fundamentals to the frontier challenges of dynamic protein design. The chapter examined protein structure, folding mechanisms, and conformational dynamics and established the essential relationship between molecular structure, protein dynamics and biological function. The detailed exploration of MD simulations highlighted two challenges: managing and analysing the vast amounts of high-dimensional data generated from MD simulations and the difficulty in transforming this complex MD trajectory data into formats suitable for ML and DL applications.

The review of protein design methodologies revealed a fundamental gap in the field: the lack of automated computational tools that can directly redesign protein dynamics, as current approaches primarily focus on static structural properties rather than dynamic behaviour. Although recent advances in computational methods and AI-based approaches have enhanced our understanding of protein structure and function, these fundamental challenges continue to limit our ability to fully leverage computational approaches for dynamic protein engineering. Integrating enhanced sampling techniques with ML offers promising directions for addressing these gaps, enabling more sophisticated protein engineering applications that can account for and control dynamic behaviour.

The goal of this thesis is to address these gaps by developing novel computational approaches that can effectively handle MD simulation data, facilitate its integration with ML methods, and provide automated tools for redesigning protein dynamics and, therefore, modifying protein function.

3 Methods

In this chapter, the methods used in this research are discussed in detail. Section 3.1 introduces ADK as the case study, explaining its biological significance and dynamic behaviour. Section 3.2 outlines the generation of MD simulation data, covering the setup of unbiased simulations and the specific protocols and software used, such as AMBER and GROMACS, along with a dedicated section on the MD data generated for each toolkit (MDSS, MDAM, and MDAP). Section 3.3 describes the data analysis and validation processes, focusing on key geometric properties, techniques used to confirm the simulations' stability and principal component analysis. Sections 3.4, 3.5 and 3.6 discuss the development of MDSS, MDAM and MDAP by integrating components from *MDAnalysis*, *PyRosetta* and gmxapi. Section 3.7 describes the deployment of the toolkits on ARCHER2, while 3.8 covers using a Docker container to ensure reproducibility across different computing environments. Section 3.9 highlights the use of Poetry for managing dependencies and packaging the toolkits.

3.1 Case study: adenylate kinase (ADK)

ADK is a small enzyme of 214 amino acids and is essential for regulating energy homeostasis within cells [155], [156]. It facilitates the reversible transfer of a phosphate group between two adenosine diphosphate (ADP) molecules, producing adenosine triphosphate (ATP) and adenosine monophosphate (AMP) [155]. The enzyme's activity is important for maintaining nucleotide balance in the cell, where AMP acts as a key regulator in cellular metabolism and signals energy deficiency [157]. Small fluctuations in the ATP-to-ADP ratio can lead to significant changes in AMP concentration, allowing ADK to function as a sensitive metabolic sensor [157]. This mechanism operates within and between cells through connected enzymes [157]. The coordination of energy transfer and nucleotide signalling by ADK, mainly via its isoform network, is crucial for processes such as mitochondrial energy, muscle contraction, and cell motility, among others [155]. Furthermore, mutations in ADK have been associated with several diseases, including primary ciliary dyskinesia and reticular dysgenesis, further highlighting the importance of the enzyme in human health [155].

The structural dynamics of ADK are defined by two key conformational states – open and closed – making it an excellent model for testing computational tools when studying examples of protein dynamics [158]. ADK consists of three main domains: CORE (res 1-29, 68-117, 161-214), LID (res 118-160) and NMP (res 30-67), each of which plays a specific role in catalysis (Figure 3.1) [159]. During the catalytic cycle, the CORE domain remains relatively stable, while

the LID and NMP domains sometimes undergo significant conformational changes to facilitate substrate binding and product release [158]. These large-scale domain movements are essential to the functionality of ADK as they allow the enzyme to switch between open and closed states during the nucleotide exchange reaction [158].

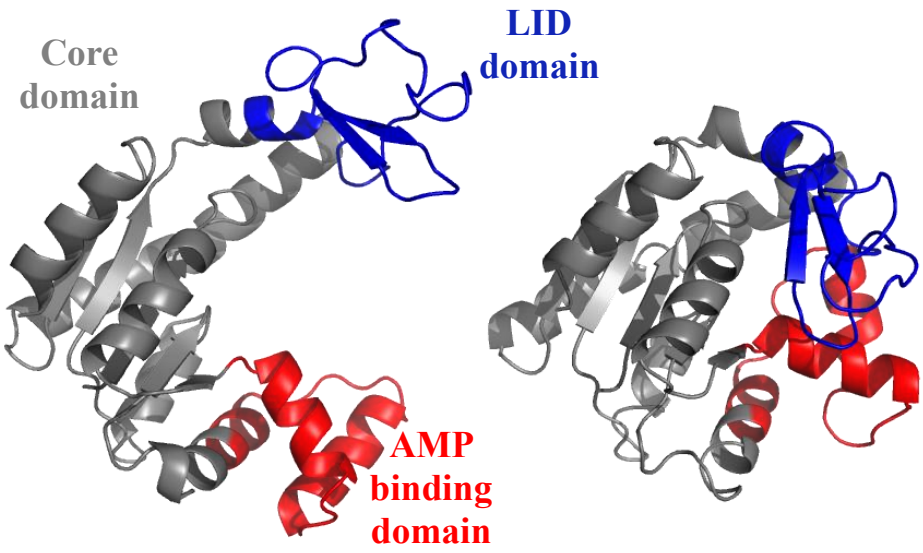


Figure 3.1 Structural representation of the open and closed conformations of ADK. The structure is showing the core domain (grey), AMP binding domain (red), and LID domain (blue). Structural figures were visualised and rendered using PyMOL [160].

Due to its extensive research history and availability of experimental data, ADK, with its clearly defined dynamics (i.e. open and closed conformational states) (Figure 3.1), provides a sound system for testing and validation of methods for the analysis of conformational dynamics from molecular simulations and, in the current work, of ML models for prediction of conformational states [161], [162], [163], [164], [165], [166], [158]. Therefore, this system was chosen to test and validate the three tools that were designed and implemented in this research (MDSS, MDAM, MDAP). Simulations were performed only on the unbound state to provide a more manageable framework for interpreting sampling and mutational effects on the intrinsic dynamics of the system. Details on the ability of apo ADK to sample closed conformational states are discussed in the context of mutations that affect (or induce) a pre-existing equilibrium between the two primary functional states.

3.2 Generation of MD simulations data

ADK was selected as a model system for testing and validation of MDSS, MDAM, and MDAP due to its well-defined and studied dynamic behaviour.

3.2.1 Unbiased simulations

To test all three tools, unbiased simulations were used to capture the intrinsic dynamics of ADK in the apo form. By allowing the protein to freely explore its conformational landscape without the influence of external forces (e.g. as used in some enhanced sampling techniques), a more complex scenario was tested with a low chance for the system to make spontaneous transitions between its two main conformational states - open and closed. The main goal was to observe the protein's behaviour under conditions typically used in the early stages of studies and under conditions easily replicated for large-scale automatic scanning of mutational effects, as presented in Chapter 4. As presented in the conclusions, MDSS and MDAP can be used directly to process data generated from enhanced sampling techniques. At the same time, code extension is required to incorporate advanced sampling techniques in the MDAM workflow.

Starting from the open state, the system rarely sampled the close conformation. By capturing and generating simulation data with only a minimal fraction of closed conformations, MDSS's ability to reduce the MD trajectories' size while preserving important information on both states could be strictly tested. The unbiased simulations provided a critical basis for evaluating how effectively the tool preserved key conformational transitions during subsampling, particularly the large-scale motions of the LID and NMP domains, which are essential to the ADK function.

Additionally, ADK's conformational transitions provided a robust system for testing MDAM's ability to redesign dynamics through automated mutation engineering. Since sampling the closed state is rarer than sampling the open state, the goal was to explore whether specific mutations could shift the equilibrium to the closed state in an unbiased simulation.

3.2.2 MD data for MDSubSampler

AMBER was used to generate MD data for testing MDSS, and it is known for its robust and accurate biomolecular simulations [167]. The ff14SB force field was chosen due to its proven accuracy in modelling protein structure and dynamics, making it particularly suitable for simulating ADK [167].

The work presented in the following section is based on previously published work [99] that details the system preparation and simulations for testing MDSS.

System Preparation

The ADK structure (PDB ID: 4AKE) was retrieved from the Protein Data Bank [168] and solvated in a truncated octahedral water box with TIP3P water molecules, maintaining a 10 Å buffer between the protein and the box edges [86]. Four sodium ions were added to neutralise the protein charge. Energy minimisation was performed in two steps using 2500 cycles of steepest descent and conjugate gradient each, initially with backbone constraints. Unrestricted minimisation followed. The non-bonded cut-off for both steps was set to 8 Å. Long-range electrostatic interactions were handled using the particle mesh Ewald (PME) method under periodic boundary conditions [94].

Equilibration

The system was equilibrated in NVT and NPT ensembles for 100ps and 250ps, respectively, with a Langevin thermostat [169] and a Berendsen barostat [83]. The temperature coupling time was set to 1.0 ps, and the pressure coupling time was 0.5 ps. These steps ensured that the system reached a stable thermodynamic state before the production phase.

Production

The production run lasted 1 μs with a time step of 2 fs, allowing extensive exploration of conformational samples of the ADK system. To test the MDSS toolkit, five independent replicas of the ADK system were generated at 1 μs starting with the open structure. Notably, one of these replicas exhibited a degree of transition between the open and closed conformational states, allowing an analysis of the LID and NMP domain movements. The resulting bimodal distribution in RMSD (from open conformation) indicated a dominant population of open conformations and a smaller closed sample, as expected by literature evidence suggesting that ADK cannot thoroughly sample a close state in the apo form. This information helped test MDSS's ability to preserve critical structural transitions during subsampling (see section 5.1). This dataset provided a valuable test case for assessing how effectively the tool captures the dynamics of important proteins without losing important information.

3.2.3 MD data for MDAutoMut

To test the MDAM toolkit, ADK was considered an ideal candidate due to its well-characterised dynamic behaviour (open and closed states). Following that, an example of mutations that could potentially alter ADK's dynamics was found in a study by Song et al. [165]. The authors

examined two sets of mutations: one on the LID domain and another on the AMP binding domain (Figure 3.2). Their simulations suggested that these mutations allowed ADK to sample the closed state directly [165]. To test the MDAM toolkit, one of these sets of mutations was used to perform a proof-of-concept study: assuming the positions of these mutations are known (135 and 142), MDAM would have to identify the correct amino acid changes (V135G, V142G) that can have the desired impact of ADK's structure (move ADK from open to closed state).

However, Song et al. [165] used short simulations of 100 ns. Considering that much longer simulations are typically required to observe an attempt at ADK's closure, a decision was made to validate the results of this study on longer timescales. This would create an internally derived reference dataset to confirm the results by Song et al. [165].

The focus was on the LID domain mutations (V135G and V142G) because of the LID's flexibility and critical role in regulating ADK's conformational state. These mutations were selected to offer a clear, interpretable example of how changes in this region impact protein closure, allowing for straightforward visual validation by an expert to assess whether the mutations promote movement toward the closed state. Therefore, to assess how these mutations affect the dynamic behaviour of the enzyme, simulations for the following four structures were generated: wild-type ADK (WT), single mutant V135G, single mutant V142G, double mutant V135G/V142G (DM).

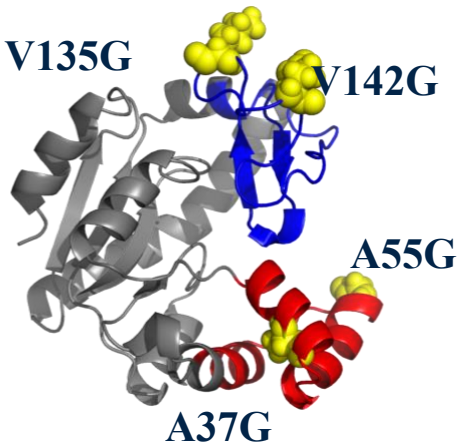


Figure 3.2 Structural representation of wild-type ADK showing with mutations studied by Song et al. [165]. Position of mutations A55G and A37G on the AMP domain (highlighted in red) and of V135G and V142G on the LID binding domain (highlighted in blue) are indicated in yellow spheres. During MD simulations, these mutations were suggested to promote ADK's closure. Structural figures were visualised and rendered using PyMOL [160].

GROMACS [2] was used for the simulation setup due to its powerful Python interface, gmxapi [2], which facilitates the simulation process automation within MDAM. This integration allowed

the preparation, running and analysis of MD simulations directly from the toolkit, ensuring the entire process could be automated. Automating simulations was critical for efficiently handling many replicas and multiple mutations to minimise human intervention and improve scalability.

System Preparation

System preparation was done using GROMACS [2] via the gmxapi Python interface. The AMBER99SB*-ILDN [70] force field was used, widely recognised for its accuracy in modelling protein dynamics. This force field is known to perform well in protein-only simulations under standard conditions and has been validated in numerous studies [70], [170], [171], [172]. The ADK structure (PDB ID: 4AKE) was retrieved from the Protein Data Bank [168] and placed in a cubic box solvated with TIP3P water molecules, ensuring a 10 Å distance between the protein and the box edges. To ensure charge neutrality, four sodium ions (Na+) were added.

Energy minimisation was performed in three stages. The first minimisation used the steepest descent algorithm with position restraints of 2000 kJ/mol/nm² for the heavy atoms of the protein and ran for 50,000 steps. This allowed the system to relax while the protein backbone remained stable. The second minimisation step used the same algorithm but with no positional restraints, allowing further protein relaxation. A conjugate gradient with flexible constraints was applied in the final minimisation step. This allowed the entire system to reach a convergence criterion where the maximum force was reduced to less than 10 kJ/mol/nm² over 10,000 steps.

Temperature and pressure equilibration

Following energy minimisation, the system underwent equilibration to stabilise temperature and pressure.

Temperature equilibration was performed in six steps under the NVT ensemble, gradually increasing the system's temperature from 200 K to 300 K. Positional restraints were applied to ensure structural integrity while allowing the system to adapt to temperature changes. The first step involved heating the system to 200 K with heavy atom position restraints of 2000 kJ/mol/nm², using the Berendsen thermostat [83] to couple the temperature of the protein and solvent. The second step increased the temperature to 250 K, reducing positional restraints to 1000 kJ/mol/nm². The system was subsequently heated to 300 K under the same restraints. The restraints were progressively reduced in the final three steps (500 kJ/mol/nm², 250 kJ/mol/nm², and none), allowing complete relaxation at 300 K.

Pressure equilibration followed under the NPT ensemble in two stages. In the first stage, the system was equilibrated to 1 bar using the Berendsen barostat [83], with position constraints of 210 kJ/mol/nm² enforced on the protein's heavy atoms. This step was run for 500 ps to ensure initial stabilisation. In the second stage, the constraints were maintained, and the system was further equilibrated for 500 ps using the Parrinello-Rahman barostat [85]. Additionally, the V-rescale thermostat [84] was employed during this stage to ensure accurate temperature coupling, with a coupling constant of 0.1 ps applied to both the protein and solvent groups. This approach ensured that the system achieved thermal and pressure stability before production.

Production

Both 1000 ns (i.e., 1 μs) and 300 ns simulations were conducted to evaluate the effects of selected mutations on the LID domain's closure dynamics. Specifically, five replicas for each structure (WT, V135G, V142G, and the DM) were run at 1 μs on the JADE2 supercomputer [173]. Given the computational intensity of the 1 μs simulations, particularly concerning data storage and analysis, ten additional replicas for each structure were created at 300 ns on the ARCHER2 [174] supercomputer. These shorter simulations allowed for broader sampling and aligned with the study by Song et al. [165], facilitating an accurate assessment of mutation-induced changes in ADK dynamics.

3.2.4 MD data for MDAutoPredict

To test MDAP, the second replica of the 1000 ns of the DM (generated on JADE2) was used (Figure 3.3). This specific replica exhibited a clear bimodal distribution in its conformational states, capturing both open and closed conformations of ADK (see section 5.2). This feature made it ideal for training and testing MDAP, as the objective was to evaluate its capability to train a predictor of protein state labels (open, closed, or intermediate) from MD trajectories. By selecting a trajectory that had sampled both conformational states, a robust dataset for developing and validating the ML based predictions was ensured, facilitating the exploration of how effectively the tool can classify and differentiate between distinct structural states.

3.3 Data analysis and validation

After generating the MD simulations, the next step was to ensure the integrity and reliability of this data. This goal is achieved through a series of tests and analysis steps that confirm that the system behaves as expected under biological conditions.

Trajectory concatenation and preprocessing

The first task was to transfer the simulation data from the supercomputers (JADE2 and ARCHER2) to external drives in the Brunel University system. This process enabled efficient local processing and analysis. Once the data was securely transferred, the next step involved concatenating the multiple trajectory files (e.g., part1.xtc, part2.xtc, part3.xtc) into a single, continuous dataset using GROMACS. The 24h limit on HPC queues meant that each simulation required multiple restarts. The concatenation was done to ensure a smooth and comprehensive simulation analysis.

JADE2

System	Trajectory size	Simulation length	Timestep	Saving interval	Replicas
WT	1000001	1000ns	2fs	1 ps	5
V135G	1000001	1000ns	2fs	1 ps	5
V142G	1000001	1000ns	2fs	1 ps	5
DM	1000001	1000ns	2fs	1 ps	5

ARCHER2

System	Trajectory size	Simulation length	Timestep	Saving interval	Replicas
WT	300001	300ns	2fs	1 ps	10
V135G	300001	300ns	2fs	1 ps	10
V142G	300001	300ns	2fs	1 ps	10
DM	300001	300ns	2fs	1 ps	10

Figure 3.3 Table summary of the MD simulations generated using JADE2 and ARCHER2 supercomputers. Simulations were performed for the WT, V135G, V142G, and DM systems. JADE2 simulations had longer trajectories (1 μs per replica), while ARCHER2 simulations used 30 ns trajectories across 10 replicas to explore short-term dynamics. All simulations used a 2 fs timestep and saved data at 1 ps intervals.

Then, water molecules and ions were removed to reduce computational time during analysis and focus exclusively on the protein structure and its dynamics. All subsequent analyses were optimised to focus on conformational changes in the protein.

Figure 3.3 lists the replicas and trajectories, providing an overview of the different runs, their durations, and the structures simulated in each case.

Visual inspection with VMD and PyMOL

To inspect the behaviour of the system during the simulation, visual inspection tools such as Visual Molecular Dynamics (VMD) [175] and PyMOL [160] were used to check the trajectory manually. This step helped ensure that the protein maintained a folded structure and

underwent natural conformational changes without signs of distortions. The .gro and .xtc files generated from the processed trajectories were loaded into these visualisation tools and enabled detailed observation of protein movements and potential issues, ensuring visual coherence of protein behaviour.

Key geometric properties

After visual inspection confirmed that the simulations appeared structurally sound, the focus shifted to analysing key geometric properties:

- **Root Mean Square Deviation (RMSD)**: RMSD was calculated focusing only on the C-alpha atoms. This measurement is crucial for assessing how much the protein's structure deviates from a reference conformation, in this case, the starting conformation, over time. A stable RMSD indicates that the protein remains properly folded and exhibits the expected conformational changes.

- **Radius of Gyration (Rg)**: Calculating the Rg provided insights into the protein's compactness during the simulation. This property is fundamental to confirm that the protein has not unfolded or collapsed, as such events would indicate that the simulation does not accurately represent the protein's natural state in this case.

- **Root Mean Square Fluctuation (RMSF)**: The RMSF was used to determine which protein regions, such as loops or active sites, had the greatest flexibility. This property is crucial for understanding local fluctuations in the protein.

- **Distance Between Two Key Residues:** Specific distances between functionally important residues were measured. These distances helped validate whether key residues involved in protein function are in the expected relative position.

- **Centre of Mass Distance (COM Distance):** The distance between the COM of the LID and AMP binding domain (AMPbd) was calculated to track large-scale domain movements during the simulation. This analysis provided insights into the relative positioning of these domains, which are critical to ADK's functionality, particularly in understanding how the protein switches between open and closed conformations.

In addition to geometric properties, their distributions were studied. Specifically, overlapping distribution plots were generated for all properties after the concatenation of all replicas for the WT and the DM. The plots included both simulated structures (WT and DM) to compare the

system's behaviour before and after the set of mutations was introduced to the LID domain of ADK. Thus, broader trends were observed, as well as differences in the dynamic behaviour of both structures and additional insights beyond individual measurements.

Energy and temperature checks

Monitoring the energy and temperature of the system over time was a crucial part of the validation process, ensuring that the system remained thermodynamically stable and behaved realistically. The energy diagrams captured relative changes in potential, kinetic, and total energy during the simulation. To validate the stability of the simulation, two key measures were applied:

- The calculation of the standard deviation as a percentage of the mean helped analyse the relative fluctuation in energy values ((SD of values/average of values) x 100). This determined the extent of energy fluctuations in the system and ensured that they remained within an acceptable range.

- Another approach to measuring energy fluctuations was to calculate the interquartile range (IQR) and compare it to the median ((Q3 – Q1) / median x 100). A result below 0.5% indicated minimal changes and confirmed the stability of the energy profile.

Likewise, the temperature was monitored to ensure it remained close to the target value of 300K, as expected from the thermostat settings. The average temperature was checked to ensure it remained close to this target with minimal fluctuations (preferably within 0.6% of the target temperature). Large temperature deviations would have suggested simulation problems or distortions, so the goal was to maintain stable thermal behaviour throughout the simulation.

Principal Component Analysis (PCA)

PCA was used to extract collective variables describing the largest amplitude motions. PCs are generally considered a good approximation of putative functional motions. By derivation, they are ranked by decreasing variance, with the first components capturing most of the system's conformational variability. The projection of the simulation data onto the top components can generally show the most critical conformational changes, such as domain opening and closing, like in this study. Additionally, a porcupine plot (see Figure 5.16) was generated as part of the analysis to visualise the collective movements, highlighting key structural changes observed in the simulation data (i.e. opening and closing of ADK).

The main objective of this analysis was to extract easy-to-interpret collective variables associated with the known (and expected) conformational changes. Analysis of the distribution of motions along the PCs was also used to confirm whether a bimodal distribution is observed in the simulation, describing the two functional states (open and closed). PCA was instrumental in extracting and confirming collective motions and measuring the extent to which the mutations affected global dynamics. The bimodal distribution was particularly evident when projecting the simulation data along the first principal component (PC1), which captures the primary conformational change corresponding to the lid closing-to-opening motion. The lid RMSD also demonstrated this bimodal behaviour, reinforcing the observation of distinct conformational states along the direction of the structural transition described by PC1.

Further analysis with VMD and PyMOL

Beyond the initial visual inspections, VMD and PyMOL were also used to examine the structural changes over the filtered trajectory accounting for only single PCs, one at a time. Comparing these projections and filtered trajectory for the WT vs DM, it was possible to show how specific mutations affected the conformational change.

Conclusion of data validation workflow

This strategy ensured that all aspects of the MD simulations were thoroughly validated. By combining automated checks of geometric properties with manual visual inspection in VMD and PyMOL, validation of the reference data used for testing the MDAM toolkit was completed successfully. All data analysis steps, and plotting were performed using R [176], and the code is included in Appendix V. PCA was performed with GROMACS.

3.4 MDAnalysis

MDSS was built on top of *MDAnalysis*, a well-established Python library for analysing MD simulations [8]. This library provides a comprehensive framework for reading, manipulating, and analysing large-scale MD datasets, making it an ideal foundation for MDSS. The integration of MDSS with *MDAnalysis* forms the basis for the automated workflows developed in this project. These workflows were further extended and integrated into the MDAM and MDAP toolkits, enabling a seamless pipeline for subsampling, mutation redesign, and predictive modelling of protein dynamics.

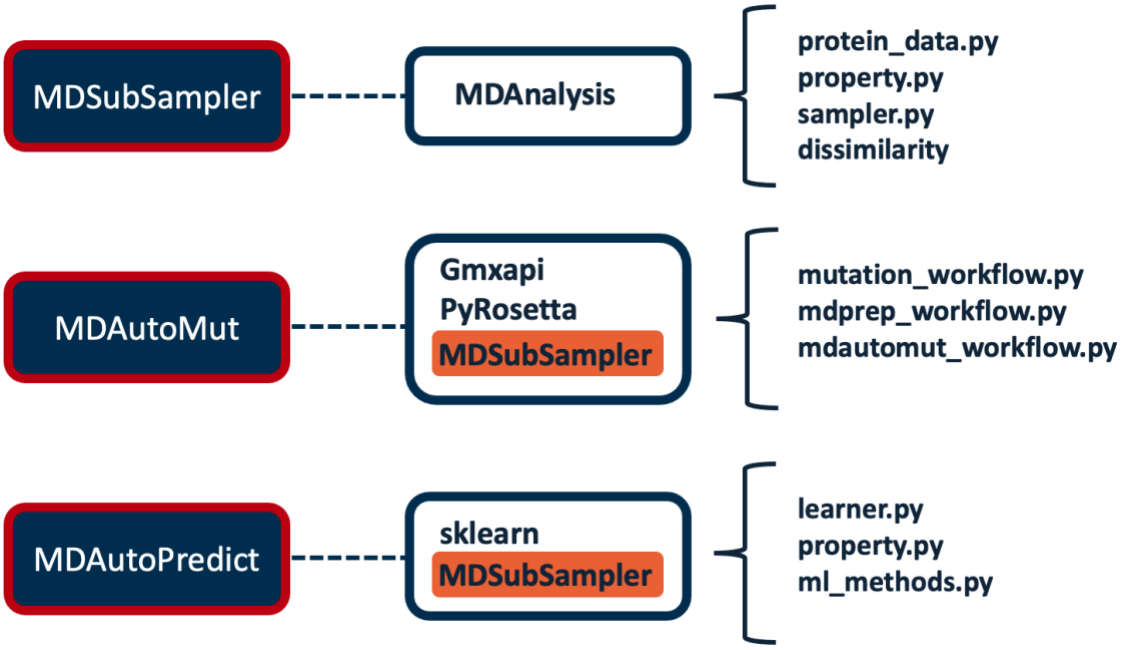


Figure 3.4 Simplified representation of the three tools MDSubSampler, MDAutoMut, and MDAutoPredict and their integration with external Python libraries (e.g., MDAnalysis, gmxapi, PyRosetta, sklearn). The figure highlights the core internal modules of each toolkit (e.g., protein_data.py, mutation_workflow.py, learner.py), illustrating the modular architecture and functional components of the workflow. The MDSubSampler, shown in orange under MDAutoMut and MDAutoPredict, indicates its foundational role in building these toolkits. This is a simplified depiction of the full design, as the complete implementation involves additional classes and modules.

Figure 3.4 provides an overview of the modular design and its core components to illustrate the relationships between the developed toolkits and their integration with existing Python libraries. This figure demonstrates how each toolkit builds upon and interacts with external libraries and internal modules, forming a cohesive computational framework for protein dynamics analysis and prediction.

The *MDAnalysis* library uses the *NumPy* package and treats atoms, residues, and trajectories as objects, making it highly efficient for extracting detailed structural information from MD simulations. The heart of *MDAnalysis* is the `Universe` class, which represents the entire simulation system, including topology and trajectory files. The `Universe` class enables easy extraction of atomic coordinates and other geometric properties using several built-in methods and attributes.

The `Universe` contains an `AtomGroup` object that represents all the atoms in the system and organises them into higher biological groupings such as `Residue` and `Segment`. This design reflects the hierarchical structure of proteins, where atoms form residues and residues are combined to form segments. Each atom belongs to a residue, and each residue is part of a segment. *MDAnalysis*'s design aimed to embed biochemical concepts directly into its class structure, making it highly intuitive for protein modelling and analysis (Figure 3.4).

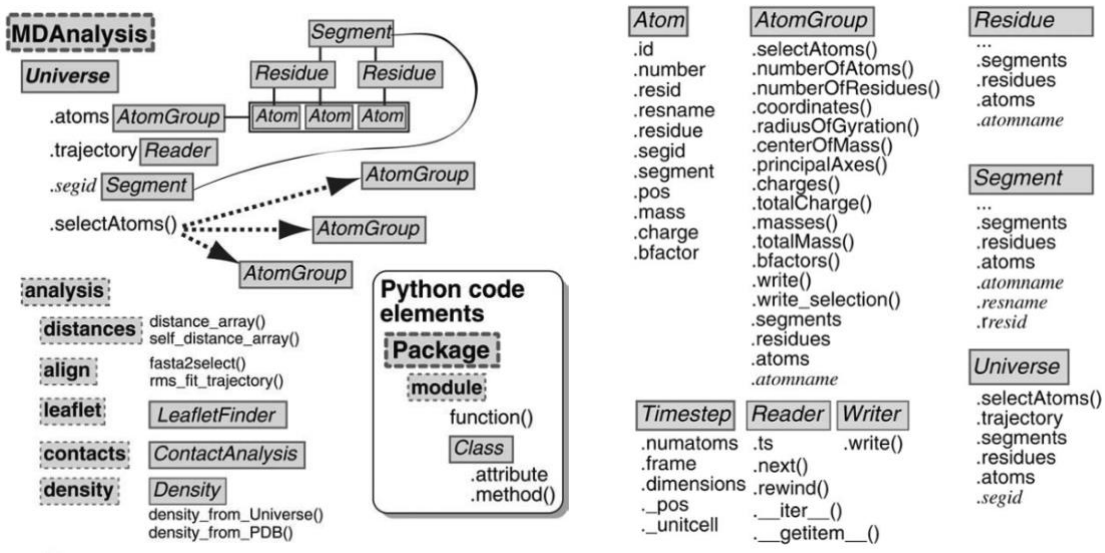


Figure 3.5 MDAnalysis class structure, highlighting the relationships between key classes such as Universe, AtomGroup, Residue, Segment, Reader and Writer classes that enable reading and writing trajectories. The ability to seamlessly integrate MDSS and MDAM into MDAnalysis highlights the strength and versatility of this library, by leveraging its modular design. The figure was taken from Michaud-Agrawal et al. [8], © Wiley Periodicals, Inc.

MDSS leverages these structural representations by using *MDAnalysis* to select and analyse specific groups of atoms within a protein trajectory. The library's functionality is extended to perform *a posteriori* subsampling operations to protein trajectories, where it selects a subset of frames from the simulation, ensuring that the distribution of important geometric properties (e.g. RMSD) is consistent between the original and subsampled trajectory. The `Universe` class selection capabilities allow users to focus on specific atoms or residues and precisely analyse critical regions such as the LID and NMPbd domains in ADK.

A key feature of *MDAnalysis* is the `TimeStep` object, which provides access to individual trajectory frames, including the coordinates of atoms and unit cell dimensions. This feature is crucial for navigating the trajectory, allowing users to jump between frames or select specific points in time for detailed analysis. The `Reader` class, responsible for reading trajectory files, works with the `TimeStep` object and facilitates extracting and analysing specific frames. MDSS and MDAM rely on this functionality to retrieve the most relevant snapshots of protein dynamics for their respective tasks.

The combination of *MDAnalysis* and MDSS enables MDAM to efficiently prepare and simulate the data, engineer mutations into the protein structure (single or multiple) and compare the dynamics between WT and Mutants by comparing distributions of geometric properties.

3.5 PyRosetta

PyRosetta [9] was used in the MDAM workflow for the mutation engineering process within the Mutation module. *PyRosetta* is a Python-based interface for the *Rosetta* software suite designed for high-resolution protein structure prediction, design and mutation analysis [9]. It provides powerful tools for manipulating protein structures at the atomic level, including the ability to introduce mutations, optimise rotamers, and calculate energetics using all-atom force fields [9].

In the MDAM workflow, *PyRosetta* [9] can perform targeted mutations (single, double or multiples) on the inputted protein structure. It is responsible for introducing new residues at selected positions and assessing their impact on the protein's overall structure. *PyRosetta* [9] facilitates this process using the `fa_standard` all-atom force field. This force field is optimised for energy calculations of all atoms and includes contributions from van der Waals interactions, electrostatics, solvation effects, and hydrogen bonds [149]. The `fa_standard` force field ensures that the mutations are energetically favourable and do not cause destabilising structural changes, making it suitable for mutation analysis [149].

The mutation process in MDAM begins with extracting a specific frame from the MD simulation, which is then converted into a pose object. This pose represents the 3D protein structure in *PyRosetta* [9]. The `ChemicalManager` is used to access the required residue types, while the `ResidueFactory` creates new residues that replace the original target residues at specific positions [128]. Once a mutation is introduced, *PyRosetta*'s `PackRotamersMover` optimises the side chain conformations (rotamers), ensuring the new residue fits into the protein structure without steric clashes or other unfavourable interactions [128].

Following the rotamer optimisation, the scoring function `fa_scorefxn`, part of the `fa_standard` force field, can evaluate the energetic impact of the mutation on the protein [149]. This scoring function provides a quantitative measure of the effect of the mutation and can assess how it affected the stability and dynamics of the protein [149]. While the score provides a general indication of structural stability, protein dynamics between WT and mutant structures are compared using MDSS. Specifically, a comparison of the distribution of relevant geometric properties between the WT and each mutant determined whether the mutations promoted the desired conformational shifts in the protein.

In the proof-of-concept study, MDAM was tested with two specific single mutations (V135G and V145G) and a double mutant (V135G/V145G) in the LID domain of ADK to investigate how these mutations can affect ADK's dynamics. This proof-of-concept was performed to validate the performance of the library. However, MDAM is a general-purpose toolkit designed to handle any set of mutations, where the user enters a list of mutations that the toolkit should handle. Depending on the user's preference, the toolkit can flexibly apply these mutations in single or multiple modes and introduce the mutations sequentially or simultaneously.

Once the mutations are applied, the toolkit generates the simulations for the mutated structures. It then compares the dynamics between the WT and each mutant until the desired change in dynamics is achieved. Additionally, MDAM includes a heuristic scanning approach that iteratively narrows the mutation search space by excluding ineffective candidates based on intermediate results. This strategy focuses computational resources on promising mutations, efficiently identifying those that shift protein dynamics toward a desired state (e.g. promoting closure in the LID domain).

Tool/Library	Version	License
MDAnalysis	2.7.0	GPL-2.0
PyRosetta	4 (2024.01 Release)	Free for academic use
gmxapi	0.4.2	LGPL-2.1
GROMACS	2022.6	LGPL-2.1
MDSubSampler	0.0.8	GPL-3.0-only
Seaborn	0.13.2	BSD-3-Clause
Dictances	1.5.3	MIT
Matplotlib	3.6.2	PSF
Numpy	1.23.5	BSD-3-Clause
Pandas	1.5.2	BSD-3-Clause
pytest	7.2.0	MIT
pytest-mock	3.10.0	MIT
psutil	5.9.4	BSD-3-Clause
Scikit-learn	1.2.2	BSD-3-Clause
Setuptools	69.0.3	MIT
Black	22.10.0	MIT
IPykernel	6.23.1	BSD-3-Clause
Jupyter	1.0.0	BSD-3-Clause

Figure 3.6 Overview of software tools and libraries used in this thesis, including their versions and licenses. This table ensures compliance with software usage guidelines, such as for PyRosetta installation. A license was obtained for necessary components.

3.6 GMXAPI

The gmxapi [177] Python interface played a central role in automating the preparation and simulation of the MD data within the MDAM workflow. Specifically, using gmxapi [177], the

simulation process – from system preparation to mutation analysis – was fully automated, ensuring efficient execution and minimal manual intervention. The `mdprep.py` module (Figure 3.4) supported the entire workflow, which was used to prepare the system and generate simulations automatically for the WT and each mutated structure.

The `mdprep.py` module, designed as a standalone script, automates the system preparation and simulation generation for MDAM workflow. This module covers all aspects of system setup, including solvation, ions addition, energy minimisation, temperature and pressure equilibration, and production. The `mdprep.py` is versatile enough to run independently in a Python interface or seamlessly integrate with gmxapi. A standard preparation protocol (see section 3.2.3) was implemented, but MDAM can be easily customised to offer different strategies for system preparation.

When a mutation is inserted in the protein structure, mdprep.py can automatically prepare the system for a new round of simulations. This process includes generating the input files needed to run the simulations and applying the protocol described in section 3.2.3, ensuring consistency across all simulation runs.

Using gmxapi also facilitates efficient resource management by allowing simulations to be submitted to HPC queues for parallel execution or paused depending on the real-time analysis of the simulation results. This flexibility in managing simulations significantly reduces the burden of conducting large-scale mutation studies, especially when testing multiple ADK mutants in parallel.

3.7 Deployment of MDAM on ARCHER2

Given the computational cost of running MDAM on large sets, particularly in preparing and generating MD simulations, it was necessary to deploy the tool on a dedicated HPC facility. Through HecBioSim (EPSRC grant EP/X035603/1), allocation access was provided to the EPCC supercomputer ARCHER2. Deployment was designed ad-hoc for it.

Initial testing on local systems revealed significant limitations in laboratory equipment and personal laptops. The high memory demands, and computing power needed to efficiently simulate MD trajectories for the WT and multiple mutants made local resources unsuitable. In contrast, the ARCHER2 supercomputer’s CPU-focused architecture optimised for software like GROMACS provided an ideal platform for deploying MDAM. Utilising ARCHER2 enabled

the large-scale analysis of mutation effects on protein dynamics, which would not have been possible with local resources.

The overall goal of deploying MDAM on ARCHER2 was to redesign protein dynamics by systematically introducing multiple mutations into the protein structure while employing a heuristic approach to efficiently explore and identify the mutation that produced the desired effect on protein dynamics. By automating the simulation process, MDAM could efficiently test the impact of each mutation, evaluating how well they induced the targeted dynamic changes. Given the intensive computational requirements of generating these simulations, ARCHER2's processing capabilities allowed MDAM to run these tasks much faster and more efficiently than on local systems.

The steps to deploy MDAM on ARCHER2 were as follows:

1. **Initial Setup:** ARCHER2 provides multiple disk partitions, including the home directory for persistent storage of user files and the working directory for high-performance, temporary storage needed during computational tasks. The home directory has limited space and is intended for configuration files and essential data. In contrast, the working directory offers significantly more space and is optimised for large-scale computational tasks. Given the space constraints in the home directory, the Python environment and its dependencies were installed in the working directory. To accommodate this setup, the .local directory (used for storing user-installed Python packages) and the ".cache" directory (used for caching downloaded dependencies) were moved to the working directory. Symbolic links were created from these directories back to the home directory, ensuring seamless functionality while utilising the expanded storage capacity of the working directory.

```
➢ cd $HOME
➢ mkdir /mnt/lustre/a2fs-nvme/work/e280/e280/$USER/pyenvs
➢ ln -s /mnt/lustre/a2fs-nvme/work/e280/e280/$USER/pyenvs .
➢ mkdir /mnt/lustre/a2fs-nvme/work/e280/e280/$USER/home
➢ mv .local /mnt/lustre/a2fs-nvme/work/e280/e280/$USER/home
➢ mv .cache /mnt/lustre/a2fs-nvme/work/e280/e280/$USER/home
➢ ln -s /mnt/lustre/a2fs-nvme/work/e280/e280/$USER/home/.local .
➢ ln -s /mnt/lustre/a2fs-nvme/work/e280/e280/$USER/home/.cache .
```

2. **Environment Creation:** A Python virtual environment was created to install and manage the required software packages. These included key libraries for MD

simulations and ML, such as *MDAnalysis*, gmxapi and *PyRosetta*. Since MDAM integrates these libraries into its workflow, setting up this environment was critical to ensure smooth execution on ARCHER2.

```
➢ module load PrgEnv-gnu
➢ module load cray-python
➢ python -m venv --system-site-packages /mnt/lustre/a2fs-
  nvme/work/e280/e280/$USER/pyenvs/mddev
➢ source /mnt/lustre/a2fs-
  nvme/work/e280/e280/$USER/pyenvs/mddev/bin/activate
➢ python -m pip install ipython seaborn scikit-learn mdanalysis
```

3. **Installing gmxapi and PyRosetta**: The installation process involved loading the appropriate GROMACS module and running the *PyRosetta* installer to ensure all dependencies were configured correctly.

```
➢ source /work/y07/shared/apps/core/gromacs/2022.4/bin/GMXRC
➢ pip install --no-cache-dir gmxapi
➢ pip install pyrosetta-installer
➢ ipython
➢ import pyrosetta_installer
➢ pyrosetta_installer.install_pyrosetta(type='MinSizeRel')
```

4. **Job Submission:** Test jobs were submitted after setting up the environment to ensure that the MDAM workflow could run smoothly on ARCHER2. A simple job submission script was written to run a Python test script to ensure the environment was configured correctly, and the job ran successfully on a single node with a CPU. The python_test.py script validated the deployment by loading a protein structure into MDAM's environment.

submission_script.sh

```
#!/bin/bash --login
#SBATCH --job-name=python_test
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1
#SBATCH --time=00:10:00
#SBATCH --account=e280-Pandini
#SBATCH --partition=standard
#SBATCH --qos=standard
```

```
module load cray-python
source /mnt/lustre/a2fs-nvme/work/e280/e280/$USER/pyenvs/mddev/bin/activate

python python_test.py
```

python_test.py

```
from mdam.protein import Protein
pdbfilename = "4AKE.pdb"
p = Protein(pdbfilename, pdbfilename, pdbfilename)
```

An MDSS and MDAM zip file was downloaded from the GitHub repository to the home directory in ARCHER2. After extracting the files, the required Python scripts (`mdprep.py`, `mutation.py`, `mdautomut_workflow.py`) were created in the home directory. These scripts were essential for managing the workflow within MDAM, including MD preparation, mutation engineering, and simulation production.

The successful deployment of MDAM on ARCHER2 demonstrated that the tool could be executed efficiently on high-performance computing resources, enabling large-scale automation of protein dynamics redesign. This deployment highlights the potential to explore multiple mutations and their impact on dynamics at a system level, which would be unfeasible with standard computing resources.

However, demonstrating successful deployment does not inherently confirm the correctness of the tool's results. To ensure accuracy, additional validation steps were implemented. These included benchmarking MDAM's outputs against known experimental data and simulated results from smaller-scale systems to verify consistency and correctness. This dual approach establishes the deployment's feasibility and validity by showing that MDAM can run effectively at scale and confirming that it produces scientifically reliable results.

Distribution and availability

To ensure accessibility for future users, the MDSS library [99], which is a critical component of this workflow, is openly available under the GPL-3.0 license at its GitHub repository: https://github.com/alepandini/MDSubSampler.

MDAM is hosted on GitHub (https://github.com/alepandini/MDAutoMut) and is going to be publicly available at time of submission of the associated manuscript.

MDAP, while not currently open source, is actively being developed to make it publicly accessible upon publication of relevant manuscript. The code for MDAP is provided in Appendix IV.

In addition, all submission scripts necessary for running the tool on ARCHER2 after deployment are included in Appendix II.

3.8 Containers – Docker

Running MDSS with all its dependencies presented significant challenges, especially when switching operating systems (lab machine-Linux, personal laptop-MacOS). The different ways these operating systems handle libraries such as *MDAnalysis* and *NumPy* made maintaining compatibility difficult. Ensuring the correct versions of these dependencies and managing environment-specific variations often resulted in conflicts that hindered the toolkit's testing progress. Given the large volume of MD trajectories to be processed, maintaining a consistent design was critical to implementing and testing the toolkit.

To solve this problem, a Docker container [178] was employed to package MDSS and its dependencies in a single, self-sufficient environment, and it is made available as part of the software package. Docker allows the user to create a container that encapsulates the operating system, necessary libraries and the application itself. This approach guarantees platform consistency and allows the software to operate smoothly regardless of the underlying system.

Using Docker brought several benefits: First, it eliminated the need to manually install dependencies, which was particularly beneficial when configuring multiple machines (lab machines, personal laptops). Second, by isolating the software environment from the host system, Docker reduced the risk of version conflicts and made the entire pipeline more manageable. It also made the results easier to reproduce because users could pull the same container and get identical results without additional configuration. The container is specifically designed to handle the MDSS workflow.

The Docker image was built using a `Dockerfile` that defined the necessary steps to set up the environment, including installing MDSS and other required libraries. Users could easily interact with the container through the command line, making the setup user-friendly and adaptable to various computational environments.

Further details on the Docker container configuration and instructions on how to use it with MDSS can be found in the MDSS GitHub repository, available at https://github.com/alepandini/MDSubSampler.

3.9 Wrapping the toolkits with Poetry

Since MDSS and MDAM are Python-based toolkits, managing dependencies was essential for ensuring smooth development and distribution. Poetry [179] was used to package the libraries and make them pip-installable, a tool specifically designed for managing Python dependencies and packaging Python projects. Poetry helped to efficiently organise all the required libraries, maintain compatibility across environments, and streamline the packaging process for both toolkits.

By defining all dependencies in a single `pyproject.toml` file, Poetry simplified the setup and ensured consistency. Additionally, it facilitated the creation of isolated virtual environments, preventing conflicts with system-wide packages or other projects. Users can easily install the libraries from the Python Package Index (PyPI) using the pip install command.

To set up and run MDSS and MDAM with Poetry, users can follow these steps:

- `poetry install # Install dependencies and create the virtual environment`
- `poetry build # Build the package`
- `poetry shell # Activate the virtual environment`

These commands ensure proper installation, packaging, and environment setup. Detailed instructions for both toolkits can be found in the respective GitHub repositories (see section 3.8 for links).

MDAP has been developed to demonstrate a proof-of-concept, extending MDAM into an ML pipeline capable of predicting conformational states for a given molecular system. Future work will focus on refining this tool and packaging it as a standalone toolkit for broader use and integration into research workflows.

3.10 Summary

This chapter presented the methods and tools used to develop, test, and validate the MDSS, MDAM, and MDAP toolkits. It detailed the protocol for setting up MD simulations using ADK as a case study, including the generation of unbiased simulations and analysis to ensure the reliability of the data. The chapter also described integrating software such as *MDAnalysis*, *PyRosetta*, and gmxapi and deploying the toolkits on ARCHER2 for scalability. Docker containers and Poetry were employed for environment management and distribution to ensure reproducibility and ease of use.

4 Design, implementation, and testing

The design, implementation, and testing of the three novel tools that were developed in this thesis are presented in this chapter: MDSubSampler (MDSS), MDAutoMut (MDAM) and MDAutoPredict (MDAP). By providing a concise overview of design, implementation, and testing, the chapter provides a compact and clear framework for understanding how these tools collectively address the complexities of managing large-scale MD data, automatically redesigning protein dynamics, and integrating MD simulations into managing the ML framework to perform predictions of MD properties. First, the software design and core components of each tool are introduced, followed by the functionality, software implementation and accessibility. Finally, the approach to testing each toolkit is shown.

4.1 MDSubSampler tool

MDSS was designed to address the need for modular tools to subsample large MD datasets and preprocess them for ML/DL workflows. MD trajectories are recorded in a structured format that inherits the convention for data recording from the early times of molecular simulation studies and is not designed to be directly used in ML/DL pipelines. Additionally, the large volume of recent long timescale trajectories makes them challenging to analyse, and this type of analysis can be computationally intensive. They generally have a low signal/noise ratio.

MDSS is designed for *a posteriori* subsampling large MD trajectories. Specifically, the toolkit can extract important protein conformations, reduce data size, and remove noise while preserving key structural information. Additionally, MDSS can reformat MD data for ML/DL purposes and can be used to compare protein dynamics between different trajectories by calculating similarity scores between values of dynamics descriptors using statistical methods.

The work presented in this chapter is based on previously published work by Oues et al. [99] that details the design, implementation and testing of MDSS.

4.1.1 Software design and core components

MDSS is an object-oriented modular Python library. Built on top of the *MDAnalysis* framework [8], it performs subsampling of large MD trajectories, reformatting data for ML pipelines, and enables comparison of time-dependent descriptors of protein dynamics across simulations. The toolkit is built around four core Python classes: `ProteinData`, `ProteinProperty`,

`ProteinSampler`, and `Dissimilarity`. Each of these classes serves a specific role within the framework, and they contain specialised subclasses to model different properties and processes associated with molecular dynamics data. The class diagram in Figure 4.1 shows the relationships between the core classes and their subclasses.

ProteinData class

The `ProteinData` class is central to the toolkit and handles protein structure, topology, and MD trajectory data. It is a wrapper class for low-level *MDAnalysis* [8] types. The class contains a dictionary for storing references to various property objects (see below), each representing a specific time-dependent calculated property (e.g., RMSD, Rg, PCs) along the MD trajectory. By organising MD data via the `ProteinData` class, MDSS enables efficient property value mapping to trajectory frames and easy manipulation of property values across frames.

ProteinProperty class and subclasses

The `ProteinProperty` class and its subclasses provide the framework for computing and storing properties that can capture time-dependent measures of protein dynamics. The primary subclasses include `GeometricalProperty` and `PCAProperty`.

The `GeometricalProperty` subclasses calculate key geometric properties such as `RMSD`, `COMDistance`, and `RadiusOfGyration`. These properties enable the analysis of structural stability, compactness, and relationships between domains within the protein. By tracking these properties across trajectory frames, `GeometricalProperty` can provide insights into conformational changes and functional movements within the protein system.

The `RMSD` property measures the average positional deviation of atoms from a reference structure, helping to monitor significant conformational changes over time. For example, `RMSD` is valuable for observing the extent of structural changes during an MD trajectory. The `COMDistance` property tracks the distance between the centre of masses of specific groups within the protein, for example, between different domains or between a protein and a ligand. This measure helps study relative movements within the protein and functionally relevant interactions. Additional properties such as `RadiusOfGyration` and `DihedralAngles` provide further insights into the protein's compactness, geometric orientation, and secondary structure transitions, making MDSS applicable to different protein dynamics analyses.

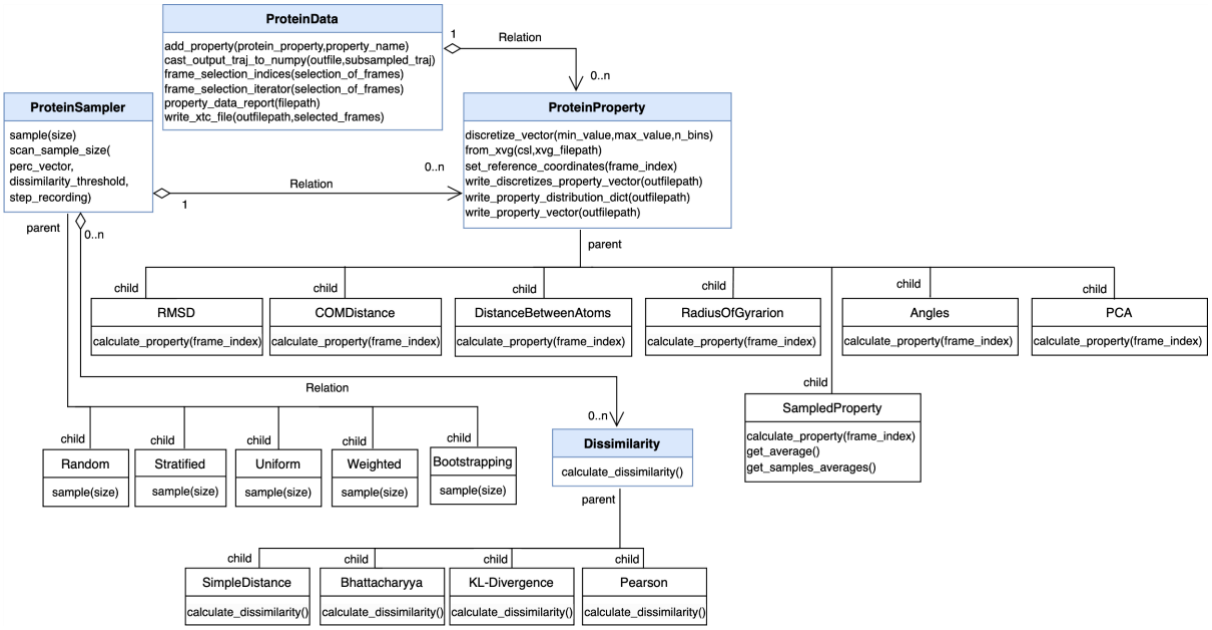


Figure 4.1 Class diagram of MDSS, showing relationships among main classes (ProteinData, ProteinProperty, etc.) and their multiplicities, with symbols indicating the number of instances each class can be linked to in relation to others. The figure was taken from the MDSS's paper by Oues et al. [99].

On the other hand, `PCAProperty` implements Principal Component Analysis (PCA), a dimensionality reduction technique. PCA identifies the principal directions or components in which the variance of atomic motion is highest, thereby isolating the dominant, slower collective motions within the protein system. These slow, large-scale movements often describe significant conformational transitions, such as the shifting between open and closed states of protein domains. The implementation of PCA in *scikit-learn* is used in the MDSS framework to perform the analysis and then to project MD trajectories onto these principal components (PCs). This projection captures the protein's key conformational transitions over time and analyses the movements most likely associated with functional changes. These essential protein movements are often the most relevant to understanding protein functional dynamics.

Within MDSS, `ProteinData` objects organise and manage MD trajectories, providing a structured way to associate specific trajectory frames with different calculated properties. A key feature of `ProteinData` objects is their dictionary attribute, which contains references to `ProteinProperty` objects. Each `ProteinProperty` object records property values across trajectory frames, allowing for straightforward mapping between calculated property values and their corresponding time frames. As the values for different properties can be mapped to the same trajectory information, any selection, subsampling or extraction of specific frames can be easily applied to the associated property value vectors. This modularity of the software

provides flexibility in analysis and enables the extraction of meaningful samples from large MD datasets.

ProteinSampler class and subclasses

Sampling strategies implemented in MDSS are encapsulated in subclasses of `ProteinSampler`, which serve the critical purpose of selecting frames and their associated property values from MD simulations. These strategies are essential to adapt the sampling process to different research objectives, such as identifying important conformational states or analysing specific system properties. Specifically, MDSS offers flexibility by allowing users to select the appropriate sampling technique depending on the research problem. The library is designed with a hierarchy of classes, with `ProteinSampler` being the main sampler class, and different sampling strategies, including `RandomSampler`, `StratifiedSampler`, `UniformSampler`, `WeightedSampler`, and `BootstrappingSampler` to represent subclasses. In this context, sampling is done *a posteriori* (i.e. on the trajectory data after the simulation has been completed as an analysis strategy). This differs from the sampling of the conformational space that is done during the simulation.

The `RandomSampler` selects frames using a random approach. Combined with the `Dissimilarity` class, it can ensure that the distribution of relevant properties (e.g., RMSD) is preserved between the original and sampled trajectory. This technique is suitable for reducing data size without information loss. To ensure that the sample trajectories preserve the system's structural information, MDSS provides distance metrics (e.g., Bhattacharyya distance) that help determine the minimum sample size required to capture the distributions of relevant properties (see the `Dissimilarity` class below).

In contrast, the `UniformSampler` is designed to ensure uniform coverage of a selected property across the value ranges. For example, if a property relates to the opening of a protein pocket, consistent sampling ensures that frames representing the full range of pocket opening states are included. This method is useful when the goal is to explore the conformational landscape of a protein uniformly across a collective variable, as it can provide a balanced representation of different states (if the original sampling spans the state transition along the collective variable).

The `StratifiedSampler` provides a more structured approach by dividing the trajectory into subsets based on predefined state labels or discrete conditions and then sampling proportionally from each subset. This stratification ensures that the protein's user-defined

group of frames (or state) is adequately represented in the final sample. This approach is valuable when different conformational states are present in the trajectory and can be labelled, as stratified sampling can recover each state proportionally, providing an equally representative view of the protein's dynamic behaviour across the defined groups (or states).

The `WeightedSampler` follows a probability-based approach and assigns selection probabilities based on the frequency or importance of conformational states. This method is ideal when certain states, such as rare or functionally critical conformations, are of research interest. By prioritising these states through increased weights, this sampling strategy ensures that important frames are well represented, and important features of the dataset are preserved that might otherwise be lost in random sampling.

Finally, the `BootstrappingSampler` involves repeated sampling with replacement, generating multiple subsamples from the original dataset. This method is suitable for statistical analysis and cross-validation because it enables the creation of robust estimates for property distributions. Additionally, it is possible to estimate confidence intervals for the dynamic properties by assessing variability between bootstrapping samples. This has not been demonstrated in the current study, but it is an implemented feature to benefit MDSS's users.

`ProteinSampler` objects in MDSS are created with a specific `ProteinProperty` object as a reference, facilitating subsampling across trajectory frames based on particular properties. Each `ProteinSampler` object returns subsampled `ProteinProperty` objects, capturing a compressed yet representative set of data points. Once subsampling is complete, MDSS allows users to save property values and their corresponding trajectory frames to an output file. This feature ensures accessibility to sampled data regardless of the original trajectory, supporting downstream analysis and integration with other tools.

Dissimilarity class

In MDSS, the sampling of trajectory frames is guided by the distribution of associated properties. The primary objective is to compare a property distribution between the original and the sampled set. To achieve this, metrics for calculating distances between distributions are essential, and the `Dissimilarity` class and its subclasses play an important role in this process.

The `Dissimilarity` class implements well-established distance measures, enabling users to quantitatively evaluate how well the subsampled dataset retains the statistical properties of

the original distribution. These metrics include the Bhattacharyya distance [101], which quantifies the degree of overlap between probability distributions, the Kullback-Leibler divergence [180], which measures divergence from one distribution to another, and the Pearson correlation distance [102], which assesses linear correlation. By applying these distance metrics, users can optimise sampling strategies to ensure that the subsampled data accurately reflects the dynamics of the original simulation.

PropertyPlot class

In addition, MDSS has a `PropertyPlot` class for visual comparison of distributions, allowing users to visualise both the original and subsampled data in a comparison framework. The class is built on top of *matplotlib* Python library. The graphical representation generated by MDSS provides immediate insight into how the subsampled data matches the overall trajectory. This visualisation complements the quantitative metrics of the dissimilarity class and provides users with a comprehensive toolkit for numerical and graphical assessment of protein dynamics.

Utility functions

In addition to its primary classes, MDSS includes a utility module that supports important tasks throughout the toolkit's workflow. This module provides functions for file I/O, data transformation, mathematical operations, and command-line argument parsing.

Logging and configuration

MDSS includes a logging module (`log_setup.py`) that captures workflow steps, warnings, and errors to support efficient debugging and process tracking. Additionally, configuration management is made easier by parser utilities that process command-line input and custom settings. This modular design with logging and configuration support ensures ease of use, reproducibility, and the flexibility to adapt to different analysis workflows.

4.1.2 Functionality

MDSS facilitates the efficient handling and analysis of large MD trajectories and provides a streamlined process for *a posteriori* subsampling, property calculation, and data preparation for ML and DL applications. Figure 4.2 shows the data flow within MDSS, highlighting both the input files and the resulting outputs generated from an MD trajectory. The toolkit accepts inputs

such as MD trajectories, reference structure, atom selection criteria, geometric property specification, sample size or range, and dissimilarity measure. The user can tailor the sampling and analysis in MDSS to meet specific research objectives, targeting the protein's different structural features or dynamic properties.

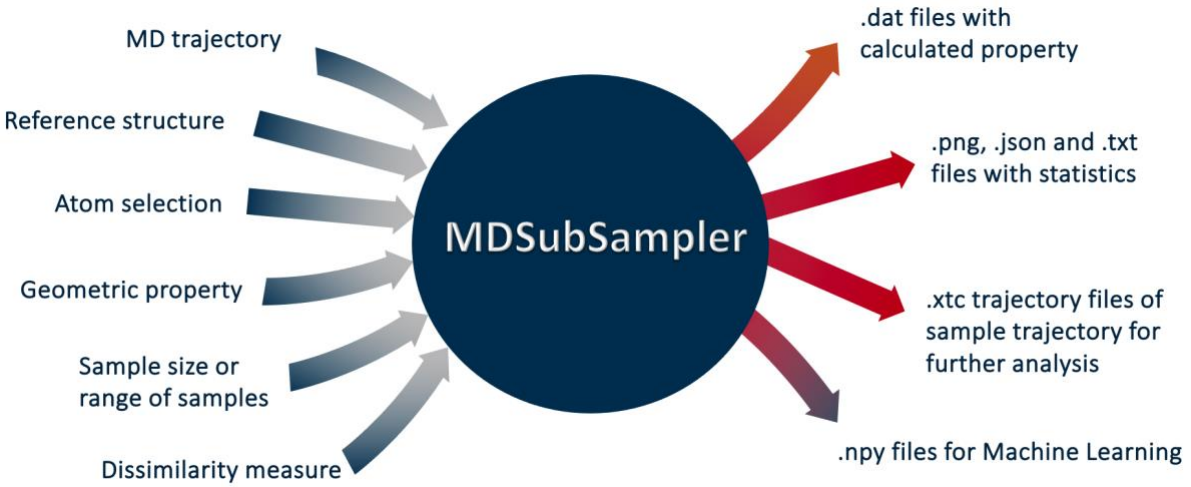


Figure 4.2 Overview of MDSS's data flow. The MDSS toolkit takes MD trajectory data along with user-defined inputs, such as geometric properties and sampling parameters, and outputs data in various formats. These outputs are tailored for different applications, from statistical analysis and visualisation to machine learning integration.

Input files and parameters

MDSS accepts user inputs that guide property calculations, subsampling strategies and output generation:

1. MD trajectory file(s) (.xtc): This file includes the atomic coordinates for each simulation frame and serves as the primary dataset for property calculations.
2. Reference structure file (.pdb, .gro): This file provides the initial (or reference) protein structure, used to calculate metrics like RMSD.
3. Atom selection: allows users to specify regions of interest within the protein, such as specific residues or domains, enabling focused analysis on areas relevant to the research question.
4. Geometric properties: defines which properties to calculate (e.g. `RMSD`, `COMDistance`, `RadiusOfGyration`). Users can choose properties based on their relevance to the protein's dynamics and intended analysis.
5. Sampling strategy and size: This specifies the subsampling method (e.g. `RandomSampling`) and the sample size of the target trajectory.

6. Dissimilarity measure: a metric (e.g. Bhattacharyya distance) for comparing property distributions between sampled and original data, ensuring that subsamples retain essential dynamic features.

The tool also accommodates the case when precalculated frame-dependent property files are available. This feature is useful when users reanalyse or compare existing data without recalculating properties, saving time and computational resources.

Output options

MDSS generates multiple output files, allowing users to choose the appropriate formats for their specific research goals. Outputs are optional, so users can specify which files they need and in which formats. Here is a breakdown of the output options:

1. .dat files store calculated property values for each frame, allowing post-processing and evaluation of subsampling by checking the distributions of these properties. The properties can capture the system's dynamics.
2. .png files include visual representations of property distributions, useful for quickly assessing sampling accuracy and comparing subsampled data against the original trajectory.
3. .json and .txt files contain statistical summaries of the calculated properties, including distribution metrics and distances.
4. .xtc trajectory files represent subsampled trajectories, allowing further analysis in other tools or workflows that require the trajectory directly.
5. .npy files contain compressed data for ML/DL workflows. MDSS offers the choice of having the MD data transformed into the appropriate format for ML/DL and splits the data into learning and testing sets (70% learning - 30% testing).

4.1.3 Software implementation and accessibility

MDSS is an open-source Python library, is hosted on GitHub (https://github.com/alepandini/MDSubSampler), and primarily developed within Visual Studio Code [181]. The project is built with git version control to ensure a robust, well-documented development process. MDSS is an installable Python package deployed using the Poetry package manager [179], which streamlines installation, resolves dependencies, and promotes reproducibility across computational environments.

Poetry facilitates ease of setup by managing all dependencies in the `pyproject.toml` file (Figure 4.3), simplifying the process of deploying MDSS as a Python package. This approach not only ensures that all required libraries are easily accessible but also guarantees consistency across different systems where MDSS might be installed.

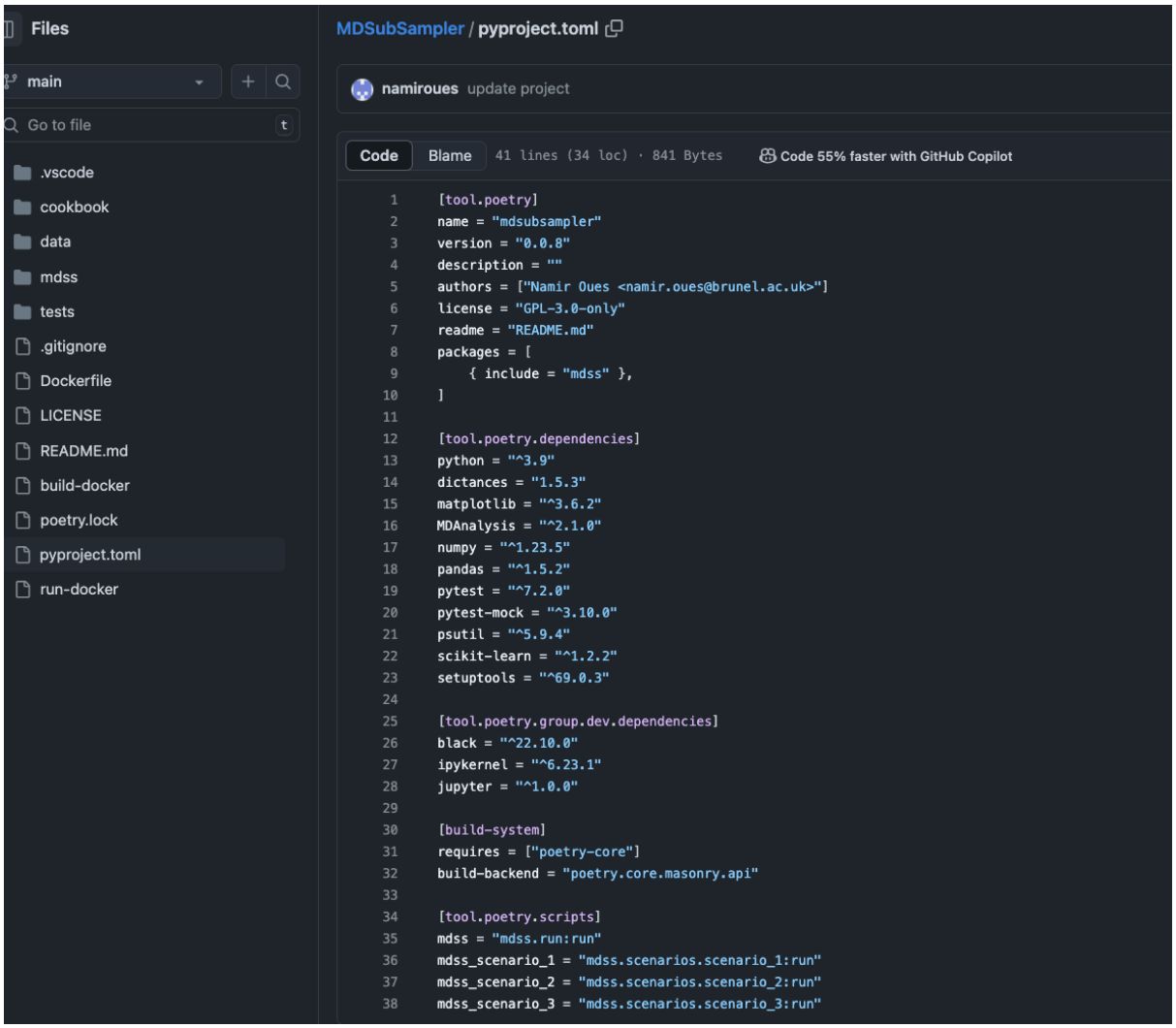


Figure 4.3 MDSS's pyproject.toml file. This configuration file defines the MDSS project settings and dependencies within the Poetry environment, streamlining the installation process and ensuring consistency across systems. Poetry manages both the primary dependencies for MDSS's core functions and additional dependencies for optional features, enhancing reproducibility and ease of use.

To further support accessibility and reproducibility, MDSS is also packaged within a Docker container, enabling users to deploy the toolkit in a self-contained, isolated environment. The Docker container includes all necessary dependencies and configurations, eliminating compatibility issues and ensuring that MDSS operates consistently across various computational systems, regardless of the underlying platform. This containerisation approach allows users to run MDSS with minimal setup, which is particularly valuable for those without

extensive experience in configuring Python environments. The repository on GitHub provides instructions to users on how to use Docker for the project.

The MDSS environment is designed for compatibility with Python version 3.9 (or later), and it incorporates a suite of essential dependencies to support protein dynamics analysis and subsampling within MD simulations. Key dependencies include *MDAnalysis* (version 2.1.0 or later) [8], which handles protein structure and trajectory management, and *NumPy* for efficient numerical processing of MD data. The *dictances* package is employed for calculating various statistical distances, supporting the subsampling accuracy assessments, while *scikit-learn* helps with PCA and integration with ML/DL pipelines. *Pandas*, *Matplotlib* and *Seaborn* packages are included to support different parts of the toolkit's analysis.

The toolkit is developed to satisfy three distinct user groups: novice users with minimal software development experience can utilise pre-prepared scenario scripts, advanced users can operate through a Unix-like command-line interface and scientific software developers can leverage reusable Python library classes for their projects.

Novice users

For beginners, MDSS offers predefined "scenario recipes" or scripts that cover a variety of use cases. These scripts are provided as standalone Python files, ready to be run independently, and as interactive Jupyter Notebooks. The Notebooks offer a flexible, easily customised template, allowing new users to explore different sampling strategies and analyses without extensive programming knowledge. All scenarios are stored in a dedicated cookbook folder on GitHub, where the community can contribute additional examples or improvements. Figure 4.4 presents an example scenario that illustrates the sampling process for size reduction with minimal information loss, comparing RMSD distributions of the original and subsampled trajectories.

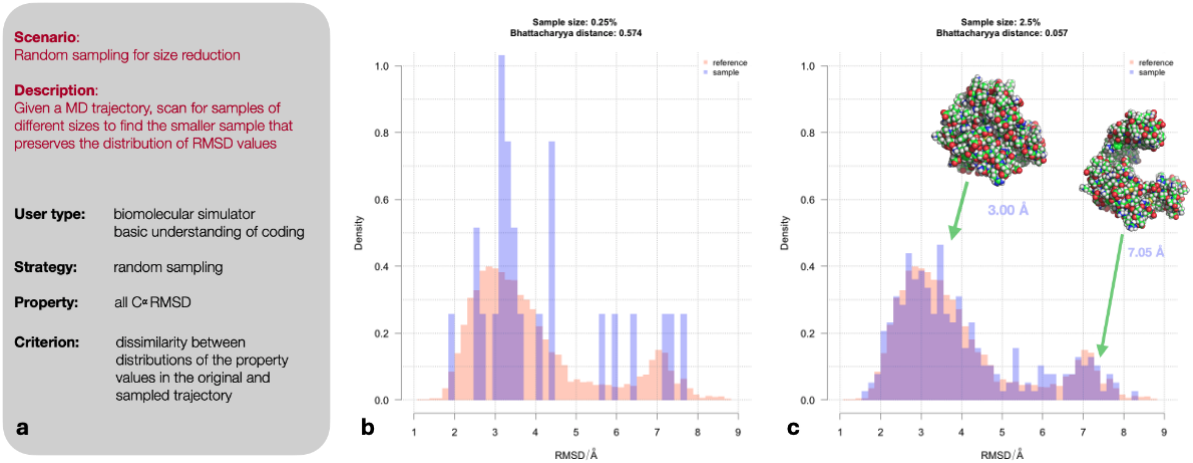


Figure 4.4 Summary description of an example scenario: Random sampling for size reduction, where different subsample sizes are extracted to preserve the information in the distribution of values for a reference property. The two plots compare the distributions of RMSD over the coordinates of all Cα atoms in the original and subsampled trajectory for sample sizes of 0.25% and 2.5%. The distance between the sampled and original distributions was calculated using Bhattacharyya distance: 0.574 (for 0.25%) and 0.057 (for 2.5%). A subset of 2.5% is the smallest sample for which the shape and peak location of the distribution of RMSD is preserved. Example structures for an open and closed conformation of ADK are reported in the top right of the second plot. Distribution plots were generated with R [176] and protein structure images with PyMol [160]. The figure was taken from the MDSubSampler paper by Oues et al. [99].

Advanced users

Advanced users, such as structural biologists, can leverage a Unix-like command-line interface, which provides greater flexibility and control over the data processing workflow.

```
..Development/MDSubSampler [main|✔] ››› python mdss/run.py -h
usage: run.py [-h] [--traj TRAJECTORY_FILE] [--top TOPOLOGY_FILE] [--xvg XVG_FILE] --prefix FILE_PREFIX --output-folder
              OUTPUT_FOLDER
              [--property {RMSD,DistanceBetweenAtoms,COMDistance,RadiusOfGyrationProperty,TrjPCAProj,DihedralAngles,Angles}]
              [--fit] [--atom-selection ATOM_SELECTION]
              [--sampler {RandomSampler,UniformSampler,WeightedSampler,StratifiedSampler,BootstrappingSampler}]
              [--seed-number SEED_NUMBER] [--strata-vector STRATA_VECTOR] [--strata-number STRATA_NUMBER]
              [--n-iterations NUMBER_OF_ITERATIONS] [--weights-vector WEIGHTS_VECTOR] [--size SIZE]
              [--dissimilarity {Dissimilarity,Bhattacharyya,KullbackLeibler,Pearson}] [--step-recording]
              [--machine-learning]

Subsampler tool

optional arguments:
  -h, --help            show this help message and exit
  --traj TRAJECTORY_FILE
                        the path to the trajectory file
  --top TOPOLOGY_FILE   the path to the topology file
  --xvg XVG_FILE        the path to the xvg file containing the calculated property
  --prefix FILE_PREFIX  the prefix for output files
  --output-folder OUTPUT_FOLDER
                        the path to the output folder
  --property {RMSD,DistanceBetweenAtoms,COMDistance,RadiusOfGyrationProperty,TrjPCAProj,DihedralAngles,Angles}
                        Property
  --fit                 Indicates the superposition of trajectory before calculating RMSD
  --atom-selection ATOM_SELECTION
                        Atom selection for calculation of any geometric property, If a selection is not specified then
                        the default CA is used
  --sampler {RandomSampler,UniformSampler,WeightedSampler,StratifiedSampler,BootstrappingSampler}
                        Sampler
```

Figure 4.5 Parser help interface on Linux command line with options and required user arguments.

This interface supports detailed configuration options, allowing users to specify the sampling method, geometric properties, and dissimilarity measures. An built-in parser `help` command

lists all available options and guides users through the setup, ensuring accessibility for users with moderate command-line experience. Figure 4.5 demonstrates the help interface for navigating these options.

Scientific software developers

MDSS supports further development and customisation, making it valuable and practical for scientific software developers. The toolkit provides a range of reusable Python library classes that form a flexible foundation for building new tools or extending MDSS's functionality within custom workflows. Developers can download the source files (tarball file) from GitHub and modify them using an IDE like Visual Studio Code [181]. Figure 4.6 illustrates the modular structure of MDSS, detailing the organisation of classes and modules, which facilitates easy navigation and customisation.

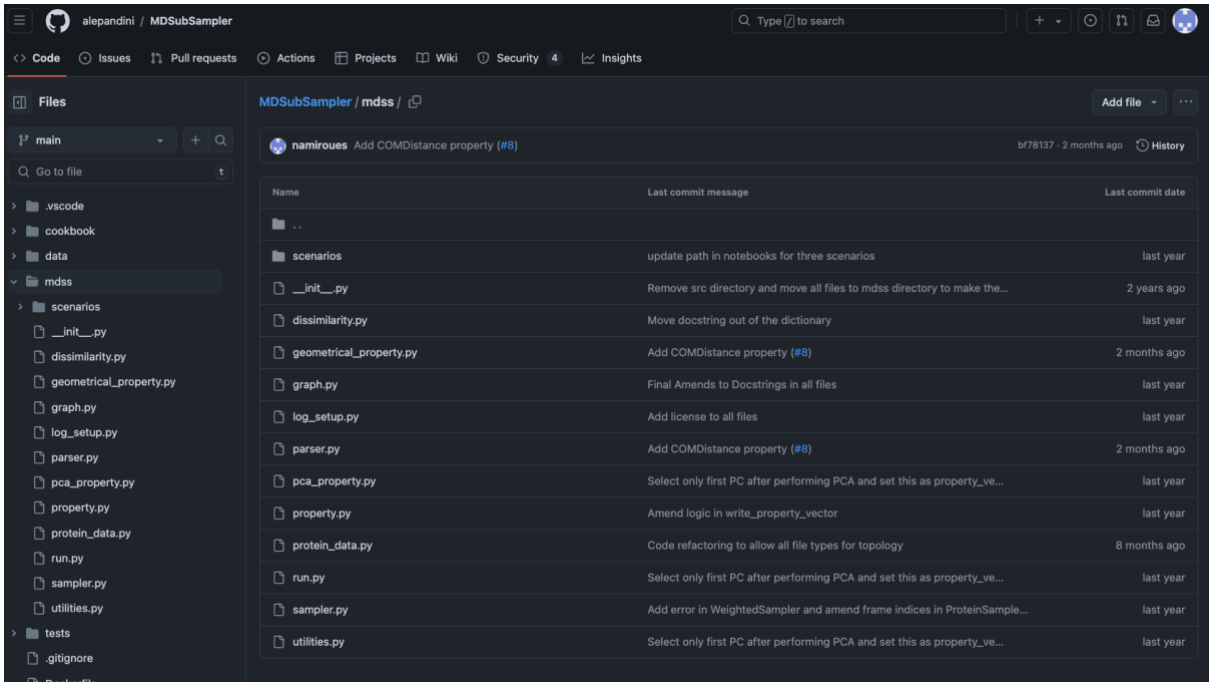


Figure 4.6 Hierarchy of files (modules) in MDSS library as is shown on GitHub page.

Licensing and contribution

MDSS is released under the GPL-3.0 license, ensuring its status as an open-source and community-accessible toolkit. Users are free to use, alter, and distribute the software, if any, under this license, provided that any derivative works remain open source under the same license. This approach encourages collaborative development and supports contributions from the research community.

Researchers and developers are encouraged to participate in the project on GitHub by adding to the codebase, reporting bugs, and making suggestions for enhancements. The `README` file offers thorough documentation, installation guidelines, and usage examples to ensure a seamless onboarding process. MDSS's open-source nature aligns with its objective of enabling reproducible and scalable analysis of MD data.

4.1.4 Testing

The functionality and performance of MDSS were tested to validate its core modules and workflow, including subsampling accuracy, property calculations, and output handling. The goal was to ensure that MDSS could effectively handle large MD data and produce reliable, meaningful subsamples while preserving essential information about the system's dynamics. Testing focused on two key areas: first, verification of each sampling strategy's accuracy, and second, validation of calculated properties and dissimilarity measures.

Testing was conducted using MD trajectories of the ADK system, a model system with well-defined conformational states. This system was selected due to its distinct open and closed states, which allowed for complete testing of MDSS's ability to capture and represent dynamic conformational changes across sampled datasets.

Initially, three scenarios were designed to test the tool. Each scenario represents a user case, demonstrating the toolkit's ability to address different research questions. These scenarios were created as user-friendly "recipes" that users can run directly, allowing for flexible testing of various aspects of MDSS's functionality. In addition to these standalone scripts, each scenario was also implemented as a Jupyter Notebook, enhancing accessibility and interactivity for all types of users.

Scenario 1: Subsampling for distribution similarity of RMSD

This scenario tested MDSS's ability to select the smallest subset of frames that retained the RMSD distribution of the original trajectory. The case is helpful for MD users seeking to reduce data volume without compromising structural information.

1. Purpose: This scenario was designed to subsample an MD trajectory containing different global protein conformations. The goal was to find the smallest subset of frames that retained a similar RMSD distribution to the original data.

2. Sampling strategy: Random sampling captured frames representative of the protein's entire conformational range.
3. Workflow:
    ◊ The tool first reads the trajectory and topology files and sets up the RMSD calculation for the entire trajectory.
    ◊ Random subsampling is performed at different sample sizes (0.25%, 0.5%, 1%, 2.5%, 5%, 10%, 20%, 25%, 50%.) specified by the user.
    ◊ MDSS computes the RMSD distribution for each sample size and compares it to the entire trajectory's RMSD distribution using the Bhattacharyya distance as the dissimilarity metric.
    ◊ The process iterates until a sample size produces a subsample with an RMSD distribution similar to the original, as determined by a user-defined threshold.

This scenario demonstrated MDSS's ability to preserve critical information of a trajectory through random subsampling, achieving significant data reduction while maintaining distributional similarity.

Scenario 2: Uniform sampling of pocket opening states

The second scenario explored MDSS's capability to capture a specific range of conformational states within a protein's binding pocket, particularly in systems where pocket geometries vary significantly.

1. Purpose: This scenario aims to obtain a subset of frames representing a broad range of pocket opening geometries by sampling frames that varied in RMSD relative to an open or closed reference state.
2. Sampling strategy: Uniform random sampling is applied to ensure even coverage of the range of pocket conformations observed in the full trajectory.
3. Workflow:
    ◊ MDSS reads the trajectory, topology, and selection criteria, specifying residues within the binding pocket.
    ◊ A reference structure establishes a baseline RMSD value for the open (or closed) state.
    ◊ MDSS calculates the RMSD distribution for pocket conformations in the full trajectory and stratifies the frames into intervals based on RMSD values.
    ◊ Uniform random sampling selects frames from each interval, ensuring a proportional representation across the full range of pocket conformations.

This scenario validated MDSS's ability to selectively capture structural diversity within a region of interest, demonstrating that MDSS could provide a representative subset of frames that covered the conformational landscape of the binding pocket.

Scenario 3: Weighted sampling based on conformational state frequency

In this scenario, MDSS's weighted sampling capabilities were tested to ensure an equal representation of all conformational states within the trajectory, including those less frequently observed. This approach is useful for users interested in capturing diverse frames that evenly span the conformational landscape rather than focusing solely on dominant states.

1. Purpose: By applying weighted sampling, the goal is to reduce the bias towards frequently occurring conformations and instead select a subset of frames that provides equal representation across the trajectory's conformational space.
2. Sampling Strategy: Weights are inversely proportional to the frequency of frames in each conformational state, ensuring that less frequent conformations are sampled more often than dominant ones.
3. Workflow:
    ◊ The trajectory and topology files are loaded, and RMSD values are calculated for each frame.
    ◊ MDSS analyses the RMSD distribution and generates a weighting vector where less frequent RMSD bins are assigned higher weights.
    ◊ Using this weighting vector, MDSS performs weighted sampling to select frames, resulting in a subsample that balances the representation of all conformational states, regardless of their original frequency.

This scenario demonstrates MDSS's flexibility in achieving a balanced representation of conformational states, making it particularly useful for studies requiring equal sampling of rare and common conformations.

Following the testing of MDSS through the initial three scenarios, additional advanced workflows were implemented in the cookbook to address more complex sampling and subsampling requirements. These advanced scenarios extend MDSS's utility by focusing on hierarchical subsampling and ML applications.

Advanced scenario: Machine learning prediction

In this scenario, MDSS was tested for its capability to reformat MD data and directly use them as inputs within ML workflows. This scenario demonstrates how subsampled MD trajectory data can be prepared for predictive modelling, enabling researchers to classify or predict specific protein conformations based on calculated properties. The workflow outlines a complete pipeline that takes subsampled data, applies ML algorithms, and evaluates model performance, showcasing MDSS's utility in ML contexts where MD data is used as input.

1. Purpose: This scenario illustrates the application of ML to predict specific conformational states within an MD trajectory using subsampled data as input. It enables researchers to classify conformations or states based on key structural features, facilitating the identification of patterns within protein dynamics.
2. In this example, the target prediction focuses on protein states categorised as "Open", "Closed" and "Noise" based on two geometric properties: the `RadiusOfGyration` and the `DistanceBetweenAtoms`. Labels for these states were defined based on expert knowledge (see section 5.1.3), with thresholds derived from density plots of these properties (Figure 5.6) This manual labelling serves as the ground truth for evaluating ML models.
3. Workflow:
   - ◊ Data preparation: The input data for this scenario consists of a pre-processed subsampled dataset of RMSD values (`ml_input.npy`) and a corresponding target label file (`target50.csv`) indicating the conformational states (Figure 5.6). *A posteriori* subsampling was performed using a random sampling strategy, ensuring that the dataset remained representative of the trajectory while reducing data volume. The Rg and inter-residue distance values were analysed to define the protein states, and density plots were used to visually confirm the separation of open, closed, and noise regions in the conformational space.
   - ◊ ML model training: Three ML models, Logistic Regression (LR), Random Forest (RM), and Support Vector Machine (SVM), were implemented and trained on the subsampled data. The data was divided into two sets: 30% for testing and 70% for training.
   - ◊ Model evaluation: Each model was evaluated using standard classification metrics:
     - o Accuracy score: Assesses the percentage of correct predictions on the test set.
     - o Confusion matrix: This matrix displays the classification model's performance by showing true positives, false positives, true negatives, and false negatives.

- Cohen's Kappa score: Evaluates the agreement between predicted and actual labels, adjusting for chance agreement.
- Classification report: Provides detailed metrics, including precision, recall, and F1-score for each class.

◊ Results saving: Model performance metrics were saved for further analysis, and trained models were saved using *joblib* for later use.

4.2 MDAutoMut tool

MDAM is a modular Python library developed to rationally redesign protein dynamics through mutation scanning. The tool aims to identify mutations that have the desired impact on protein dynamics. Mutation scanning is performed either systematically or through a heuristic approach. The toolkit is suitable for single, double, or multiple mutation scanning.

The work presented in the following section details the design, implementation and testing of MDAM.

4.2.1 Software design and core components

MDAM is designed as a fully automated, modular Python library that combines mutation engineering, generation of MD simulations, and evaluation of changes in protein dynamics. The tool is structured to explore and then identify mutations that have a desired impact on protein dynamics. Specifically, the tool can create workflows with mutation scanning until the system samples the desired dynamics in the system. The toolkit contains two core classes: `Protein` and `Mutation`, and Figure 4.7 displays the class (i.e. module) diagram of the tool. The tool imports `ProteinData` from the MDSS library.

Protein class

The `Protein` class uses the *MDAnalysis* framework [8] to load and manage the protein data based on the user's input files, including .xtc, .pdb or .gro formats. This class encapsulates structural and simulation data, allowing users to use pre-existing trajectory files or dynamically generate data within the MDAM workflow.

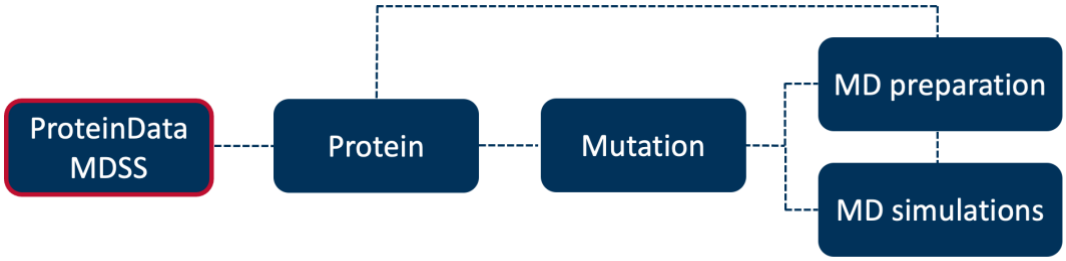


Figure 4.7 Class and module diagram of MDAM, depicting its integration with MDSS and modular approach for mutation engineering and dynamic simulation in MD workflows.

Mutation class

The `Mutation` class facilitates mutation engineering in MDAM, using *PyRosetta* [9] to introduce mutations while ensuring structural stability. Users provide a list of mutations in an input file, specifying the target residue positions and desired amino acid substitutions. The `output_mode` option allows mutations to be applied individually (single) or in combinations (multiple), enabling comprehensive testing of different mutation scenarios. When initialising a `Mutation` class object, users select the reference structure to insert the desired mutation from a protein trajectory frame (defaulting to the last frame if unspecified), which is then extracted and converted into a *PyRosetta* `Pose` object for mutation operations.

The method replaces the target residue for each mutation, repacks side chains, and calculates a full-atom energy score before and after mutation to assess structural stability. Specifically, the new residue is first generated using *PyRosetta*'s `ChemicalManager` within the `fa_standard` force field for full-atom calculations and is substituted at the target position. The method then repacks neighbouring side chains to refine local conformations, using the default scoring function within the same force field to ensure stability.

Workflows

MDAM includes three modular workflow scripts: `mutation_workflow.py`, `mdprep_workflow.py`, and `mdautomut_workflow.py`. While the `mutation_workflow.py` and `mdprep_workflow.py` scripts are designed to function independently as standalone workflows, the `mdautomut_workflow.py` integrates all modules from MDAM, facilitating the fully automated method for rationally redesigning protein dynamics.

The `mdprep_workflow.py` script prepares the protein system for MD simulations. This workflow manages essential steps such as solvation, in addition, and energy minimisation, creating a simulation-ready structure. Users can customise parameters, including force field selection and other simulation parameters (currently in the form of a .mdp file), allowing flexibility in adapting the workflow to various protein systems and research objectives.

The `mutation_workflow.py` script supports targeted mutation engineering by enabling the insertion of single, double, or multiple mutations. Utilising *PyRosetta* [9], this script offers control over mutation engineering, allowing users to specify desired amino acid changes to target specific changes in the system's dynamics. The flexibility to perform single and multiple mutations makes this workflow suitable for exploring various mutational impacts.

The `mdautomut_workflow.py` script provides an integrated end-to-end pipeline, combining system preparation, mutation engineering, and evaluation of change in dynamics within a single workflow. This script automates the entire mutational scanning process, aiming to identify the impact of specified mutations on protein dynamics.

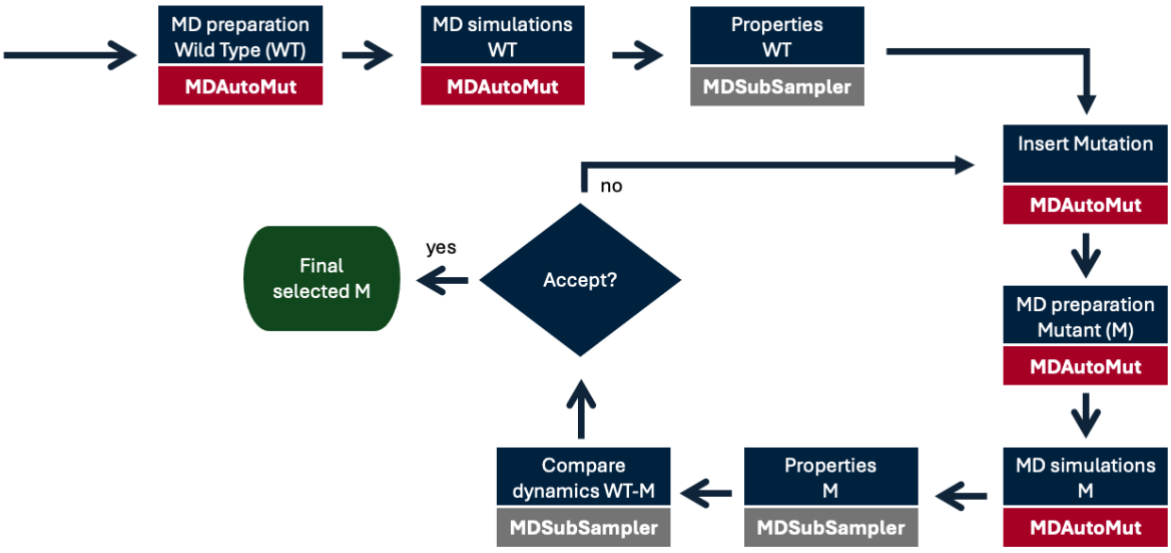


Figure 4.8 A simplified version of the entire workflow of MDAM, showing the step-by-step process of system preparation, mutation engineering, and dynamic simulation for the rational redesign of protein dynamics. The decision point depends on the acceptance criterion, or the predefined threshold and it can either end the loop or continue exploring more mutations until the desired dynamics are achieved.

Figure 4.8 illustrates a simplified version of a workflow within MDAM, outlining the sequential steps from system preparation to mutation engineering and dynamic simulation. The flow starts by preparing the protein system for MD simulations for the WT, introducing mutations, simulating the mutants, and then comparing the WT with each mutant to check if the desired

dynamics are achieved. A simple acceptance criterion or a predefined threshold is used to enable the automation of the scanning process. This simplified workflow is easily extendable due to MDAM's module structure.

Utility functions

In addition to the core classes and workflow scripts, MDAM includes a set of utility — modules `args_utils.py`, `file_io.py`, `mutation_utils.py`, `protein_utils.py` — that support essential tasks within the workflow. These modules handle command-line argument parsing, file management, mutations-specific operations and protein structure manipulation, providing the flexibility and customisation needed for efficient mutation screening and analysis of protein dynamics.

Property and dissimilarity calculations

To assess how mutations affect protein dynamics, MDAM integrates with MDSS for property calculation and comparison between wild-type (WT) (or target property) and mutated structures. Specifically, it calculates and compares geometrical properties between WT against each mutant, enabling a quantitative assessment of how each mutation impacts protein dynamics. Using MDSS's Property and `GeometricProperty` modules, MDAM can compute key properties such as `RMSD`, `RadiusOfGyration`, and `COMDistance`, which can serve as indicators of changes in protein conformations.

MDAM can apply MDSS's `Dissimilarity` module for each mutant to quantify differences between WT and mutant property distributions. This dissimilarity analysis uses statistical measures (e.g., Bhattacharyya distance) to evaluate the impact of mutations, allowing the toolkit to identify those mutations that most effectively alter the protein's dynamics. Additionally, the `plotting.py` module visualises the differences in distributions for the selected properties between the WT (or target property) and each mutant, representing mutation effects on dynamics.

Logging and configuration

The `log_setup.py` module provides real-time logging across the full MDAM workflow, capturing each step and recording warnings or errors encountered during execution. The logs offer users detailed insights into each step's progress and status, enabling efficient troubleshooting and validation of results.

4.2.2 Functionality

MDAM provides a flexible and user-friendly interface for mutation-based analysis of protein dynamics. The toolkit is designed with modular workflows and customisable options, allowing researchers to define specific mutations, configure MD simulations and assess changes in dynamics through properties.

Mutations and mode

To begin the analysis, users define a list of target mutations in an input file, specifying the position and the amino acid change. MDAM supports single, double or multiple mutations, enabling users to test a range of mutation scenarios systematically. Through the `output_mode` setting, users control whether mutations are applied individually or in combinations, providing flexibility in exploring all mutation impacts on protein dynamics.

Simulation configuration

In MDAM, the entire simulation configuration process is automated into a stepwise script. Users predefine the steps and parameters before execution of this script. Once the mutations are defined, users configure the MD simulation parameters within MDAM. The toolkit provides options for customising the parameters for system preparation and production of MD data. The preparation process begins with force field selection during the pdb2gmx step, where users can choose from several forcefields, such as AMBER [71] or CHARMM [64], depending on the system choice. Additionally, they can specify the water model, such as TIP3P [86], to solvate the box.

Following that, users define the simulation box type, such as cubic or truncated octahedron, and set the distance between the protein and the edges of the box to ensure adequate solvation. The system is then solvated, and ions are added to neutralise the system's charge. Researchers can also adjust the ionic concentration, controlling the electrostatic environment.

Furthermore, users can configure the energy minimisation steps by adjusting parameters to control the algorithm used and the convergence criteria and define the strength of positional restraints on the protein. Similarly, users define the choices for the temperature and pressure equilibration (NVT and NPT ensembles). In each stage, users can control the duration, thermostat settings and barostat options through .mdp files. Finally, the length of the simulation is defined before it goes into the production phase.

For the proof-of-concept validation of the MDAM toolkit, pre-configured .mdp files and all necessary parameters for the ADK system are provided, including forcefield selection and simulation settings (see section 3.2.3).

Mutation scanning workflows: systematic and heuristic approaches

MDAM supports two primary mutation scanning workflows: systematic and heuristic. These approaches address different research needs, balancing thorough exploration and computational efficiency. The terms “systematic” and “heuristic” are defined in this context to differentiate the workflows: The systematic workflow involves a complete exploration of the mutation list, testing every possible mutation. In contrast, the heuristic workflow is tailored for scenarios where the mutation space is too large for exhaustive exploration, requiring selective prioritisation of mutations for testing.

The systematic workflow best suits smaller mutation sets where exhaustive exploration is feasible. MDAM generates and evaluates all possible mutations across specified positions in this approach. For instance, a single mutation scan mutates a target position to all 20 standard amino acids. In contrast, for double mutations, every pairwise combination of amino acids across two positions is tested. This approach is ideal for obtaining detailed insights into specific mutations’ influence on protein dynamics. However, as the number of mutation sites increases, the computational demand grows exponentially, making systematic scanning challenging for larger trajectories or multiple mutation sites.

To address these limitations, MDAM includes an adaptive heuristic workflow that reduces the search space through a selective and iterative exclusion strategy. This approach is beneficial for complex systems with many potential mutations or limited computational resources. The heuristic workflow uses a table of mutation combinations. For example, starting with a random pair of mutations (e.g., AA, AG, GA), MDAM evaluates their effects on dynamics. If a combination (e.g., AG) fails to produce the desired outcome, the tool excludes all mutations involving amino acid “A” at the first position from further testing. This exclusion strategy systematically narrows the search to focus on promising mutations, rapidly filtering out ineffective combinations. As a result, the heuristic approach accelerates the identification of impactful mutations, making it a practical choice for larger mutation sets.

Synthetic distribution as a target for desired dynamics

A critical feature of MDAM is its ability to guide the mutation evaluation towards sampling specific dynamic states, which may not naturally occur in the WT. MDAM provides a feature

that allows the user to specify a desired distribution for the values of a selected MDSS property and use this distribution as the target for redesigning protein dynamics. A synthetic distribution of the target CV representing the desired dynamics in ADK was generated to test this functionality in a proof-of-concept study. Specifically, the goal was to sample a closed state in ADK, represented by a specific value of the `COMDistance` between its LID and AMPbd domains.

The synthetic distribution was generated by introducing a second peak to the `COMDistance` histogram, corresponding to values expected for the closed state alongside the existing peak for the open state observed in the WT (Figure 4.9). This artificial bimodal distribution mimicked the dynamics of a protein system with both open and closed states, allowing a direct comparison of the mutants' dynamics with the target.

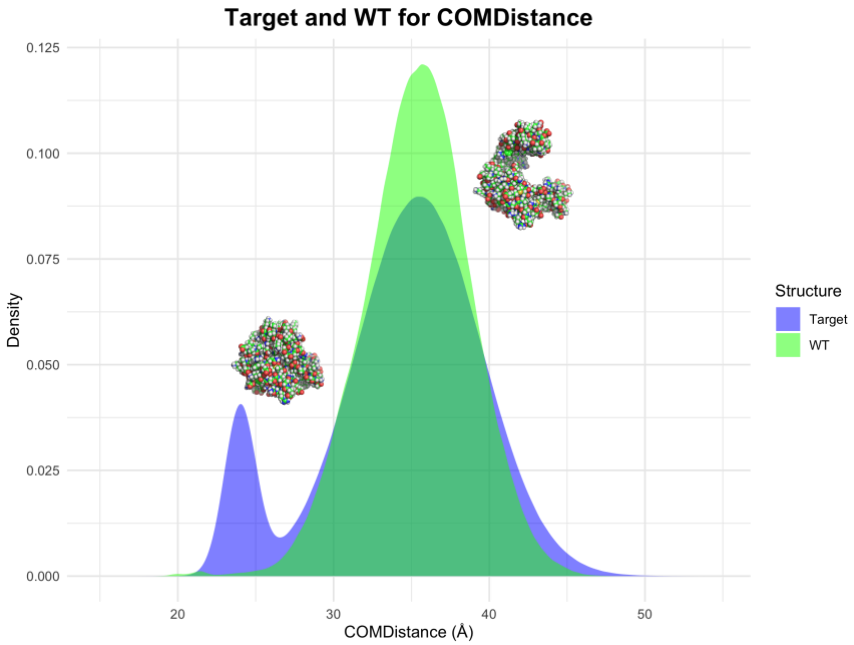


Figure 4.9 Synthetic distribution for desired dynamics in ADK. The COMDistance density distribution of the WT (green) for ADK system, representing the natural dynamic state with a single peak corresponding to the open conformation. The synthetic target distribution (purple) was generated by introducing a second peak corresponding to the desired closed conformation. This artificial bimodal distribution mimics a system with both open and closed states, enabling a direct comparison of mutant dynamics to the target. Representative structural snapshots of the open (top) and closed (bottom) conformations are overlaid on the density plots.

Reducing combinational complexity

To further manage combinatorial complexity, MDAM's heuristic workflow can group amino acids into functionally similar clusters, reducing the overall number of combinations while maintaining diversity within the mutational landscape. Since testing all 400 combinations for

two positions would be computationally intensive, this grouping method simplifies the search space. Each cluster represents a set of amino acids with shared physicochemical properties, such as size, polarity, or charge. For instance, small nonpolar amino acids (like A, G, V, and P) are represented by a single amino acid, such as Alanine, commonly used in mutagenesis studies. This approach ensures that key characteristics are still sampled while focusing computational resources on representative amino acids.

By combining functional grouping and adaptive exclusion, MDAM makes exploring large mutational landscapes feasible and ensures efficient and targeted mutation scanning.

Comparison of dynamics

To automatically assess how mutations affect protein dynamics, MDAM calculates specific properties (user's choice) for the WT and all mutants and then compares their property distributions. The selected properties should capture key aspects of the protein's structural and dynamic behaviour. The primary objective is to measure how closely each mutant's property distribution aligns with the desired target dynamics. This comparison can be carried out by either using the WT distribution as a reference or, in cases where a specific target dynamic state is desired, by comparing it to a predefined target distribution.

The overlap between the WT (or target) and mutant distributions is measured via distance metrics from MDSS. MDAM allows users to define a threshold for reference when calculating the distance between distributions. Suppose the calculated distance for a mutant is below the threshold. In that case, the mutation (or set of mutations) produces dynamics that closely resemble the desired state (e.g. the closed state of ADK).

When the distance between the WT or target and mutant distributions meets the specified threshold, MDAM identifies these mutations as those with the desired effect on protein dynamics. By iteratively scanning through each mutation or combination of mutations and comparing their impact on dynamics, the toolkit narrows down on specific changes that drive the protein towards the targeted dynamics.

4.2.3 Software implementation and accessibility

MDAM was developed as an open-source Python library, with the project structured and managed through GitHub (https://github.com/alepandini/MDAutoMut) and developed primarily within Visual Studio Code [181]. The toolkit is organised to facilitate reproducibility and ease

of installation by leveraging the Poetry [179] package manager. Poetry enables streamlined management of dependencies, ensuring that MDAM can be easily set up with all required libraries across different environments. The configuration of dependencies is specified within the `pyproject.toml` file (Figure 4.10), simplifying the installation process and maintaining consistency in the software environment.

The MDAM environment requires Python version 3.9.1 or later, and it integrates multiple dependencies essential for mutation scanning and generation of MD simulations. The primary dependencies include gmxapi version 0.4.2 for interfacing with GROMACS, *PyRosetta*, and *MDAnalysis*. Additionally, MDSS supports property and dissimilarity calculations, while the *seaborn* library facilitates data visualisation, particularly in the analysis of property distributions. *PyRosetta*, which requires a specific license, was obtained and installed to enable mutation modelling.

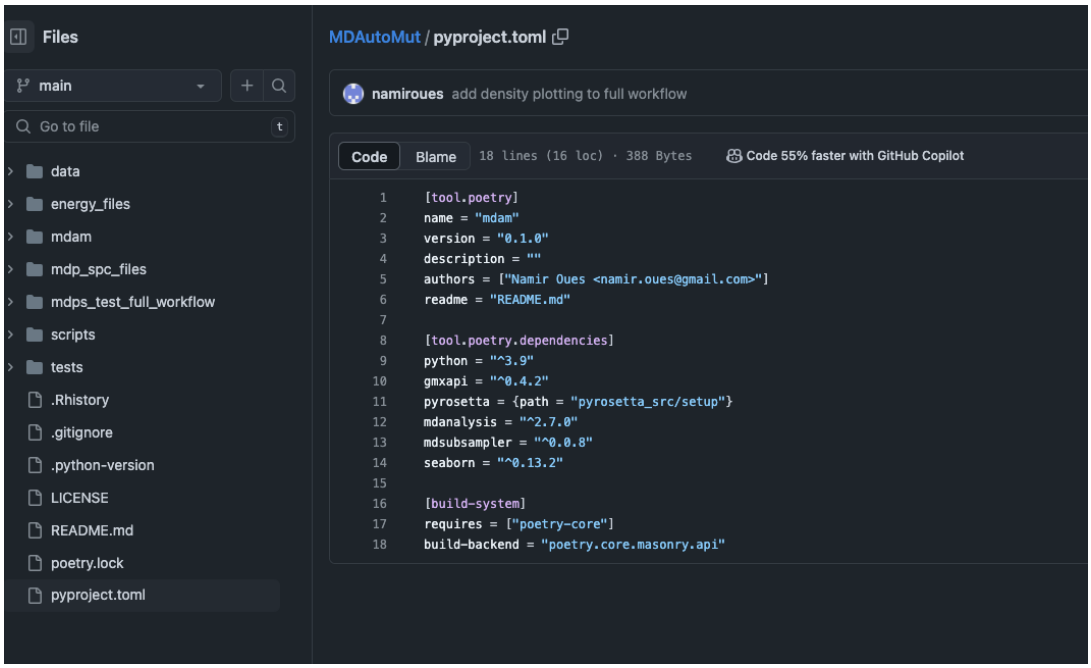


Figure 4.10 MDAM's pyproject.toml file. This configuration file defines the MDAM project settings and dependencies within the Poetry environment, streamlining the installation process and ensuring consistency across systems. Poetry manages both the primary dependencies for MDAM's core functions and additional dependencies for optional features, enhancing reproducibility and ease of use.

Usage and example command-line workflow

MDAM contains a command-line interface that allows users to define mutation lists, configure MD parameters, and initiate mutational scanning workflows directly. An example command-line workflow is illustrated in Figure 4.11, which demonstrates selecting specific input parameters, setting up the environment, and generating desired outputs. The figure highlights

the key steps, from defining mutations to initiating simulations, making it accessible for novice and experienced users.

```
..opment/lab/MDAutoMut [main|●13…7] ››› python mdam/mutation_parser.py --help
usage: mutation_parser.py [-h] [--traj TRAJECTORY_FILE] [--top TOPOLOGY_FILE]
                          [--pdb PDB_FILE] [--frame-number FRAME_NUMBER]
                          [--mutation-file MUTATION_FILE] --output-folder
                          OUTPUT_FOLDER --output-mode OUTPUT_MODE --prefix
                          FILE_PREFIX

MDAutoMut

optional arguments:
  -h, --help            show this help message and exit
  --traj TRAJECTORY_FILE
                        the path to the topology file
  --top TOPOLOGY_FILE   the path to the topology file
  --pdb PDB_FILE        the path to the pdb file
  --frame-number FRAME_NUMBER
                        the frame number to perform the mutation
  --mutation-file MUTATION_FILE
                        the .txt file with a list of mutations to be performed
  --output-folder OUTPUT_FOLDER
                        the path to the output folder
  --output-mode OUTPUT_MODE
                        the mode of single or multiple mutations
  --prefix FILE_PREFIX  the prefix for output files
```

```
..opment/lab/MDAutoMut [main|●13…7] ››› python mdam/mdprep_parser.py --help
usage: mdprep_parser.py [-h] --prefix FILE_PREFIX --system-name SYSTEM_NAME
                        [--mutation MUTATION] --outpath-directory
                        OUTPATH_DIRECTORY [--pdb-file PDB_FILE]
                        [--spc216-gro SPC216_GRO] [--ions-mdp IONS_MDP]
                        [--em-1-mdp EM1_MDP] [--em-2-mdp EM2_MDP]
                        [--em-3-mdp EM3_MDP] [--nvt-1-mdp NVT1_MDP]
                        [--nvt-2-mdp NVT2_MDP] [--nvt-3-mdp NVT3_MDP]
                        [--nvt-4-mdp NVT4_MDP] [--nvt-5-mdp NVT5_MDP]
                        [--nvt-6-mdp NVT6_MDP] [--npt-1-mdp NPT1_MDP]
                        [--npt-2-mdp NPT2_MDP] [--prod-mdp PROD_MDP]

mdprep

optional arguments:
  -h, --help            show this help message and exit
  --prefix FILE_PREFIX  prefix for file naming
  --system-name SYSTEM_NAME
                        system name for file naming
  --mutation MUTATION   mutation to be performed
  --outpath-directory OUTPATH_DIRECTORY
                        directory for output files
  --pdb-file PDB_FILE   the path to the pdb file
  --spc216-gro SPC216_GRO
                        the path to the spc216.gro file for solvation
  --ions-mdp IONS_MDP   the path to the ions.mdp file for adding ions step
  --em-1-mdp EM1_MDP    the path to mdp file for grompp_em_1 step
  --em-2-mdp EM2_MDP    the path to mdp file for grompp_em_2 step
```

Figure 4.11 Parser help interface for MDAM library. The Linux command line contains options and required user arguments for both mdprep.py and mutation.py workflows.

Licensing and contribution

MDAM is released under the GPL-3.0 license, ensuring the toolkit remains open-source and accessible to the broader scientific community. This license allows researchers and developers to freely use, modify, and distribute the software while requiring that any derivative work remains open source under the same license, allowing collaborative improvement of the toolkit.

The project is hosted on GitHub, and researchers and developers are encouraged to report issues, suggest improvements, and contribute directly to the codebase. A `README` file provides detailed installation instructions, usage guidelines, and examples to help new users get started.

4.2.4 Testing

MDAM's capability to identify mutations that impact protein dynamics was tested through a proof-of-concept study conducted using ADK as a model system (see section 3.2.3). ADK is known for its significant dynamic allostery, with distinct open and closed states facilitated by the flexible LID and AMP-binding (AMPbd) domains. This example was chosen because ADK's dynamic behaviour has been extensively studied, providing detailed experimental insights into the effects of specific mutations on the protein's conformational states. Song et al. [165] demonstrated how LID domain mutations, such as VAL135GLY and VAL142GLY, enhance interactions with the AMPbd domain, shifting ADK towards a more closed conformation (Figure 4.12). These specific mutations were set as a "target" dynamic outcome within the scanning process, enabling an evaluation of whether MDAM could identify these known mutations among all 20 other possible changes in each position.

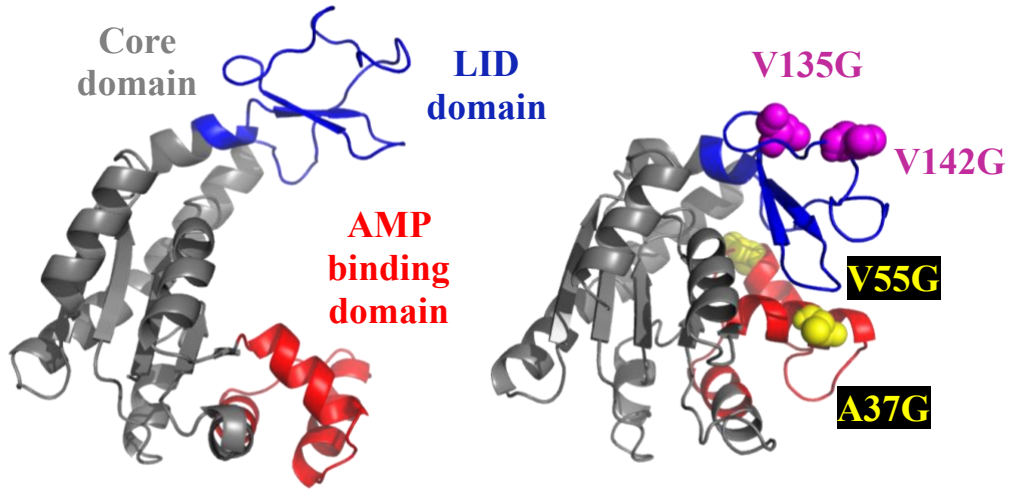


Figure 4.12 Structural illustration of ADK highlighting the domains and mutations. The left structure shows the WT in the open apo state, with the core domain in grey, the AMP-binding domain (AMPbd) in red, and the LID domain in blue. The right structure depicts the identified LID mutations (V135G and V142G, marked in magenta spheres), which enhance interactions between the LID and AMPbd domains, favouring a closed conformation. Additional mutations on the AMPbd domain (A37G and V55G, marked in yellow spheres) were included for completeness but were not the focus of this study. This representation demonstrates the successful identification of dynamic state-altering mutations using MDAM, validated by experimental findings from Song et al. [165].

In this scenario, the MDAM toolkit scanned with two specific positions in the LID domain, VAL135 and VAL142, but without prior knowledge of the exact mutation required at each position. By employing systematic and heuristic scanning approaches, MDAM aimed to find mutations that produced the desired conformational shift towards a closed state, mimicking the findings of Song et al. [165].

MDAM's mutation scanning process was optimised, and combinational complexity was reduced by grouping amino acids into five categories based on shared physicochemical properties: small non-polar, large non-polar, polar uncharged, positively charged, and negatively charged. This categorisation allowed for a focused selection of representative amino acids from each group, minimising the number of combinations while retaining a diverse range of mutational effects. Selecting one representative amino acid per group reduced the complexity from 20 x 20 = 400 to 5 x 5 = 25 combinations, making the scan computationally feasible.

The following representatives were chosen for each group:

- Group 1 (Small non-polar): Glycine (G)
  Glycine is a small, flexible amino acid commonly used in mutagenesis studies because it can introduce conformational flexibility.

- Group 2 (Large non-polar): Leucine (L)
  Leucine, a frequently occurring amino acid, represents typical hydrophobic characteristics and is often found in protein cores.

- Group 3 (Polar uncharged): Serine (S)
  Serine is small and polar, often used in mutagenesis studies for its ability to participate in hydrogen bonding while remaining uncharged.

- Group 4 (Positively charged): Lysine (K)
  Lysine has a long side chain and basic properties, making it a typical representative in studies exploring charge effects on protein structure and dynamics.

- Group 5 (Negatively charged): Aspartic Acid (D)
  Due to its small size and charge, Aspartic acid is frequently used as a representative acidic amino acid in mutagenesis studies.

Using these representative amino acids, the toolkit explored a manageable subset of the mutation landscape while capturing key physicochemical variations. This approach helped to perform the proof-of-concept study, where the objective was to identify known mutations (e.g., GLY substitutions) within the LID domain that induce a shift towards the closed conformation. This systematic reduction allowed for thorough exploration and computational efficiency, ensuring that the most impactful mutations could be identified effectively.

For both the systematic and heuristic scanning processes, a list of potential mutations at the VAL135 and VAL145 positions was prepared to identify the set of mutations favouring the target closed conformation of ADK. Each scanning approach was set up to evaluate these mutations by generating the new mutated structures, running MD simulations, and comparing the dynamics of each mutant with the WT of ADK and the synthetic target distribution Each mutation was evaluated based on the `COMDistance` property between the LID and AMP-binding domain. This measure typically shows a bimodal distribution representing these two conformations (open and closed state of ADK) (Figure 4.9).

Systematic scanning approach

In the systematic approach, MDAM iteratively evaluated all 25 combinations of the chosen representative amino acids (GLY, LEU, SER, LYS, and ASP) at the VAL135 and VAL142 positions. The `COMDistance` between the of LID and AMPbd domains was calculated and compared to the synthetic target distribution for each combination. This target distribution represented a bimodal pattern characteristic of a transition to the closed state, with peaks reflecting both open and closed conformations (Figure 4.9). MDAM used the Bhattacharyya distance to quantify the similarity between each mutant's `COMDistance` distribution and this property's WT or target distribution. Mutations producing a Bhattacharyya distance below the 0.05 threshold were successful, indicating that the induced dynamics closely matched the desired closed conformation.

Heuristic scanning approach

In the heuristic approach, MDAM applied an adaptive selection strategy to streamline the scanning process further. MDAM initially tested random pairings of these residues at both positions, starting with the five representative amino acids. Based on the Bhattacharyya distance outcome, the heuristic approach eliminated less promising combinations (those with distances well above the 0.05 threshold), thereby reducing the search space and focusing on combinations that showed potential alignment with the target distribution. This iterative

exclusion strategy allowed for more efficient exploration of the mutation landscape, accelerating the identification of impactful mutations while maintaining computational feasibility.

4.3 MDAutoPredict tool

MDAP is a standalone toolkit developed to perform predictions using MD trajectories and ML methods. It uses a simplified representation of proteins as input data—Cartesian coordinates (x, y, z) of atoms—and generates frame-level predictions. MDAP can fully integrate with both MDSS and MDAM frameworks while also functioning independently for property prediction tasks.

MDAP's core contribution lies in its implementation of a modular, reusable Python package that integrates ML with MD workflows. The tool provides functionality to handle noisy MD data, streamline the prediction pipeline, and evaluate the effectiveness of ML methods for property prediction.

4.3.1 Software design and core components

MDAP is implemented as a modular Python toolkit that contains two primary classes: `MDTrajLearner` and `MLProperty`.

MDTrajLearner class

The `MDTrajLearner` class manages the ML pipeline. It generates input features by transforming trajectory data into a 2D matrix representation, trains ML models on this data, and evaluates their performance. The input features are derived from space atoms' Cartesian coordinates (x, y, z), representing the protein's conformations. The matrix representation ensures compatibility with standard ML workflows while maintaining flexibility for future integration with alternative data representations, such as graph embeddings.

The `generate_and_save_matrix` method integrates with the MDSS `ProteinData` class (see section 4.1.1) and can convert MD trajectory data into a flattened matrix format, where each row corresponds to a frame, and the columns capture the spatial arrangement of atoms. This matrix is saved in .npy format, allowing efficient reuse in subsequent analyses. The `train_and_test` method can train user-specified ML models and store the trained model

as a binary object. Performance metrics such as accuracy, confusion matrices, and classification reports are logged and saved for further evaluation.

MLProperty class

The `MDTrajLearner` class extends the `ProteinProperty` class in MDSS, enabling ML predictions to label trajectory frames or return predicted values for the desired properties. It incorporates a trained ML model to predict frame-level properties directly from the input data. The `calculate_property` method applies the model to predict property values for all frames, storing the results in the `property_vector` attribute. The class also provides functionality for assigning and exporting labels to trajectory frames.

Thanks to the modularity of these two classes, the MDAP toolkit can operate independently or be wholly integrated with other tools like MDAM and MDSS. The design allows users to test different ML methods, preprocess noisy trajectory data, and annotate frames with predicted labels, all within a cohesive framework.

4.3.2 Functionality

MDAP is a supervised ML tool that analyses the trajectory data of MD simulations to build a predictive model. It can classify protein conformational states into distinct categories based on their structural features. The tool provides a flexible ML framework for classifying protein conformational states or other properties from MD simulations.

MDAP takes pre-processed trajectory data and structural labels as input. The output (i.e. the target variable) consists of predicted conformational states.

Data transformation

MDAP leverages MDSS to preprocess MD trajectories into an ML-compatible format. MDSS's ProteinData module can transform trajectories from 3D atomic coordinate matrices into 2D tabular datasets. In this representation, rows correspond to trajectory frames, while columns encapsulate cartesian coordinates, making the data readily usable for supervised learning algorithms.

Target variable definition

Labels for conformational states are derived through a density-based clustering approach (see section 5.4.1). This method projects the trajectory data onto a two-dimensional principal component analysis (PCA) space, where high-density regions correspond to free energy minima associated with biologically relevant conformational states (e.g., open, closed, intermediate) (Figure 5.33). Frames outside these clusters are classified as non-states, ensuring comprehensive labelling of the trajectory.

Supervised classification framework

MDAP's classification framework is modular and extensible, allowing researchers to integrate and apply several ML models to their data. The `methods.py` module provides a dictionary-based structure for quickly adding new classifiers, enabling users to adjust the toolkit to their specific research questions and systems.

4.3.3 Software implementation and accessibility

MDAP was developed as an object-oriented Python library integrated with well-established packages such as *MDAnalysis, numpy, scikit-learn, joblib,* and MDSS. MDAP was developed and tested using Visual Studio Code [181] and was packaged with Poetry [179]. MDAP is compatible with Python 3.9.

Usage and example workflow

The `workflow.py` script allows users to easily modify input paths, model parameters, and output settings. Its modular design ensures ease of use and extensibility to integrate additional ML models or features, ensuring that MDAP can adapt to different research questions.

Licensing and Contribution

MDAP is currently being finalised for open-source release. Once complete, the GitHub repository will be made public, allowing users to contribute to its development. The repository will include the codebase, detailed documentation, and example workflows to guide users in applying MDAP to their research.

4.3.4 Testing

The functionality and performance of MDAP were validated to ensure its robustness and reliability in analysing MD trajectories and predicting protein conformational states. Testing focused on evaluating MDAP's ability to classify conformational states using supervised ML models, ensuring the tool could handle complex datasets effectively and provide accurate predictions.

The testing was initially conducted using the second replica of DM (1 μs) on ADK (see 3.2.4 and 3.3) generated for the MDAM testing. The trajectory was subsampled at 40 ps intervals to produce a manageable dataset of 25,001 frames while maintaining a representative view of the system's conformational dynamics (open and closed state of ADK).

Six machine learning models were selected for comparison: Decision Tree, Random Forest, Gradient Boosting, Support Vector Machine (SVM), Logistic Regression, and Multilayer Perceptron (MLP). These techniques, which involve tree-based, ensemble, linear, and Neural Network-based techniques, were selected for their effectiveness in classifying ML predictions. They offer a balanced performance evaluation across different datasets.

The trajectory dataset was split into training (70%) and testing (30%). Each model was trained. Standard metrics like precision, recall, and F1 Scores were used to compare their performances. This comparison evaluated the models' capacity to categorise dominant and transitional conformational states in detail.

The outputs from MDAP were validated through comprehensive visual and statistical evaluations, including confusion matrices, calibration curves and precision-recall graphs. These validation steps provided critical insights into the reliability and confidence of the models, ensuring that MDAP's predictions aligned with the underlying biological dynamics.

MDAP was demonstrated as a flexible and reliable tool for analysing protein conformational states in MD datasets by integrating diverse ML methods and rigorous evaluation metrics.

4.4 Summary

This chapter presented the design, implementation, and testing of MDSS, MDAM, and MDAP. By integrating these toolkits into a unified framework, the chapter demonstrated how these tools address key challenges in subsampling and preprocessing MD data, rational redesign of protein dynamics through mutation scanning, and applying ML to predict protein states. The combined approach highlights how design, implementation, and testing ensure robust,

reproducible, and scalable workflows. These tools collectively form a reliable pipeline that advances the field of computational protein engineering by automating the process of redesigning protein dynamics.

5 Results

This chapter presents the research findings, focusing on the MD simulations conducted on ADK, the model protein system. The results are structured into four key sections. The first section presents the results from MDSS, a tool developed to perform *a posteriori* subsampling of MD trajectories while preserving relevant geometric properties. The second section focuses on the validation data generated as part of the proof-of-concept study for the MDAM framework, which demonstrates the dynamic behaviour of the ADK system for the WT and other mutants. The results for MDAM are presented in the third section, particularly its automated workflows for introducing mutations and assessing their impact on protein dynamics. Finally, the fourth section provides the results for MDAP by performing a classification prediction of protein states.

5.1 MDSubSampler results

Three use-case scenarios are presented to demonstrate the potential uses of MDSS: random sampling for size reduction, pocket sampling for ensemble docking and sampling by most frequently observed conformations. An example of an advanced scenario was implemented, machine learning prediction, to demonstrate the tool's expansion into more complicated workflows. The testing of MDSS was conducted using a 1 µs trajectory with a timestep of 2 fs, resulting in datasets comprising approximately 100,000 frames. Chapter 3 (section 3.2.2) provides information regarding the simulations. The simulation could sample both ADK states, so the trajectory contains frames with a close protein conformation.

5.1.1 Scenario: random sampling for size reduction

In this scenario, MDSS was tested for its ability to perform random subsampling of MD trajectory data, specifically focusing on preserving the RMSD distribution of Cα atoms in a protein structure. The goal was to identify the smallest subset of frames that retained the key structural features of the original trajectory while reducing dataset size.

The RMSD over Cα atoms was calculated for each frame of the ADK trajectory relative to a reference structure (open conformation after minimisation and thermalisation in the solvent of the crystallographic structure). All frames were superimposed onto the reference structure before computing RMSD to remove rigid roto-translation and estimate structural variability only from intrinsic dynamics. The distribution of the RMSD value of Cα atoms along the full

trajectory of the ADK system, comprised of 100,000 frames, reveals a bimodal distribution indicative of two dominant conformations: a closed state and an open state of the ADK LID domain (Figure 5.1). The preservation of the bimodal RMSD distribution is critical because it reflects the dynamic transition between the open and closed conformations of the ADK LID domain, which are functionally significant for substrate binding and product release.

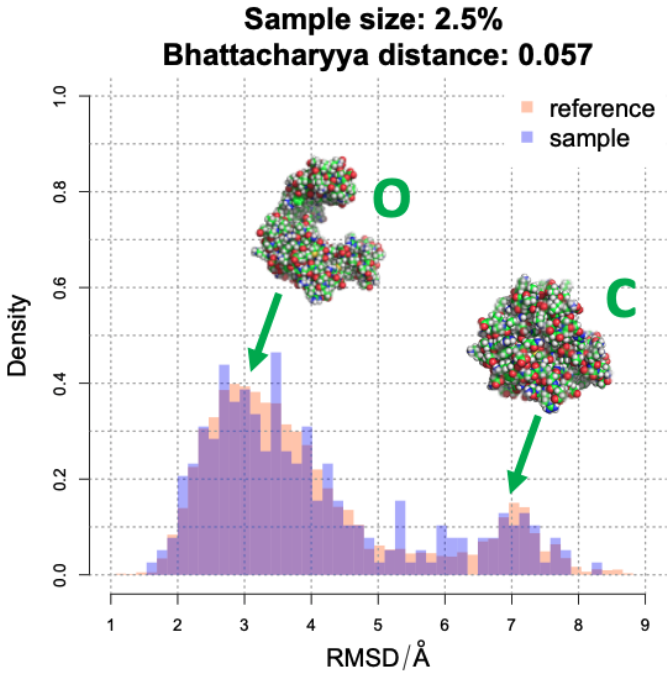


Figure 5.1 RMSD distribution for random sampling at 2.5% of the total trajectory frames (100,000 frames reduced to 2,500 frames). The comparison between the full trajectory (orange) and the subsampled set (purple) demonstrates that 2.5% preserves the bimodal distribution of RMSD values corresponding to the two dominant conformational states of the ADK protein. These states are identified as the "closed" (RMSD ~7.05 Å) and "open" (RMSD ~3.00 Å) conformations of the ADK lid domain. The Bhattacharyya distance used to quantify the similarity between distributions is minimal (0.057), confirming that the subsample successfully retains the key structural dynamics of the original trajectory while achieving significant data reduction.

The performance of the random sampling technique for size reduction was accessed through the following: subsets of the trajectory were selected at varying sizes (0.25%, 0.5%, 1%, 2.5%, 5%, 10%, 20%, 25%, and 50% of the total 100,000 frames). The distributions of RMSD values for each subset were compared to the full trajectory using Bhattacharyya distance as a dissimilarity metric. The Bhattacharyya distance is a statistical metric that measures the difference (i.e. similarity) between two probability distributions. Lower values of the measure indicate greater similarity between the two distributions.

The comparison across all subsets highlights a trend: as the sample size increases, the RMSD distribution of the subsampled data progressively converges with that of the reference (i.e. full) trajectory (Figure 5.2). While smaller subsets such as 0.25% and 0.5% (250 and 500 frames,

respectively) showed notable dissimilarity (Bhattacharyya distance values of 0.574 and 0.337, respectively), larger sample sizes displayed reduced dissimilarity (Figure 5.2). Specifically, a 2.5% sample size (2,500 frames) achieved a Bhattacharyya distance of 0.057, indicating preservation of the bimodal RMSD distribution with both peaks for open and closed states while maintaining a substantial reduction in data size (Figures 5.1, 5.2). Given the computational cost and storage requirements of handling 100,000 frames, reducing the dataset to 2.5% (2,500 frames) offers an excellent advantage for downstream analyses, such as ML pipelines or ensemble docking studies, without compromising structural accuracy.

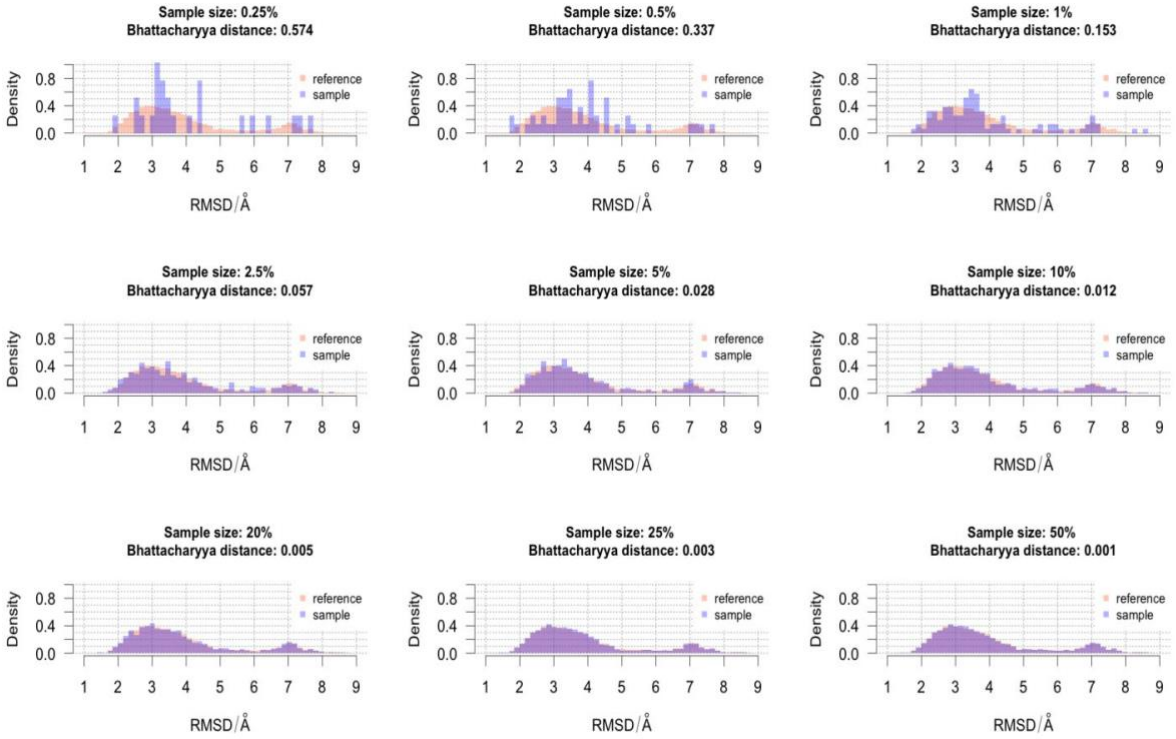


Figure 5.2 Presentation of results for "Random sampling for size reduction" scenario: The distributions of RMSD are compared over the coordinates of Cα atoms between the original and subsampled trajectories of the following sample sizes: 0.25%, 0.5%, 1%, 2.5%, 5%, 10%, 20%, 25%, and 50%. The Bhattacharyya distance was the dissimilarity measure selected to calculate the difference between the two distributions for each sample size. A sample of 2.5% is the smallest subset for which the shape and peak location of the distribution of RMSD is preserved.

5.1.2 Scenario: pocket sampling for ensemble docking

The second scenario evaluated the ability of MDSS to capture structural diversity in a protein's binding pocket, ensuring even representation of conformations across the range of pocket

openings. This scenario could be useful in ensemble docking studies, where an accurate representation of pocket geometries enables effective ligand binding analyses.

The RMSD values of Cα atoms for the binding pocket LID (residues 120-160) were calculated for all frames of the ADK's trajectory after fitting to the reference structure. The resulting RMSD distribution revealed a continuous range of conformational states, from the fully closed to the open lid. Uniform sampling was done by dividing the range of RMSD values into 200 intervals, and 10% of the frames within each interval were selected randomly. This uniform sampling strategy ensured that the subset of frames proportionally covered the entire range of pocket openings.

The effectiveness of this approach was validated by comparing the RMSD distributions of the subsampled trajectory and the original full trajectory. As shown in Figure 5.3, the RMSD histogram of the subsampled set (purple) equally spans the range of values in the original trajectory (orange) (Bhattacharyya distance is 0.127), confirming the proportional representation of the pocket conformations. The result demonstrates that the uniform sampling method effectively preserves the structural representations within the binding pocket while reducing the dataset size. Such equal sampling of pocket geometries is beneficial for ensemble docking analyses, where accurate and diverse conformations are essential for predicting ligand-binding interactions [182].

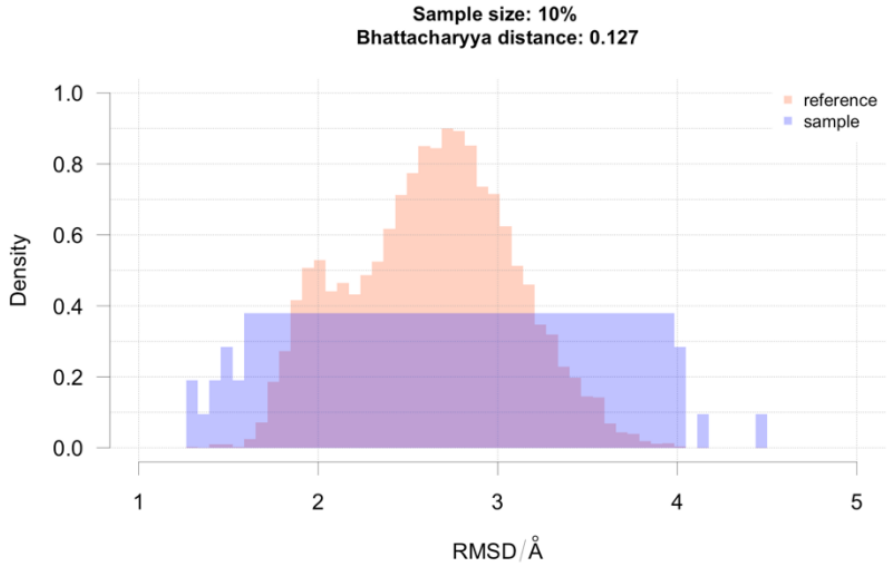


Figure 5.3 Summary result for "Uniform sampling of pocket opening for ensemble docking" scenario. RMSD over the Cα atoms of the ADK LID was calculated for all frames. The range of RMSD values from closed to open state was divided into 200 intervals, and for each interval, a random sample of 10% of frames was selected. This set of frames equally samples the range of possible openings for the protein's binding site. The difference between the sampled and original distributions was calculated using Bhattacharyya distance.

5.1.3 Scenario: sampling by most frequently observed conformations

The third scenario evaluated MDSS's ability to generate a representative subsample that captures dominant and less frequent conformational states within an MD trajectory. By prioritising frames based on the frequency of reference property, this approach allows for a more balanced representation of the trajectory's structural states, removing any bias toward the most frequent confirmation.

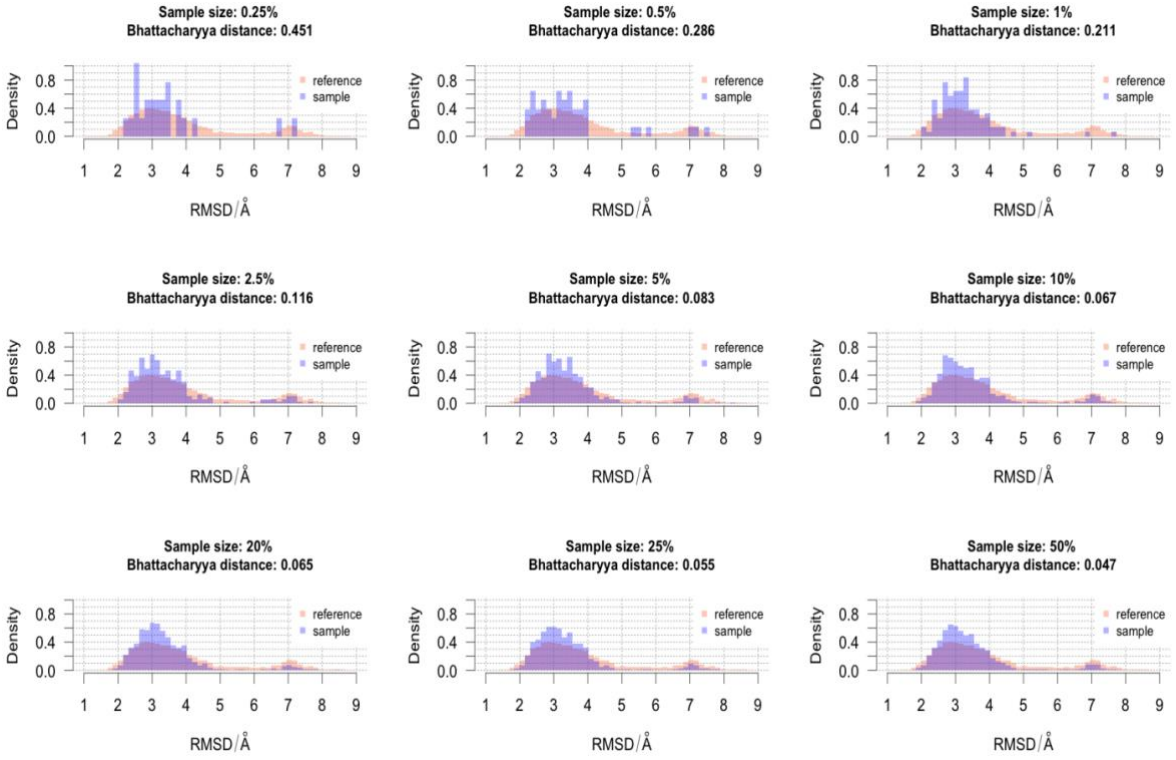


Figure 5.4 Summary results for "Weighted sampling of pocket openings for ensemble docking" scenario. RMSD values for ADK LID opening were calculated for each frame. The range of values was then discretised in 100 bins, and frequency counts were recorded for each bin and used as a weight for each frame. The resulting set of frames was 10% of the original trajectory and was extracted by weighted random sampling. This set contains random structures selected from the most frequently observed conformations in the original trajectory. As a result, an enrichment of the close conformations was generated compared to unweighted random sampling.

The RMSD values of Cα atoms in the binding pocket LID (residues 120-160) were calculated for each frame relative to a reference structure. These RMSD values were then discretised into 100 bins, where the frequency of each bin represented the number of frames with RMSD values in that range. A weighting vector was generated based on these frequencies, which informed the weighted random sampling process. This method ensures that frames are selected proportionally across all conformational states, regardless of their frequency in the original trajectory.

Using weighted random sampling, 10% of the total frames (10,000 out of 100,000) were selected. This strategy successfully enriched the representation of less frequent conformations—particularly the closed state—while maintaining the dominant conformations, such as the open state. Compared to unweighted random sampling, which tends to favour the most frequent conformations, the weighted approach achieved a more balanced distribution of states, as shown in Figure 5.4.

The RMSD histograms illustrate the following: both open and closed conformations are equally represented in the subsampled trajectory, demonstrating the effectiveness of the weighted strategy in avoiding bias. The effectiveness of the weighted sampling strategy was further quantified with a Bhattacharyya distance value of 0.067 for a 10% sample size, indicating a strong alignment between the weighted subsample and the original RMSD distribution. Comparatively, smaller sample sizes (e.g., 2.5% with 0.116) showed more significant dissimilarity, highlighting the importance of sample size in achieving a balanced representation of both states.

5.1.4 Advanced scenario: machine learning prediction

This scenario demonstrates the ability of MDSS to prepare subsampled trajectory data for ML workflows, enabling the prediction and classification of protein conformational states. The goal is to offer a combined solution to sample and reshape frame coordinates to a suitable format for ML tasks. The workflow begins with subsampled trajectory frames saved as *NumPy* arrays, which are then reshaped from their original 3D format (atoms × Cartesian coordinates × frames) into a 2D tabular format. In this transformation, each row represents a trajectory frame (F1, F2, …, Fn), while the columns correspond to the atomic Cartesian coordinates that serve as features for the ML models (Figure 5.5).

The input data, therefore, consists of atomic coordinates extracted from the trajectory, which capture the structural state of the protein at each frame in the format of single observations over a series of variables (the cartesian coordinates in this case). To enable supervised learning, a target variable was added to represent the conformational state of the protein (Figure 5.6). For the purpose of this scenario, a simple state label was created for each frame. These labels were generated using a combination of geometrical information and expert choices, to avoid a trivial ML task where there is linear dependency between input and output variable.

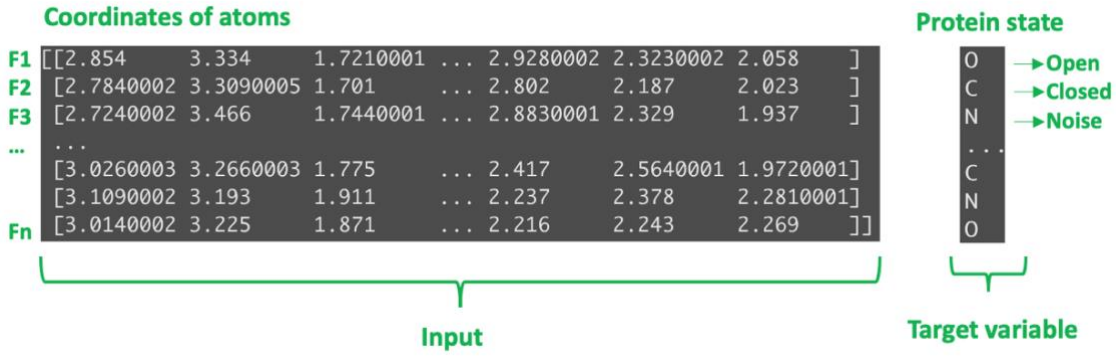


Figure 5.5 Transformation of subsampled trajectory data into a machine learning-compatible format. The input matrix consists of atomic Cartesian coordinates for each trajectory frame (F1, F2, ..., Fn), while the target variable corresponds to the manually labelled protein states: Open (O), Closed (C), and Noise (N).

First a conformation space was created using two informative variables for the process: Rg and distance between two key residues (G55 and P127). Then density analysis was performed on the space to detect high-density region putative to be minima in the conformational space (see Figure 5.6). Frames in the high-density region corresponding to higher Rg and distance values were labelled as the open state (O), frames with lower values were categorised as the closed state (C), and all remaining frames were assigned as noise (N). This manual labelling process effectively translates the complex protein dynamics into three clearly defined states that can be predicted using ML.

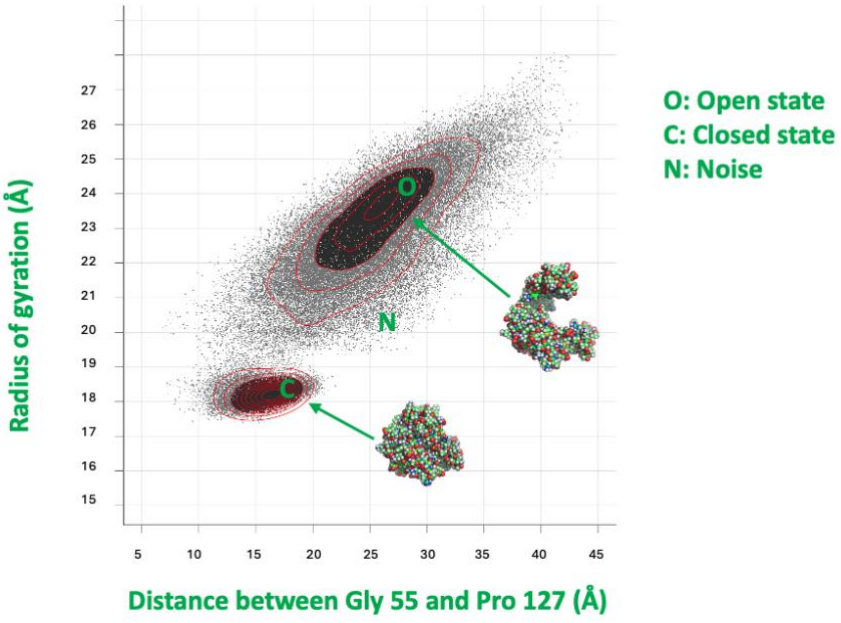


Figure 5.6 Approximate view of the conformational space representation using Radius of Gyration (y-axis) and distance between two key residues (x-axis). The open state (O) corresponds to larger Rg and distance values, forming a distinct cluster in the upper region. The closed state (C) forms a separate cluster in the lower region, with reduced Rg and distance values. Frames scattered outside these clusters are labelled as noise (N).

Given a target label, a model was trained on 70% of the frames and tested on the remaining 30%. The evaluation performance for all three ML methods used was done through assessment of accuracy, Cohen's Kappa scores, and confusion matrices to assess their ability to classify the protein conformational states.

Cohen's Kappa score is a statistical metric used to assess the level of agreement between predicted and actual labels while adjusting for the likelihood of chance agreement. A score of 1 signifies perfect agreement, whereas a score of 0 indicates no agreement beyond what is expected by chance. This metric provides a more robust evaluation than accuracy alone, particularly for imbalanced datasets or multi-class problems such as the classification of protein conformational states.

Logistic Regression (LR):

LG achieved an accuracy of 76.72% and a Cohen's Kappa score of 0.62, providing a reliable baseline for comparison. However, the confusion matrix (Figure 5.7, top) highlights challenges in correctly classifying the Open and Noise states due to their overlapping structural properties. While the model successfully identifies the Closed state, it frequently misclassifies frames between the other two states.

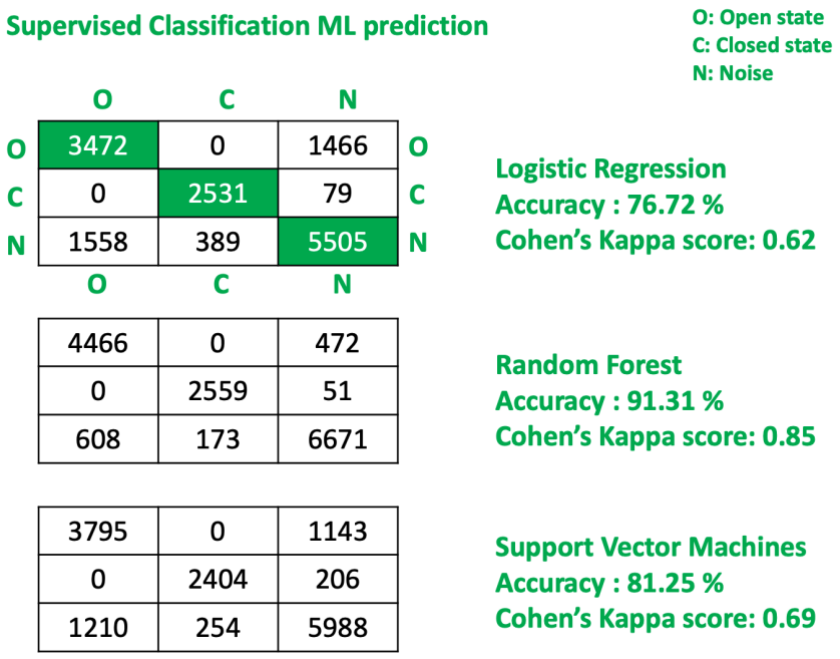


Figure 5.7 Confusion matrices for the three machine learning models—Logistic Regression (top), Random Forest (middle), and Support Vector Machine (bottom)—were used to predict protein conformational states. Each matrix shows the distribution of actual and predicted labels for the Open (O), Closed (C), and Noise (N) states. The best performance is achieved by the Random Forest model, as it is correctly classifying most frames with minimal misclassification.

Random Forest (RF):

The RF model delivered the best performance, achieving an accuracy of 91.31% and a Cohen's Kappa score of 0.85, indicating close agreement with the ground truth labels. The confusion matrix (Figure 5.7, middle) shows excellent classification across all three states, with minimal misclassification of frames between Open, Closed, and Noise states. This result reflects the model's robustness in capturing complex, non-linear relationships within the feature space.

Support Vector Machines (SVM):

The SVM classifier achieved an accuracy of 81.85% and a Cohen's Kappa score of 0.69. The confusion matrix (Figure 5.7, bottom) reveals that the model performs well for the Closed state, accurately distinguishing it from the other two states. However, it struggles to differentiate between the Open and Noise states, leading to misclassifications where structural properties overlap.

5.2 Validation of MD simulations for proof-of-concept

The following section presents the results of the MD simulations conducted on the ADK system, including the WT, single mutants V135G and V142G and the double mutant (DM) V135G_V142G. The results highlight the structural differences induced by the mutations and their subsequent impact on protein dynamics. The primary objective was to assess whether these mutations facilitate the sampling of the closed state. The evaluation was carried out through an examination of geometric properties.

5.2.1 Rationale for generation and validation of MD data for MDAM

The generation and validation of MD simulations were crucial for evaluating the MDAM toolkit. The goal was to provide a validated reference dataset and use it to test MDAM's ability to evaluate the dynamic impact of mutations in ADK automatically. The choice of mutants and the decision to run microsecond simulations builds upon a study by Song et al. [165], which relied on relatively short simulation timescales (100 ns). Simulations on a longer timescale would better verify that specific mutations (V135G, V145G) impact ADK's closure.

While initial simulations were conducted for 300 ns trajectories across 10 replicas, these did not capture the closure of ADK in any replicas. Therefore, only the results from the 1000 ns

trajectories are reported in this section. These extended simulations allowed for a more complete sampling of ADK's open and closed states, with ADK's closure being observed only in the second replica of the DM simulation. This extension enabled comparisons between WT, single mutants, and DM, validating the effect of mutations in ADK's closure.

5.2.2 Data integrity and trajectory validation

The integrity of the generated trajectories was first validated using `gmx check -f` to ensure that the files were not corrupted or that there were no data inconsistencies. Additionally, visual inspection with VMD [175] confirmed dynamical changes without anomalous disruptions. These validations established reliable MD data datasets for further dynamic analyses.

Geometric properties: assessing dynamics

The geometric properties of the system were analysed to validate its dynamic behaviour and assess the closure of ADK. Properties evaluated include RMSD, RMSF, Rg, the distance between key residues (P127 and G55), and the COM Distance. For the time series plots, a single replica per structure (WT, V135G, V142G, DM) was used to illustrate representative behaviour over the simulation time.

Trajectories from all five replicas of 1000 ns simulations across all four structures were concatenated and compared for distribution plots. An exception was made for the COM Distance analysis, where only the second replica of the 1000 ns simulations for WT and DM was used to generate the density plot. This combined approach provided a complete view of the system's stability and dynamic sampling through time series and distribution analysis.

1. **RMSD**: Calculations were performed on the Cα atoms of the protein to measure deviations from the initial structure. As shown in Figure 5.8, the time series analysis revealed that the WT, single mutants and DM systems stabilised after the equilibration phase. While the WT exhibited relatively consistent RMSD values throughout the simulation, the DM demonstrated higher fluctuations at specific points, indicating increased sampling of conformational variability. This behaviour suggests that the DM has a greater propensity to explore distinct conformational states, which is consistent with its role in promoting domain closure in ADK.

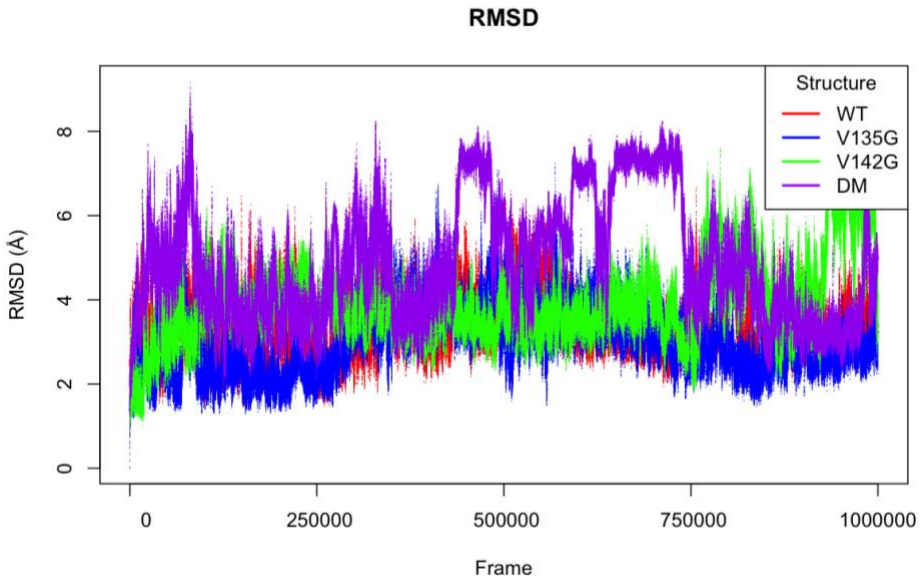


Figure 5.8 Time series RMSD analysis of ADK Cα atoms. The plot shows the smooth-running average of RMSD throughout the simulation for the WT (red), single mutants V135G (blue) and V142G (green), and the double mutant (DM) V135G/V142G (purple). While all systems stabilise after equilibration, the DM exhibits higher RMSD values intermittently, reflecting increased conformational flexibility.

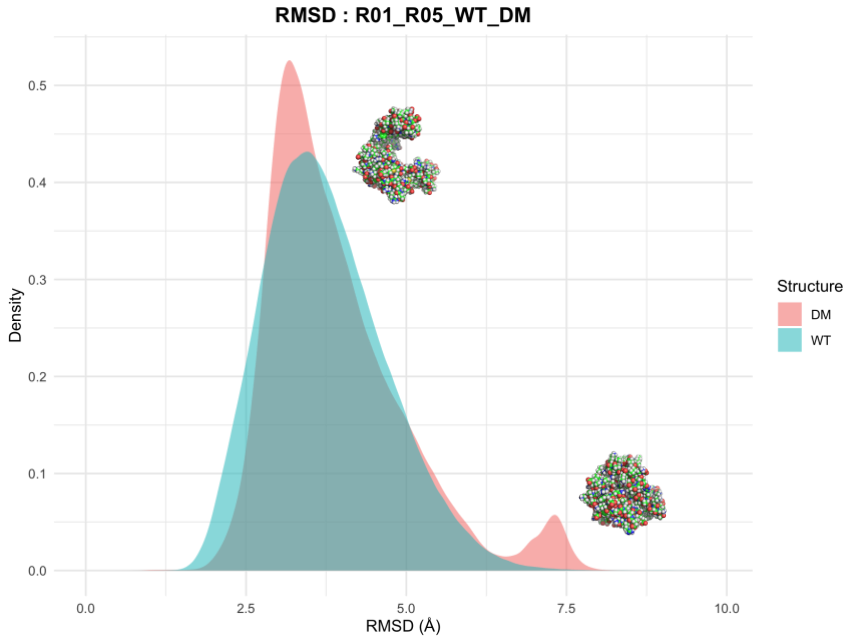


Figure 5.9 Density distribution of RMSD values for WT and DM systems. The plot compares the RMSD distributions for WT (blue) and DM (red). The WT system shows a unimodal distribution centred around 3 Å. In comparison, the DM system displays a bimodal distribution with peaks at 3 Å and 7 Å, indicating sampling of both open and closed states.

The density plot in Figure 5.9 highlights the distinct dynamic behaviour of the WT and DM systems. The WT trajectory exhibits a narrow unimodal distribution, centred around ~3.00 Å, indicating a stable open-state conformation throughout the simulation.

In contrast, the DM shows a bimodal distribution, with peaks near ~3.00 Å and ~7.00 Å. This bimodal behaviour reflects the DM's ability to sample both open and closed states, suggesting that the combined V135G and V142G mutations (i.e. DM) facilitate transitions to the closed-state conformation. These results provide evidence that the mutations promote the closure of ADK, which is consistent with the study's objectives [165].

2. **RMSF**: Calculations were performed on the Cα atoms of the protein to evaluate the flexibility of individual residues within the ADK structure. As shown in Figure 5.10, the RMSF analysis highlights regions of increased flexibility, particularly in the LID (residues 120–160) and AMPbd (residues 30–60) domains, which are functionally important for ADK closure. WT and DM display similar overall flexibility patterns, with peaks in the same regions. However, the DM shows slightly higher fluctuations in the LID domain, consistent with its enhanced capacity to transition into the closed conformation. This increased flexibility in the LID domain is likely a key factor driving the observed differences in the dynamic behaviour between the two systems. The single mutants, V135G and V142G, exhibit intermediate flexibility, with V142G showing slightly more significant fluctuations than V135G.

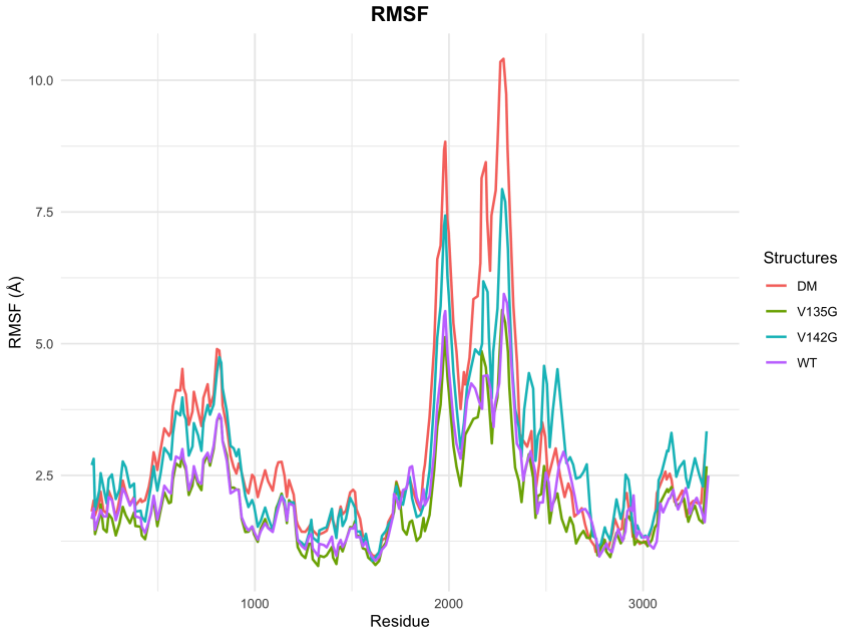


Figure 5.10 RMSF analysis of Cα atoms in ADK. The plot displays residue-wise RMSF for WT (purple), single mutants V135G (green) and V142G (blue), and double mutant (DM) V135G/V142G (red). Peaks in the LID (residues 120–160) and AMPbd (residues 30–60) domains indicate higher flexibility, with the single mutants showing moderate increases in LID fluctuations. At the same time, the DM exhibits the most pronounced flexibility, facilitating conformational transitions.

3. **Radius of Gyration (Rg)**: Calculations were done to evaluate the compactness of the ADK structure throughout the simulations. The time series data in Figure 5.11 confirm that all four systems maintain consistent Rg values, with no significant collapse or expansion observed during the trajectories. This stability in Rg suggests that the overall structural integrity of the protein is preserved across all variants.

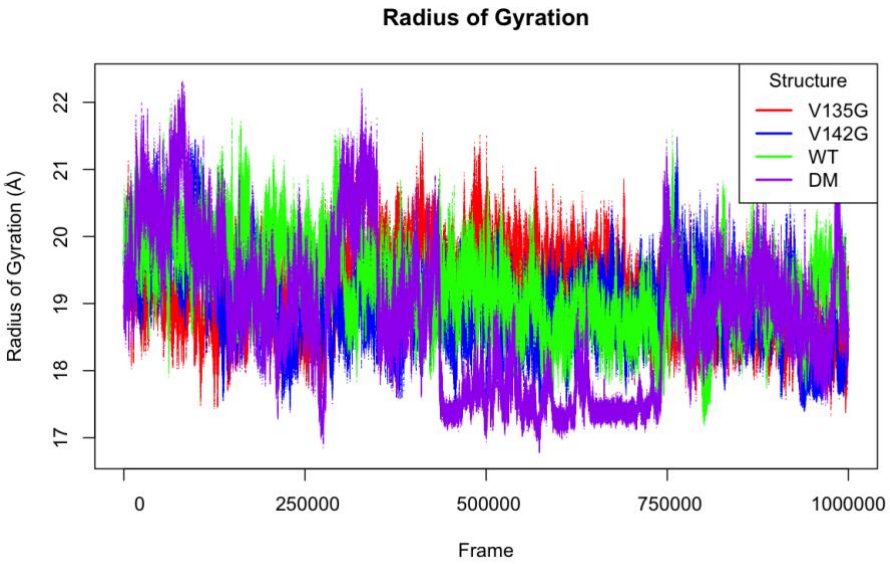


Figure 5.11 Time series Rg analysis for ADK. The plot shows Rg values over time for WT (green), single mutants V135G (red) and V142G (blue), and the double mutant (DM) V135G/V142G (purple). Stable values across all systems indicate maintained structural compactness during the simulations.

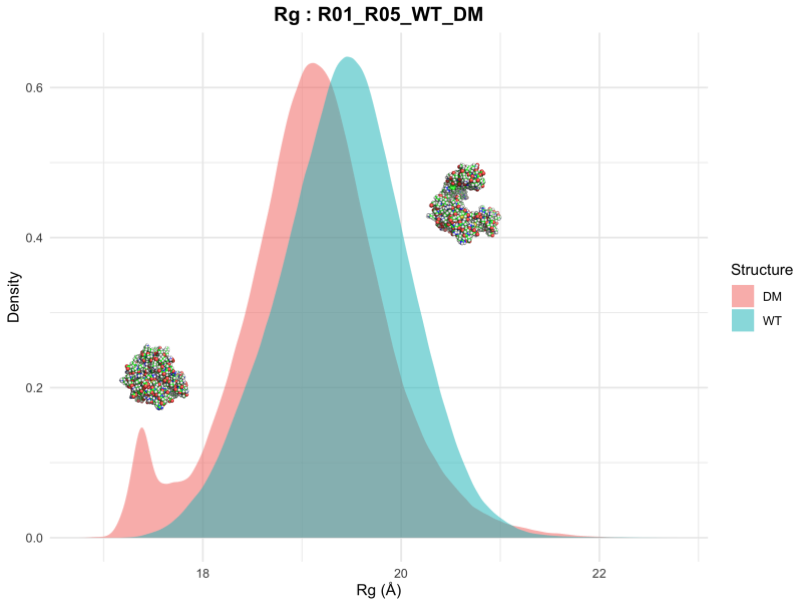


Figure 5.12 Rg density distribution for ADK. The plot compares Rg distributions for WT (blue) and DM (red). The WT exhibits a unimodal distribution centred near 19.5 Å, while the DM shows a broader, bimodal distribution, indicating increased conformational diversity.

The WT exhibits a unimodal Rg distribution centred near ~19.5 Å, indicative of a stable and predominantly compact state throughout the simulation (Figure 5.12). In contrast, the DM shows a broader, bimodal distribution with peaks near ~19 Å and a secondary peak around ~17 Å. Therefore, the DM's ability to sample a more diverse range of conformational states is consistent with occasional structural rearrangements and domain movements that facilitate transitions toward the closed conformation.

4. **Distance between key residues**: The distance between two key residues, P127 (on the LID domain) and G55 (on the AMPbd domain), was calculated to evaluate ADK's closure mechanism. The distribution of these distances, shown in Figure 5.13, highlights key differences between the variants.

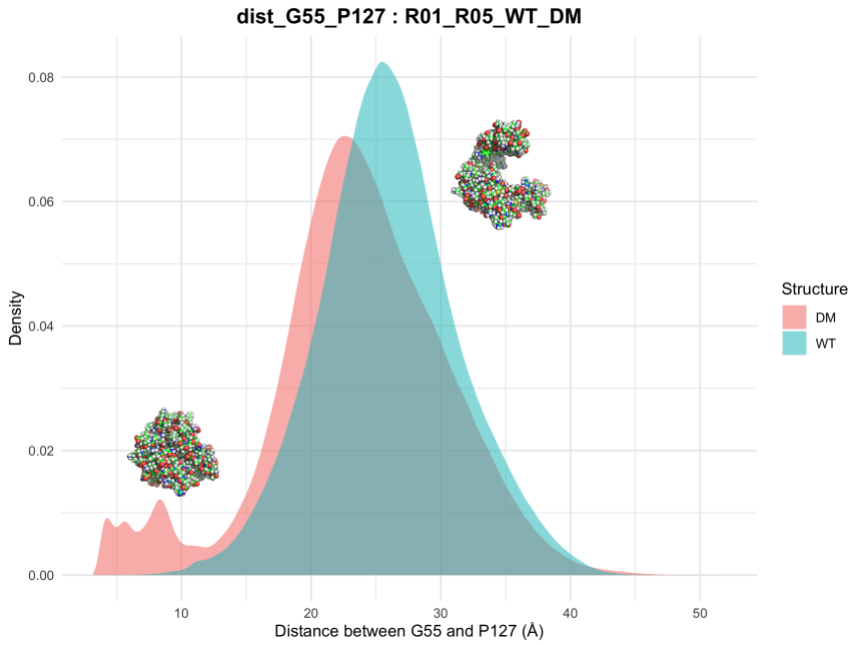


Figure 5.13 Distance distribution between residues P127 and G55 in ADK. The plot compares the distributions of distance between these two residues for WT (blue) and DM (red). The WT shows a unimodal distribution at ~27 Å, while the DM displays a bimodal distribution with peaks at ~25 Å (open state) and ~5 Å (closed state).

The WT exhibits a unimodal distribution centred near ~25 Å, representing a predominantly open conformation. In contrast, the DM shows a bimodal distribution, with one peak near ~23 Å corresponding to the open state and a second peak near ~8 Å corresponding to the closed state.

5. **COM Distance**: The closure of ADK was assessed by calculating the COM Distance between the LID (residues 120–160) and AMPbd (residues 30–60) domains. This

property is a robust indicator of the conformational state of ADK, as a reduction in COM Distance correlates directly with the transition toward the closed state. As shown in Figure 5.14, the WT maintains a consistent COM Distance, indicative of a predominantly open state. In contrast, the DM displays a bimodal distribution with a with two peaks, reflecting its ability to sample closed conformations.

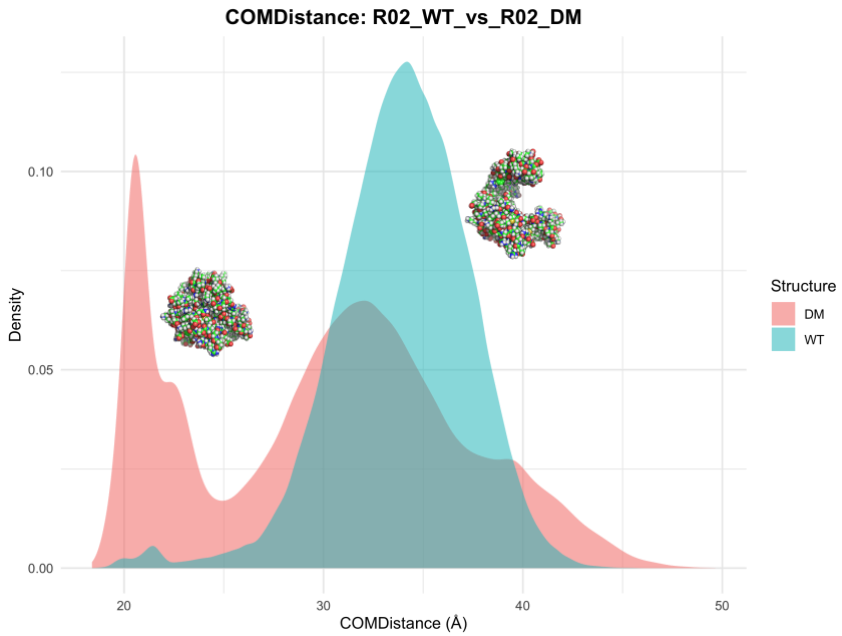


Figure 5.14 COM Distance distribution for ADK between the WT and the DM. The plot compares the COM distance between the LID and AMPbd domains for WT (blue) and DM (red). The WT exhibits a distribution centred near ~35 Å, consistent with an open state. In comparison, the DM shows a bimodal distribution with two peaks near ~33 Å and ~22 Å, indicative of both open and closed states respectively. This plot was derived using only the second replica of the 1000 ns simulations for WT and DM.

Energy and temperature stability

The stability of the simulations was assessed by analysing potential energy and system temperature, using `gmx eneconv` and `gmx energy` to combine and extract data from energy files. Across all structures and replicas, the system temperature remained steady at the target level (~300 K), confirming the proper functioning of the thermostat. Similarly, total energy exhibited no significant drifts or trends in any simulation, demonstrating energetic stability throughout the trajectories.

Figure 5.15 illustrates the energy and temperature profiles for the second replica of the DM structure. The total energy remains constant, with no evidence of instability, while the

temperature profile shows consistent values around ~300 K, confirming the reliability of the simulation conditions.

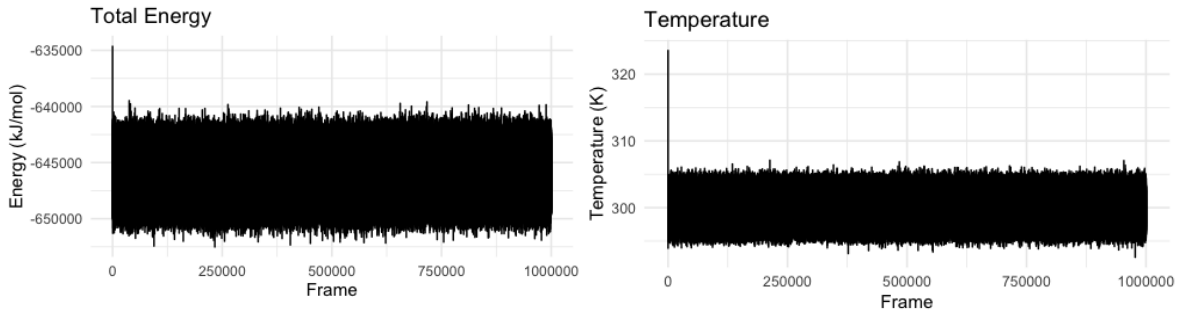


Figure 5.15 Example energy and temperature profiles for the DM Structure (Second Replica). Left: Total energy, confirming the energetic stability of the system. Right: The system's temperature profile demonstrates stable temperature around the desired ~300 K. These results represent observations across all structures and replicas.

Advanced analysis: Principal Component Analysis (PCA)

PCA was performed to analyse the protein's collective motions and distinguish differences in the conformational sampling between the WT and the DM of ADK. PCA calculations were carried out on concatenated trajectories of the Cα atoms, enabling the identification of key motions captured by the first two principal components (PC1 and PC2). PC1 and PC2 explained approximately 63% of the total variance, with PC1 describing the primary motion of domain closure and PC2 capturing the LID's twisting movement during closure. The projections of individual replicas onto the PC1-PC2 space are shown in Figure 5.17.

The density plots reveal distinct conformational sampling behaviours for WT and DM. The WT primarily occupies a high-density region near the origin and exhibits limited movement toward the closed state within the simulation timescale. In contrast, the DM displays broader sampling, including direct transitions to the closed state, emphasising the mutations' role in facilitating ADK's closure. These findings align with the results by Song et al. [165] even when extending them to longer timescales.

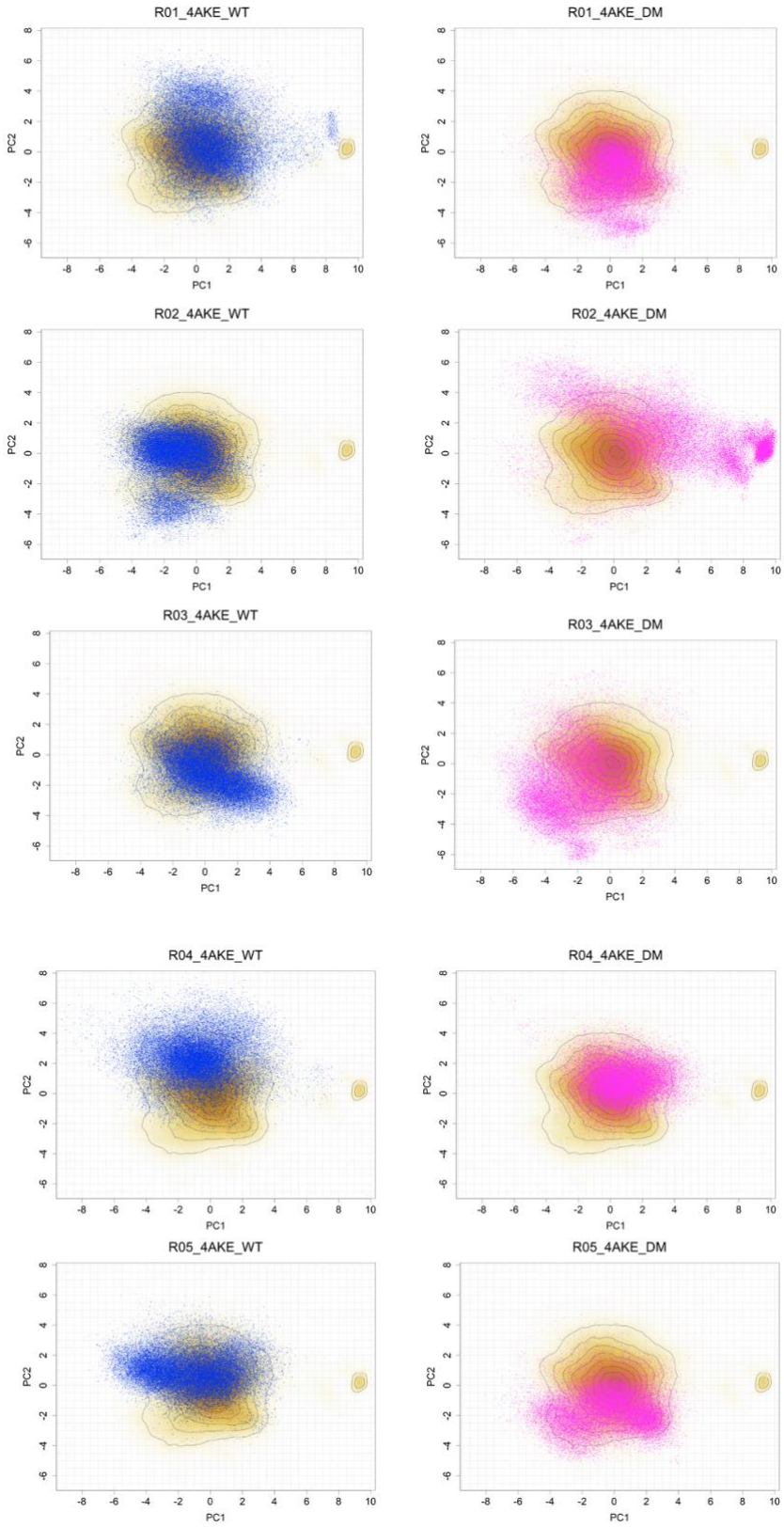


Figure 5.16 PCA Projections for WT and DM. Density plots of the PC1-PC2 space show WT (blue) primarily sampling open and intermediate conformations, while DM (magenta) samples closed states. Data is based on five concatenated replicas for each structure.

Porcupine plots were generated to interpret further the motions captured by PC1 and PC2 (Figure 5.16, right). These plots visualise the direction and collective motions associated with each PC:

- PC1: Captures the closing motion of the LID and AMPbd domains toward the catalytic core of ADK, consistent with functional closure.
- PC2: Represents a twisting of the LID domain that happens during the closing motion.

Figure 5.16 (left) provides an approximate view of the conformational space sampled by the WT and DM systems, aggregated across all five replicas. The density contours in the PC1-PC2 space reveal key differences in sampling behaviour. The WT primarily occupies a high-density region near the origin, representing the open and intermediate conformations. In contrast, the DM shows a broader distribution, extending into regions associated with the closed state. The secondary cluster observed for the DM reflects its ability to directly sample the closed conformation, highlighting the impact of mutations in facilitating ADK closure. This plot offers a qualitative perspective on how the mutations influence the conformational landscape.

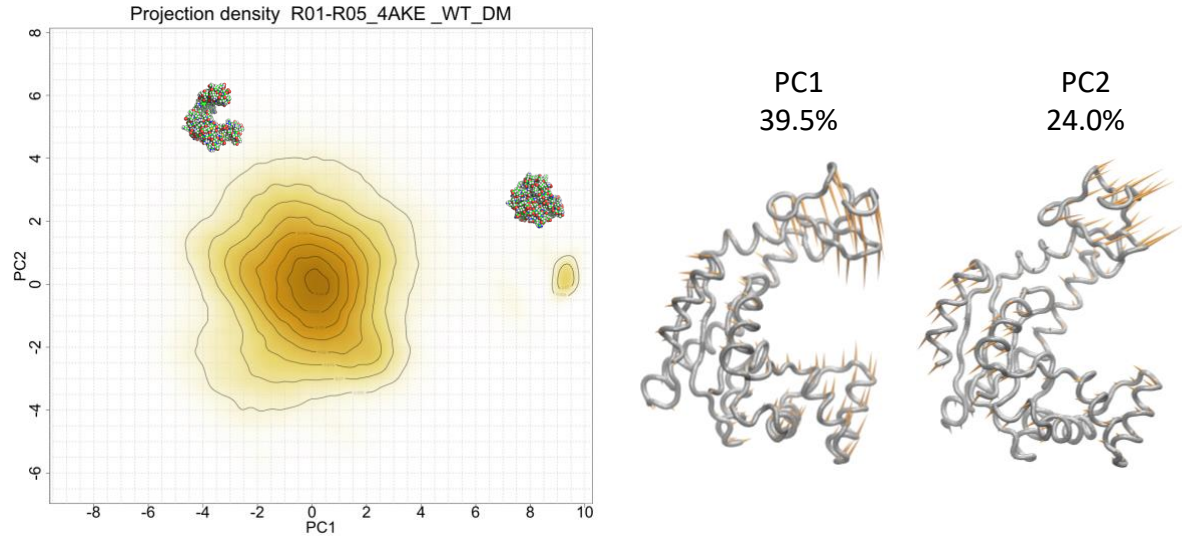


Figure 5.17 Right: Porcupine plots of PC1 and PC2. The plots illustrate the direction (arrows) and magnitude (length of arrows) of motions associated with PC1 and PC2. PC1 captures domain closure, while PC2 depicts lateral shifts of the LID domain. Left: Combined PCA Density for WT and DM Systems. The plot shows an approximate view of the conformational space sampled by WT and DM in the PC1-PC2 projection. WT primarily samples the open state, while DM exhibits broader sampling, including transitions to the closed states.

5.3 MDAutoMut results

This section provides the results from validating the MDAM toolkit, focusing on its ability to automatically generate simulations, introduce mutations and evaluate their impact on ADK's protein dynamics. The impact of mutations was assessed by analysing the distributional differences of COM Distance between each mutant and the WT and a target distribution for COM Distance. This analysis used MDSS's property and dissimilarity classes to quantify which mutant approximates the desired dynamic behaviour.

MDAM includes functionality to quickly generate distribution plots for visualising results, which are saved in the output folder for immediate analysis. However, for this thesis, the plots were refined and enhanced using R [176] to ensure a more polished presentation.

The proof-of-concept demonstrated the toolkit's capability to identify mutations at specific positions (135,142) that could influence ADK's dynamics to approximate a target conformation (closed conformation). Both single and double-mutation workflows were explored. Single mutations systematically evaluated all 20 possible amino acid substitutions at positions 135 and 142 separately. For double mutations, the combinational complexity of testing all 400 possible pairs of amino acid substitutions was reduced by grouping amino acids into five categories (e.g., minor non-polar, large non-polar, polar uncharged, positively charged, negatively charged). Representative amino acids from each group were selected for systematic testing, resulting in 25 combinations. Additionally, a heuristic approach was employed to refine the search space further and focus on the most promising double mutants.

Short simulations of 50 and 200 ps were initially run to test the proof-of-concept. These short simulations ensured that the tool could automate the mutation workflow and generate valid outputs. Following the tool's deployment on ARCHER2, the goal was to extend these simulations to longer timescales, such as 500 ns or 1000 ns, for more robust validation of the mutations' impact on ADK's closure.

5.3.1 Mutation workflow

The mutation workflow (Figure 3.4) (`mutation_workflow.py`) within MDAM was implemented using *PyRosetta*. This automated process introduces one, two, or more mutations through a defined list of target positions and substitutions. It simplifies mutation

generation and integrates seamlessly into the pipeline, ensuring consistency and reproducibility across multiple systems.

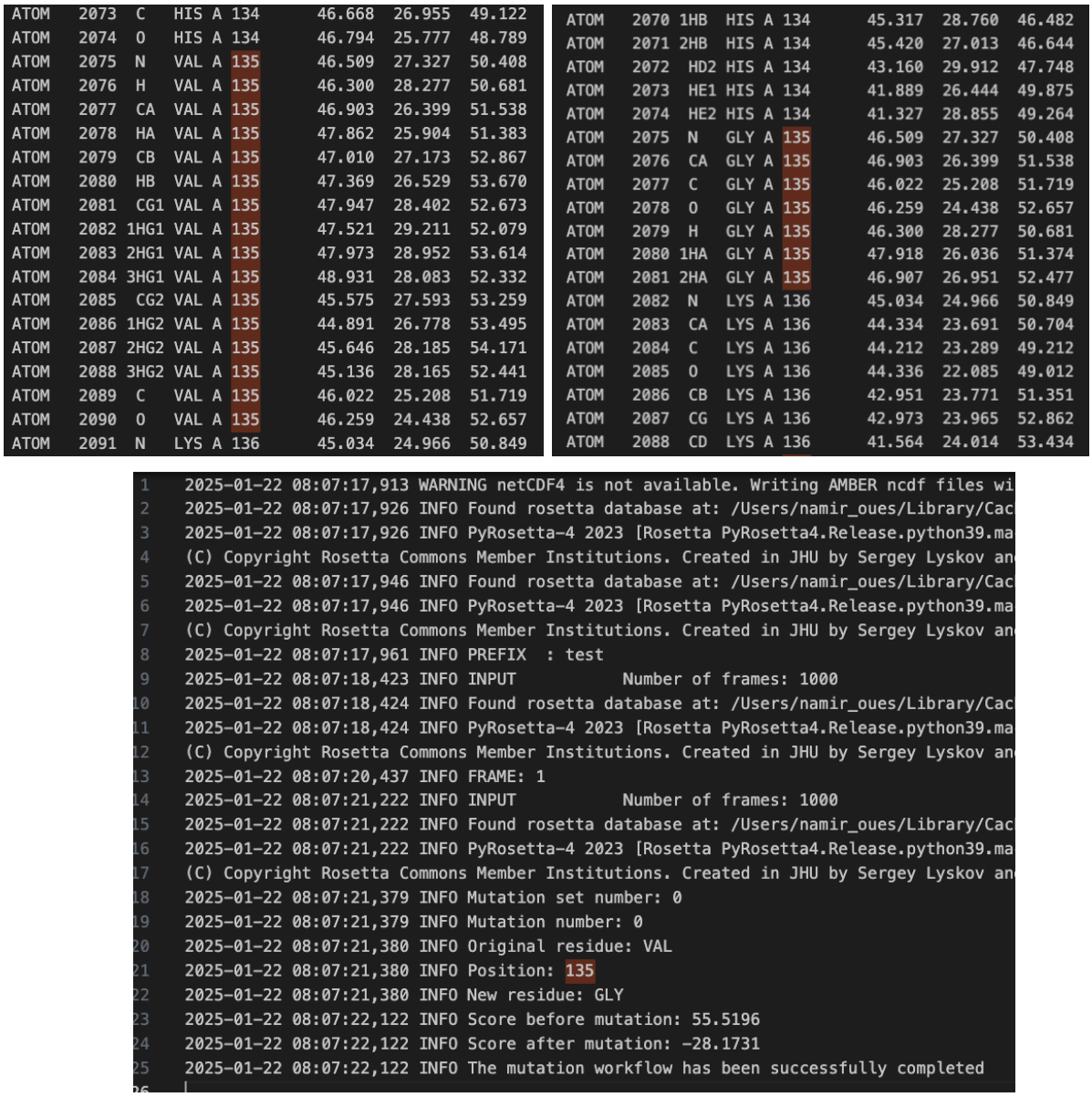


Figure 5.18 Information and log file for the mutation workflow of MDAM. The top panel shows the WT for ADK structure (left) and the generated V135G mutant structure (right). The bottom panel displays the logging output, detailing the mutation at position 135 and the associated Rosetta energy calculations before and after the mutation, emphasising its impact on structural stability.

The workflow modifies the structure for single mutations by substituting a specified amino acid at the desired position in the PDB file. The changes are logged throughout the workflow, providing a detailed record of the mutation, including the specific position, the substituted amino acid, and any energy changes calculated by *Rosetta*. Figure 5.18 (top) shows this

example, where the single mutation V135G was introduced, and the modified PDB structure was generated.

The system's *Rosetta* energy was calculated before and after the mutation engineering to assess the impact of each mutation. These energy values indicate the system's stability following the mutation. The results demonstrate the workflow's ability to modify structures while maintaining structural integrity efficiently. Figure 5.18 (bottom) shows the workflow's logging output sample, including the mutation details and corresponding energy calculations.

5.3.2 System preparation and simulation workflow

The system preparation and simulation workflow (Figure 3.4) (`mdprep_workflow.py`) within MDAM utilises the gmxapi python interface from GROMACS software, automating the preparation and production of MD simulations. This workflow follows a sequence of 10 steps, ensuring the consistent preparation of the system with gradual thermalisation and equilibration before the production phase (see section 3.2.3). For this test case, a predefined recipe, detailed in the Methods chapter, was used to prepare and simulate the system.

```
(C) Copyright Rosetta Commons Member Institutions. Created in JHU by Sergey Lyskov and PyRosetta Team.
2025-01-22 10:13:28,994 INFO                Importing gmxapi.
2025-01-22 10:13:29,000 INFO                Importing gmxapi.operation
2025-01-22 10:13:29,006 INFO                Importing gmxapi.commandline
2025-01-22 10:13:29,010 INFO                Importing gmxapi.runtime
2025-01-22 10:13:29,012 INFO                Importing gmx.workflow
2025-01-22 10:13:29,013 INFO                Using schema version gmxapi_workspec_0_1.
2025-01-22 10:13:29,013 INFO                Importing gmxapi.simulation.mdrun
2025-01-22 10:13:29,014 INFO                Importing gmxapi.simulation.read_tpr
2025-01-22 10:13:29,016 INFO                Importing gmxapi.simulation.modify_input
2025-01-22 10:13:29,021 INFO STEPS          The mdprep workflow has started.
2025-01-22 10:13:29,021 INFO INPUT          Output path directory: /MDAutoMut/mdautomut_workflow_results
2025-01-22 10:13:29,021 INFO INPUT          Prefix : test
2025-01-22 10:13:29,021 INFO STEPS          Creating prefix directory: /MDAutoMut/mdautomut_workflow_results/test
2025-01-22 10:13:29,021 INFO INPUT          WT subfolder path: /MDAutoMut/mdautomut_workflow_results/test/WT
2025-01-22 10:13:29,022 INFO STEPS          Creating WT subfolder path: /MDAutoMut/mdautomut_workflow_results/test/WT
2025-01-22 10:13:29,022 INFO STEPS          Checking for WT files or preparing them if needed.
2025-01-22 10:13:29,022 INFO STEPS          Files for WT not found. System preparation and simulations for WT will start.
2025-01-22 10:13:29,022 INFO INFO           Preparation of the system has started
2025-01-22 10:13:29,868 INFO STEPS          Step 1: pdb2gmx ran successfully!
2025-01-22 10:13:29,920 INFO STEPS          Step 2: editconf ran successfully!
2025-01-22 10:13:30,346 INFO STEPS          Step 3: solvate ran successfully!
2025-01-22 10:13:30,615 INFO STEPS          Step 4: grompp ran successfully!
2025-01-22 10:13:30,785 INFO STEPS          Step 4: genion ran successfully!
2025-01-22 10:13:30,984 INFO OUTPUT         Index file created successfully
2025-01-22 10:13:30,992 INFO OUTPUT         Protein H index file created successfully
2025-01-22 10:13:31,109 INFO STEPS          Step 6: genrestr for posre2000 ran successfully!
2025-01-22 10:13:31,221 INFO STEPS          Step 6: genrestr for posre1000 ran successfully!
2025-01-22 10:13:31,329 INFO STEPS          Step 6: genrestr for posre500 ran successfully!
2025-01-22 10:13:31,443 INFO STEPS          Step 6: genrestr for posre250 ran successfully!
2025-01-22 10:13:31,842 INFO STEPS          Step 7.1: grompp_em_1 ran successfully!
```

Figure 5.19 Log file for mdprep workflow in MDAM. Details all the sequential steps, including energy minimisation and equilibration is provided. The file represents only a sample of the full log.

Figure 5.19 provides a sample logging file overviewing the workflow's sequential progression. Additionally, the workflow's execution generated a structured directory and subdirectory system to organise all input, intermediate, and output files. Figure 5.20 provides a structured view of the directories.

This pipeline not only automates the traditionally manual and error-prone steps of system preparation but also ensures compatibility with subsequent analyses by following a standardised recipe that can be adjusted. Additionally, the workflow was designed with a clear structure and detailed logging, making it easy to debug and troubleshoot errors if they arise during the preparation or simulation process.

```
test/
├── V135G/
│   ├── test_V135G_pdb2gmx.gro                # STEP_1_pdb2gmx
│   ├── test_V135G_topology.top               # STEP_1_pdb2gmx
│   ├── test_V135G_editconf.gro               # STEP_2_editconf
│   ├── test_V135G_solvate.gro                # STEP_3_solvate
│   ├── test_V135G_genion.*                   # STEP_4_add_ions (Represents .gro, .tpr)
│   ├── test_V135G_index.ndx                  # STEP_5_index_file
│   ├── test_V135G_protein_h_index.ndx        # STEP_5_index_file
│   ├── test_V135G_posre.itp                  # STEP_6_position_restraints
│   ├── test_V135G_posre250.itp               # STEP_6_position_restraints
│   ├── test_V135G_posre500.itp               # STEP_6_position_restraints
│   ├── test_V135G_posre1000.itp              # STEP_6_position_restraints
│   ├── test_V135G_posre2000.itp              # STEP_6_position_restraints
│   ├── test_V135G_em1.*                      # STEP_7.1_energy_minimisation_1 (Represents .gro, .log, .tpr)
│   ├── test_V135G_em2.*                      # STEP_7.2_energy_minimisation_2 (Represents .gro, .log, .tpr)
│   ├── test_V135G_em3.*                      # STEP_7.3_energy_minimisation_3 (Represents .gro, .log, .tpr)
│   ├── test_V135G_nvt1.*                     # STEP_8.1_Equilibration_Temp_1 (Represents .gro, .log, .tpr)
│   ├── test_V135G_nvt2.*                     # STEP_8.2_Equilibration_Temp_2 (Represents .gro, .log, .tpr)
│   ├── test_V135G_nvt3.*                     # STEP_8.3_Equilibration_Temp_3 (Represents .gro, .log, .tpr)
│   ├── test_V135G_nvt4.*                     # STEP_8.4_Equilibration_Temp_4 (Represents .gro, .log, .tpr)
│   ├── test_V135G_nvt5.*                     # STEP_8.5_Equilibration_Temp_5 (Represents .gro, .log, .tpr)
│   ├── test_V135G_nvt6.*                     # STEP_8.6_Equilibration_Temp_6 (Represents .gro, .log, .tpr)
│   ├── test_V135G_npt1.*                     # STEP_9.1_Equilibration_Press_1 (Represents .gro, .log, .tpr)
│   ├── test_V135G_npt2.*                     # STEP_9.2_Equilibration_Press_2 (Represents .gro, .log, .tpr)
│   ├── test_V135G_prod.*                     # STEP_10_Production (Represents .gro, .xtc, .pdb, .tpr)
├── V142G/
│   ├── ...                                   # Same structure as V135G
```

Figure 5.20 Hierarchy of directories generated by the MDAM mdprep workflow, ensuring systematic management of input, intermediate, and output files for streamlined analysis and debugging.

5.3.3 Full MDAM workflow

This section presents the outcomes of the complete MDAM workflow, including the systematic exploration of single and double mutations and the heuristic approach for double mutants. By integrating mutation engineering, system preparation, and MD simulations, the MDAM workflow provides an efficient automated pipeline for identifying mutations that promote desired conformational changes in ADK.

To demonstrate the complete workflow, an example of a log file is presented in Figure 5.21 for the heuristic approach, capturing all steps involved in the analysis. A hierarchical representation of the output directories and files is also shown in Figure 5.22 (left), and an example of a mutation list used as input is presented on the right. In order to perform the proof-of-concept, the results focus on evaluating the effectiveness of mutations by calculating the COM Distance between the AMPbd and LID domains of ADK. Calculations were done with

the MDSS library. It then measured the distance between the distributions of COM Distance for each mutant with WT and the target distribution (see section 4.2.2 and Figure 4.9).

```
2025-01-15 15:02:24,286 INFO STEPS          The MDAutoMut workflow has started.
2025-01-15 15:02:24,287 INFO INPUT          Output path directory: /mdautomut_workflow_results
2025-01-15 15:02:24,287 INFO INPUT          Prefix : 1301
2025-01-15 15:02:24,287 INFO INPUT          Prefix directory already exists: /mdautomut_workflow_results/1301
2025-01-15 15:02:24,287 INFO INPUT          WT subfolder path: MDAutoMut/mdautomut_workflow_results/1301/WT
2025-01-15 15:02:24,287 INFO INPUT          WT subfolder path already exists: /mdautomut_workflow_results/1301/WT
2025-01-15 15:02:24,287 INFO STEPS          Checking for WT files or preparing them if needed.
2025-01-15 15:02:24,287 INFO INPUT          Files for WT already exist under prefix 1301. Skipping system preparation and simulations for WT.
2025-01-15 15:02:30,907 INFO INPUT          Number of frames: 25001
2025-01-15 15:02:30,909 INFO                Found rosetta database at: /Users/namir_oues/Library/Caches/pypoetry/virtualenvs/mdam-nQ2z-BL3-py3.9/lib/python3.9/site
2025-01-15 15:02:30,910 INFO                PyRosetta-4 2023 [Rosetta PyRosetta4.Release.python39.mac 2024.01+release.00b79147e63be743438188f93a3f069ca75106d6 2023
2025-01-15 15:02:30,923 INFO STEPS          Calculating properties for wild-type (WT).
2025-01-15 15:02:30,924 INFO INPUT          Atom selection: name CA
2025-01-15 15:02:30,924 INFO INPUT          Property name: COMDistance
2025-01-15 15:02:36,026 INFO OUTPUT         COMDistance calculated and saved to /mdautomut_workflow_results/1301/WT/1301_4AKE_WT_COMDistance.xvg
2025-01-15 15:02:36,026 INFO OUTPUT         WT properties calculated successfully.
2025-01-15 15:02:42,460 INFO INPUT          Number of frames: 25001
2025-01-15 15:02:42,463 INFO                Found rosetta database at: /Users/namir_oues/Library/Caches/pypoetry/virtualenvs/mdam-nQ2z-BL3-py3.9/lib/python3.9/site
2025-01-15 15:02:42,463 INFO                PyRosetta-4 2023 [Rosetta PyRosetta4.Release.python39.mac 2024.01+release.00b79147e63be743438188f93a3f069ca75106d6 2023
2025-01-15 15:02:42,475 INFO STEPS          Reading mutations from file: /data/mutations_list.rtf
2025-01-15 15:02:42,477 INFO STEPS          Loading target property (if provided).
2025-01-15 15:02:42,477 INFO OUTPUT         Comparing Mutant vs Target for the selected property.
2025-01-15 15:02:44,808 INFO STEPS          Heuristic evaluation mode.
2025-01-15 15:02:44,808 INFO STEPS          Processing mutation set 1/25
2025-01-15 15:02:50,187 INFO INPUT          Number of frames: 25001
2025-01-15 15:02:50,190 INFO                Found rosetta database at: /Users/namir_oues/Library/Caches/pypoetry/virtualenvs/mdam-nQ2z-BL3-py3.9/lib/python3.9/site
2025-01-15 15:02:50,190 INFO                PyRosetta-4 2023 [Rosetta PyRosetta4.Release.python39.mac 2024.01+release.00b79147e63be743438188f93a3f069ca75106d6 2023
2025-01-15 15:02:52,790 INFO STEPS          Mutated structure already exists: /mdautomut_workflow_results/1301/V135G_V142L/1301_4AKE_V135G_V142L_prod.pdb. Skipping
2025-01-15 15:02:58,524 INFO INPUT          Number of frames: 25001
2025-01-15 15:02:58,526 INFO                Found rosetta database at: /Users/namir_oues/Library/Caches/pypoetry/virtualenvs/mdam-nQ2z-BL3-py3.9/lib/python3.9/site
2025-01-15 15:02:58,526 INFO                PyRosetta-4 2023 [Rosetta PyRosetta4.Release.python39.mac 2024.01+release.00b79147e63be743438188f93a3f069ca75106d6 2023
2025-01-15 15:02:58,540 INFO INPUT          Atom selection: name CA
2025-01-15 15:02:58,540 INFO INPUT          Property name: COMDistance
2025-01-15 15:03:03,089 INFO OUTPUT         COMDistance calculated and saved to /mdautomut_workflow_results/1301/V135G_V142L/1301_4AKE_V135G_V142L_COMDistance.xvg
2025-01-15 15:03:03,089 INFO STEPS          Comparing mutant vs WT for mutation V135G_V142L
2025-01-15 15:03:03,110 INFO OUTPUT         Dissimilarity score: 0.03513
2025-01-15 15:03:03,746 INFO OUTPUT         Plot saved successfully to /mdautomut_workflow_results/1301/V135G_V142L/V135G_V142L_vs_WT_COMDistance_density.png
2025-01-15 15:03:03,747 INFO STEPS          Comparing mutant vs Target for mutation V135G_V142L
2025-01-15 15:03:05,577 INFO OUTPUT         Dissimilarity score: 0.06614
2025-01-15 15:03:18,768 INFO OUTPUT         Plot saved successfully to /mdautomut_workflow_results/1301/V135G_V142L/V135G_V142L_vs_Target_COMDistance_density.png
2025-01-15 15:03:18,819 INFO OUTPUT         Successful mutation pair: V135G_V142L
2025-01-15 15:03:18,819 INFO STEPS          Processing mutation set 2/25
2025-01-15 15:03:24,427 INFO INPUT          Number of frames: 25001
```

Figure 5.21 Log file for the full MDAM workflow for the proof-of-concept analysis. The panel shows an example log file, capturing all steps involved in the double mutation heuristic approach.

Due to the constraints of computational resources, the workflow was tested using an approach based on a “dry run” where simulations are precalculated and the MDMA workflow is executed. The simulations used in testing were derived from 500 ns trajectories (see section, chapter 3), processed using a stride of 20 ps with GROMACS to produce 25,000 frames per trajectory for all tested mutants, including the WT. As discussed in Chapter 4 (see section 4.2.4), to reduce the combinational complexity, only 5 representative amino acids were used for the testing. Therefore, 25 pairs of mutations were generated and evaluated for the double mutation workflow.

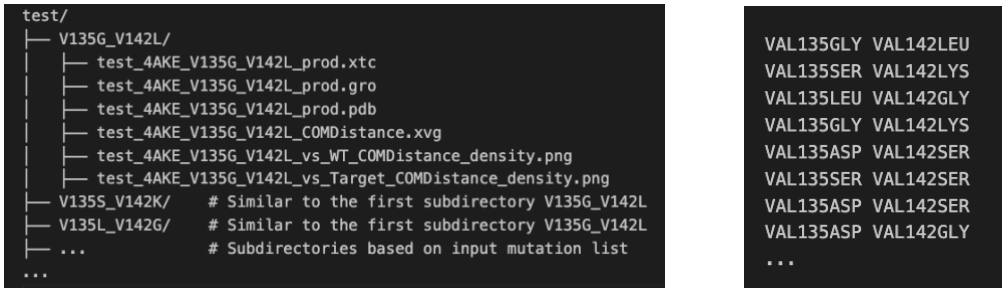


Figure 5.22 Hierarchical organisation of output directories and files generated during the MDAM workflow. The right panel includes a sample mutation list used as input for the analysis.

For this proof-of-concept, the focus was on ADK's backbone conformational changes. In order to accelerate calculations, the analysis was limited to Cα atoms only, as the transition between open and closed states is predominantly a backbone-driven event. The results for all three scenarios—single mutation, double mutation systematic, and double mutation heuristic—provided below are on Cα atoms only.

Single mutation

The single mutation workflow systematically evaluated all 20 possible amino acid substitutions, first at position 135 and then at position 142, to assess their impact on ADK's conformational dynamics. The COM Distance between the AMPbd and LID domains was calculated on Cα atoms. The distribution of values was compared for all mutants against the WT and the target distributions (see section 4.2.2).

Figure 5.23 illustrates the COM Distance density for the V135G mutant. The WT (red) exhibits a unimodal distribution centred around ~35 Å, consistent with a predominantly open-state conformation. The target distribution (blue), representing the desired protein dynamics, reflects both open and closed conformation of ADK, with a bimodal pattern characterised by peaks near ~25 Å and ~35 Å. For V135G (green), a subtle shift is observed toward shorter COM distances, with a broadening towards more open conformations. This result suggests a limited tendency for V135G to sample conformations closer to the closed state. The Bhattacharyya distance of 0.23 between V135G and the target, compared to 0.09 for the WT, indicates that this mutation alone did not achieve the desired target dynamics.

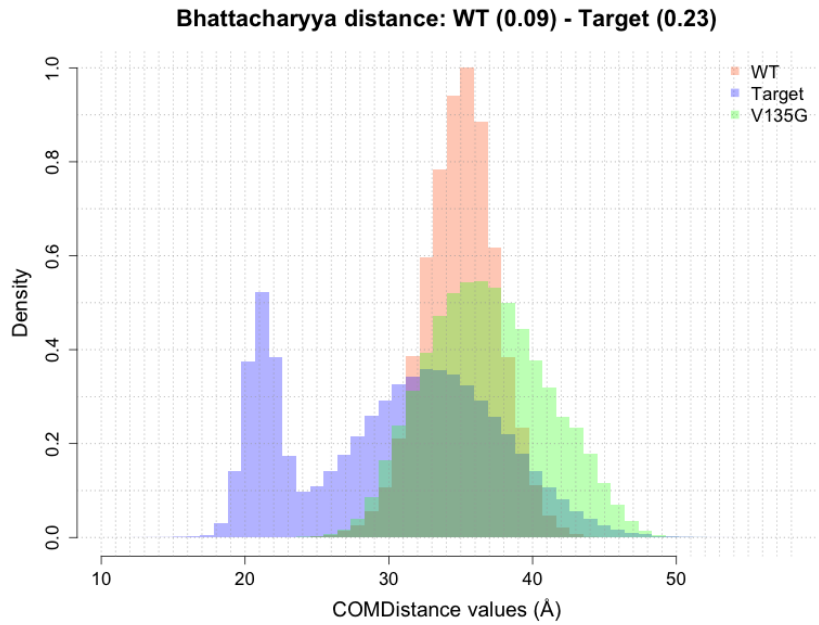


Figure 5.23 COM Distance distribution for the V135G mutant compared to the WT and target distributions. The WT (red) exhibits a unimodal distribution centred at ~35 Å, representing a predominantly open-state conformation. The target distribution (blue) is bimodal, reflecting both open (~ 35 Å) and closed (~ 25 Å) conformations.

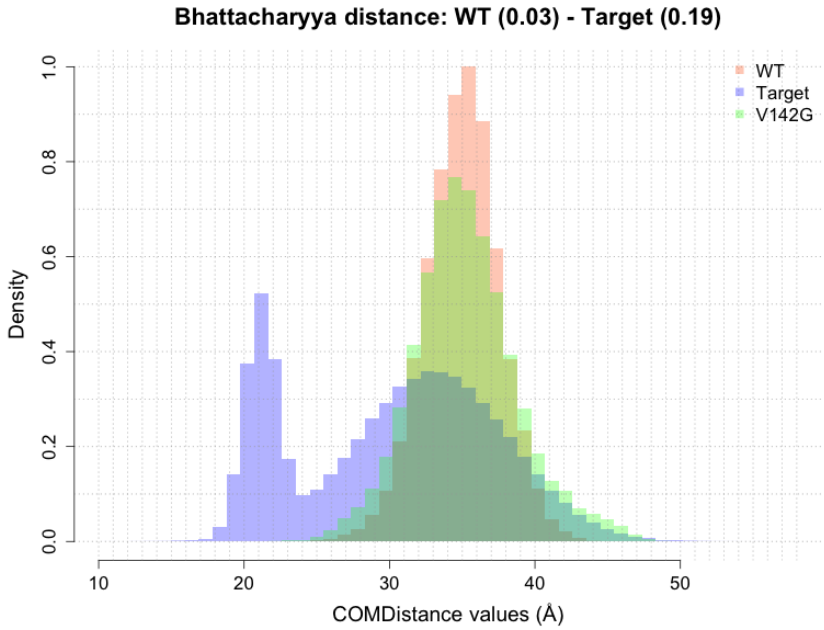


Figure 5.24 COM Distance distribution for the V142G mutant compared to the WT and target distributions. The WT (red) maintains a unimodal distribution centred at ~35 Å, while the target distribution (blue) is bimodal, reflecting both open and closed conformations. The V142G mutant (green) is closer to the WT, and it fails to replicate the bimodal target behaviour fully. The Bhattacharyya distance of 0.19 from the target indicates an incremental improvement over V135G.

Similarly, Figure 5.24 shows the COM Distance density for the V142G mutant. Compared to the V135G, the V142G behaves more like the WT, with a Bhattacharyya distance of 0.19 (i.e. slightly better than V135G). Therefore, the V142G single mutation alone did not achieve the desired closure of ADK.

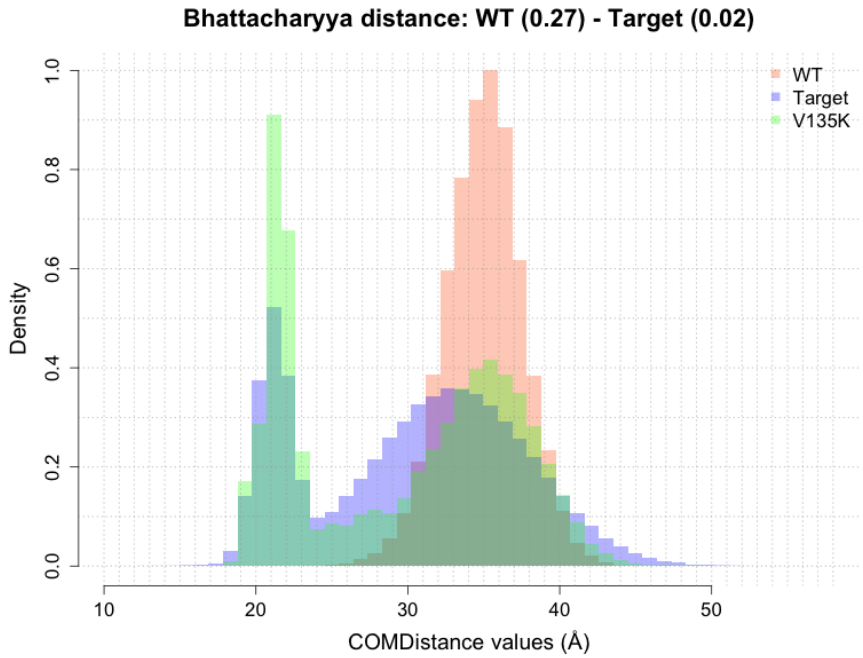


Figure 5.25 COM Distance distribution for the V135K mutant compared to the WT and target distributions. Unlike other single mutants, V135K (green) displays dual peaks at ~35 Å and ~25 Å, indicating its ability to sample both open and closed states. The Bhattacharyya distance of 0.02 to the target (blue) reflects a high degree of overlap. However, the closed-state sampling is more pronounced than in the target, suggesting an imbalance in its dynamics. This result highlights the potential of V135K to modulate ADK dynamics partially, but it does not fully achieve the equilibrium observed in the target distribution.

An interesting result emerged from the single mutation V135K, shown in Figure 5.25. Unlike the other single mutants, V135K demonstrated the ability to sample the closed state directly, with a distinct peak near ~25 Å in its COM Distance distribution. This behaviour aligns much more closely with the target distribution, as indicated by a significantly lower Bhattacharyya distance of 0.02 compared to the target. This result suggests that the substitution of Lysine at position 135 induces a structural shift in ADK dynamics, enabling sampling of the closed state.

However, while V135K demonstrates the ability to sample both states, the closed-state sampling dominates the distribution relative to the target. This imbalance indicates that the mutation alone does not achieve the dynamic equilibrium observed in the target distribution. These findings highlight the potential for single mutations, such as V135K, to modulate ADK dynamics but also emphasise their limitations in achieving the balanced dynamics necessary for full functional closure. However, a more detailed analysis of state transition is required to

have a conclusive view on this, but this is not part of the scope of this work and is more relevant for a broader understanding of ADK's dynamics.

These findings confirm that the single mutations V135G and V142G, as well as other substitutions apart from V135K, are insufficient to induce the conformational shift required for ADK closure. This result aligns with the study by Song et al. [165], which demonstrated that ADK's transition to the closed state is achieved through a combination of mutations, specifically the DM (V135G_V142G), rather than individual amino acid changes. As such, these insights from the single mutation workflow provide a foundation for exploring double mutations and their potential impact on the dynamics to perform a complete framework for this proof-of-concept.

Double mutation systematic approach

The systematic double mutation workflow tested combinations of amino acid substitutions at positions 135 and 142, generating a ranked list of mutations. The COM Distance distributions for each double mutant were compared against the WT and the target distributions to evaluate their ability to approximate the desired closed-state dynamics of ADK. Figure 5.26 presents the ranked list of mutations based on their Bhattacharyya distance to the target distribution, highlighting the top-performing mutants.

mutation_name	Bhattacharyya
V135G_V142G	0.01021
V135L_V142L	0.09543
V135L_V142S	0.12508
V135D_V142D	0.13499
V135S_V142G	0.16749
V135D_V142K	0.17061
V135L_V142D	0.20243
V135D_V142L	0.21583
V135S_V142L	0.21756
V135K_V142L	0.22079
V135G_V142L	0.22467
V135G_V142D	0.22836
V135K_V142S	0.23698

Figure 5.26 Results from the double mutation systematic approach. Ranked list of some of the evaluated double mutants sorted by their Bhattacharyya distance to the target COM Distance distribution, with V135G_V142G (DM) achieving the lowest distance (0.01), indicating the closest match to the target.

After systematically scanning all 25 double mutants, the tool identified V135G_V142G (DM) as the closest to the target, with a Bhattacharyya distance of 0.01. This result is particularly significant as it aligns with Song's et al. [165] study, which demonstrated that this specific DM

promotes ADK closure. The tool's ability to correctly rank V135G_V142G as the top-performing mutation serves as a successful proof-of-concept, validating the effectiveness of the systematic approach in identifying mutations that induce the desired conformational dynamics.

Figure 5.27 illustrates the COM Distance density for V135G_V142G. The WT distribution (red) remains unimodal with a peak at ~35 Å, while the target distribution (blue) is bimodal, reflecting open and closed states. The V135G_V142G mutant (green) achieves near-complete overlap with the target, with peaks at ~25 Å and ~35 Å, capturing both conformational states. This is also demonstrated with the Bhattacharyya distance of 0.01 between the DM and the target. This result underscores the effectiveness of the systematic double mutation workflow in identifying functionally significant mutations.

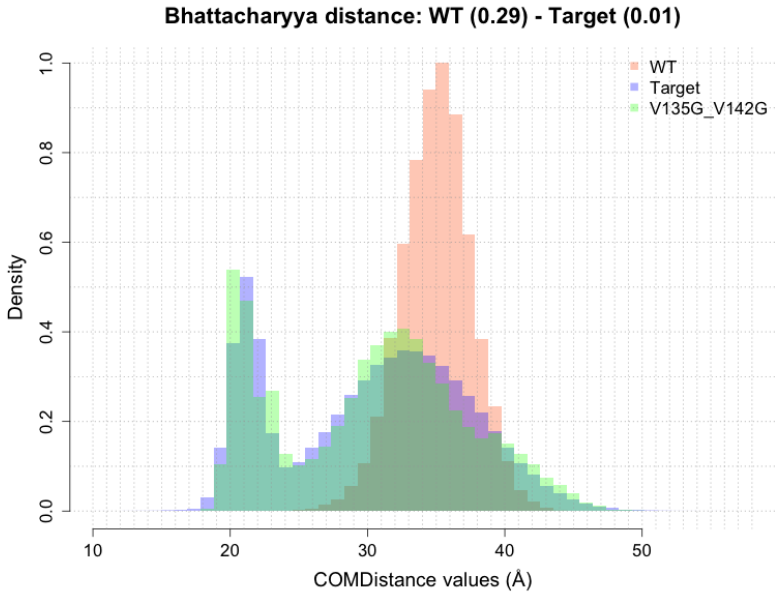


Figure 5.27 Results from the double mutation systematic approach. COM Distance distribution for V135G_V142G compared to the WT (red) and target (blue). The V135G_V142G mutant (green) closely aligns with the target, exhibiting a bimodal distribution with peaks near ~25 Å and ~35 Å.

In addition to V135G_V142G, the second-ranked mutant, V135L_V142L, also demonstrated promising behaviour (Figure 5.28). The Bhattacharyya distance of 0.095 is slightly higher than V135G_V142G, but it still aligned better with the target than other tested mutations. The COM Distance density for V135L_V142L (green) displays a bimodal pattern, suggesting partial sampling of both open and closed states, but there is no significant closure compared to V135G_V142G.

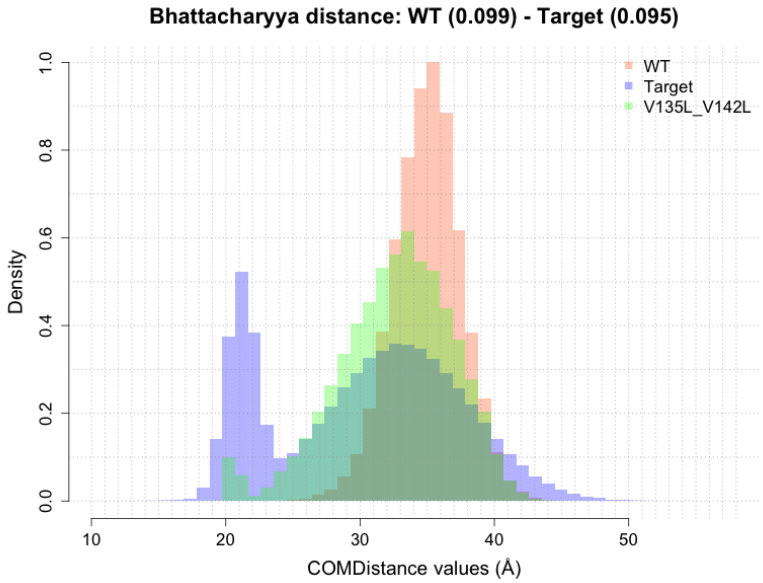


Figure 5.28 COM Distance distribution for V135L_V142L compared to the WT (red) and target (blue). The V135L_V142L mutant (green) approximates the target with a bimodal pattern but emphasizes the open state more than the closed state, as reflected in its Bhattacharyya distance of 0.095.

These results demonstrate the MDAM toolkit's performance in identifying mutations that have the desired impact on ADK's dynamics. While V135G_V142G represents the optimal mutation for promoting closure, other candidates, such as V135L_V142L, demonstrate the potential for alternative mutational strategies.

Double mutation heuristic approach

The heuristic approach employed a targeted search strategy to refine the mutation space by focusing on promising double mutations while excluding combinations unlikely to meet the desired target dynamics.

Iteration	Mutation Pair	Bhattacharyya Distance (Vs Target)	Outcome
1	V135G_V142K	0.27803	Unsuccessful
2	V135S_V142G	0.24816	Unsuccessful
3	V135G_V142G	0.01021	Successful
4	V135K_V142V	0.02779	Successful
5	V135G_V142S	0.24816	Unsuccessful
…	…	…	…

Figure 5.29 Results for the heuristic approach for double mutations. The table is showing the iteration-by-iteration evaluation of mutation pairs, their Bhattacharyya distances to the target, and their outcomes. Successful mutations (e.g., V135K_V142V and V135G_V142G) are highlighted in green, while unsuccessful pairs exceed the dissimilarity threshold (0.05).

This process began by randomly selecting an initial mutation pair from the mutations list. After calculating its Bhattacharyya distance to the target distribution, the search iteratively moved to neighbouring combinations by altering one mutation (at position 135) at a time while keeping the other constant (at position 142). At each step, evaluated combinations were checked against a user-defined dissimilarity threshold of 0.05. Combinations exceeding this threshold were excluded, while successful mutations were retained and added to the output list of "successful mutations" (Figure 5.29).

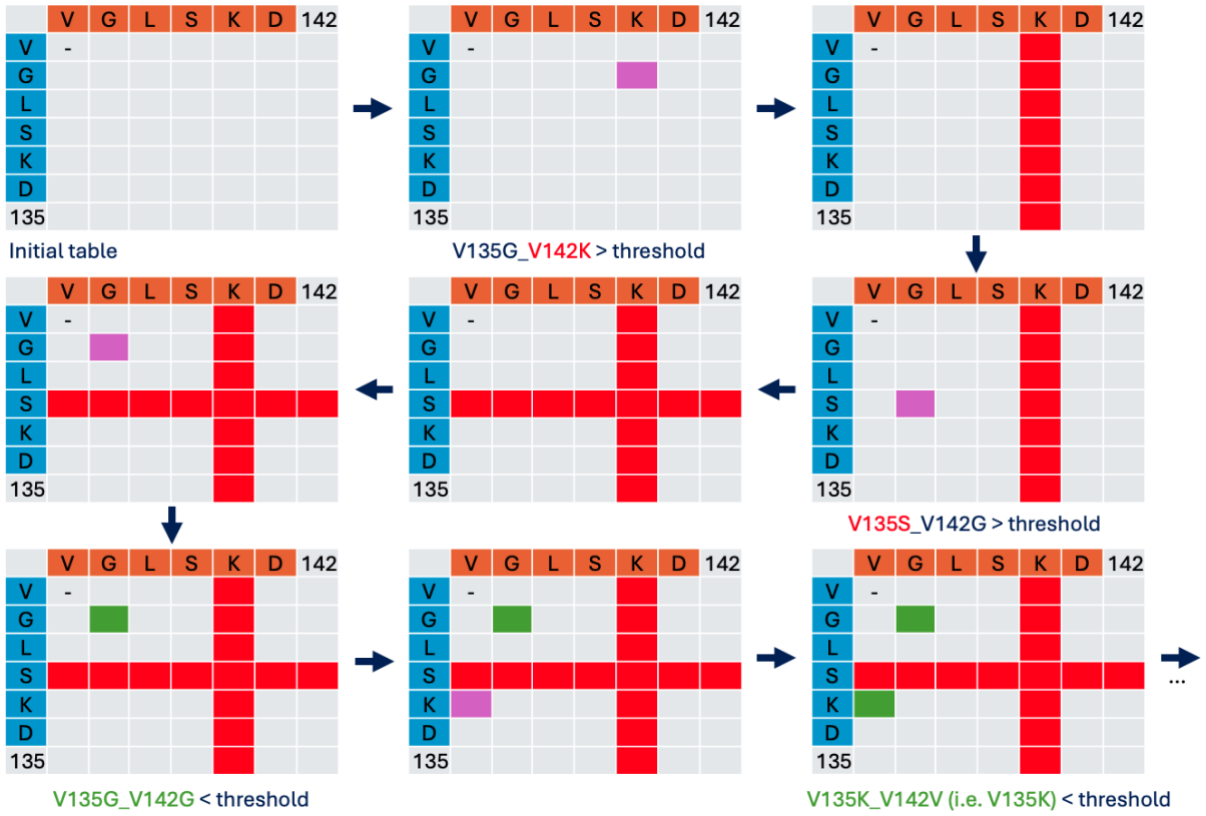


Figure 5.30 Visualisation of the heuristic search process for double mutations. A grid representing the evaluated mutation space, with rows and columns corresponding to mutations at positions 135 and 142, respectively. Evaluated combinations are marked with checkmarks, while excluded combinations (e.g., those with K at position 142) are highlighted in red. This representation illustrates how the heuristic search narrows the mutation space iteratively by focusing on promising candidates.

Figure 5.30 illustrates the search process, with rows and columns representing possible mutations at two positions (rows for position 135 and columns for position 142). The search starts at a random pair (e.g., V135G_V142K) and moves iteratively to neighbouring pairs based on proximity in the mutation space. The method starts with evaluation of the first random pair V135G_V142K. The evaluation process shows that K in position 142 fails to meet the user-defined dissimilarity threshold (0.05), hence all mutations with K at position 142 (shown in red) are excluded. The process goes on until it finds mutations that have the desired impact

on the dynamics (highlighted in green) and saves them in an output list. This exclusion strategy reduced the search space, directing computational resources toward more promising mutations. This iterative process allowed the search to explore the mutation landscape efficiently, narrowing the focus to high-performing mutations while avoiding redundant evaluations

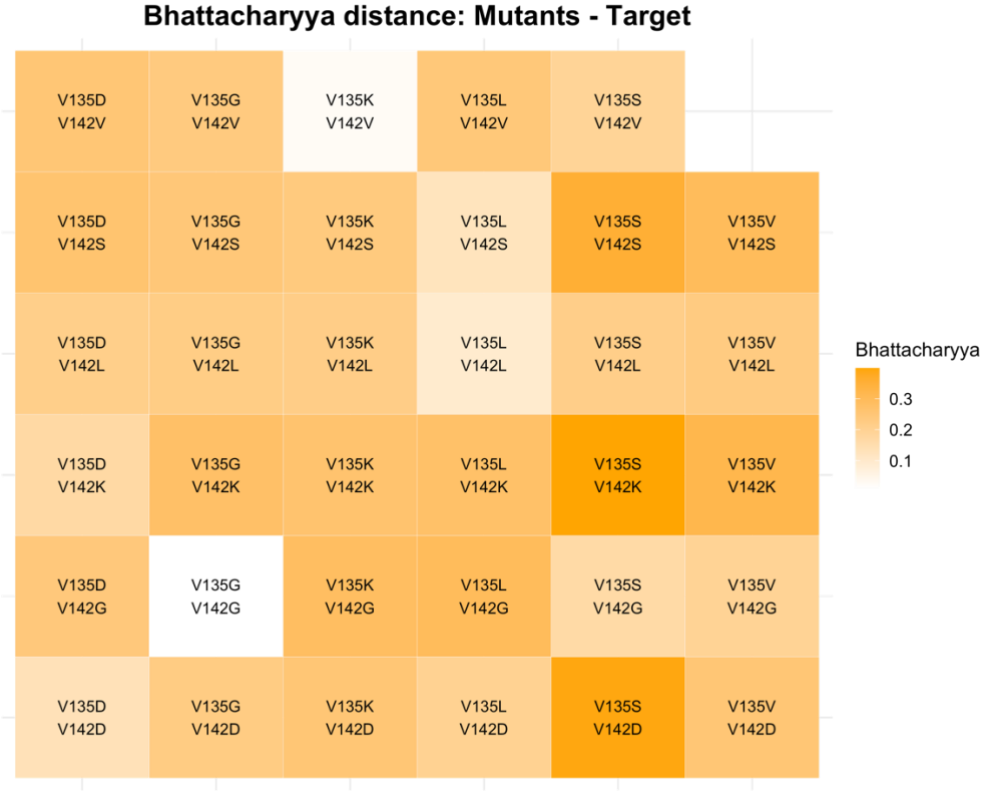


Figure 5.31 A heat map of Bhattacharyya distances for all double mutations evaluated during the heuristic search. Lighter regions represent mutations with distributions closer to the target. The optimal mutation, V135G_V142G, is highlighted with the smallest Bhattacharyya distance (0.01021), and V135K_V142V emerges as another strong candidate with a distance of 0.02. Other promising candidates, such as V135L_V142G, are also visible, illustrating the efficiency of the heuristic approach in identifying high-performing mutations.

Figure 5.31 presents a heat map of Bhattacharyya distances for all double mutants that were evaluated during the heuristic search. Lighter regions in the heat map represent mutations with distances closer to the target, highlighting promising candidates for further exploration. Notably, V135G_V142G was identified as the top-performing mutation, with the smallest Bhattacharyya distance of 0.01, confirming its ability to approximate the target dynamics. In addition, V135K_V142V (single mutant V135K) emerged as another strong candidate with a Bhattacharyya distance of 0.02 (as discussed in the single mutation workflow section). This demonstrated that the heuristic approach can uncover alternative mutations beyond the known optimal result. The heat map also highlights other interesting mutations, such as

V135L_V142G, with distributions that closely align with the target, suggesting additional candidates worth investigating.

While this heuristic search approach provides a simplified framework to validate and test the search process, it also provides flexibility. In this study, the search space for two positions (135 and 142) was defined by 35 possible double mutant combinations, and the primary criterion applied was the Bhattacharyya dissimilarity threshold. However, the heuristic approach can be extended to incorporate more complex criteria, such as additional exclusion rules, prioritised mutation types, or libraries of predefined compatible or incompatible mutation pairs. For example, integrating a plugin that provides prior knowledge of biologically or structurally incompatible mutations would allow the search to exclude non-favourable candidates upfront, further improving efficiency and precision. This adaptability ensures that the heuristic approach can be tailored to diverse research needs while maintaining computational efficiency. Different heuristic strategies can also be easily implemented in MDAM thanks to the modular structure of the code.

5.4 MDAutoPredict results

The MDAP toolkit was validated by performing a supervised classification prediction. The goal was to classify ADK's protein conformational states into four categories: open (A), closed (B), intermediate (I) and non-states (N). The results presented here evaluate MDAP's performance using a dataset derived from the second MD trajectory replica (R02) of 1 μs (1000 ns) generated to validate MDAM. In order to perform the validation process quickly and due to limited computer resources, R02 was subsampled at 40 ps intervals, resulting in 25,0001 frames. The testing process involved six ML models, focusing on accuracy, precision, recall and computational efficiency.

5.4.1 Target variable definition

The labels of the four states were derived through a density-based analysis of the approximate conformational space representation to perform the labelling exercise. This space was constructed by projecting the trajectory data onto the first two principal components (PC1 and PC2), revealing a two-dimensional free energy landscape. The density of points in this space was calculated, with the expectation that high-density regions may be representative of possible free energy minima. These minima were associated with the open (A), closed (B), and intermediate (I) states based on prior literature and visual inspection of structural features.

Low-density regions were classified as non-states (N) for simplicity since the four labels already fit the purpose of the validation process.

The plot in Figure 5.32 shows an approximate view of the conformational space. Each point represents a conformation (i.e. each state)—the structural snapshots of the protein help in understanding the biological interpretation of these associated conformations.

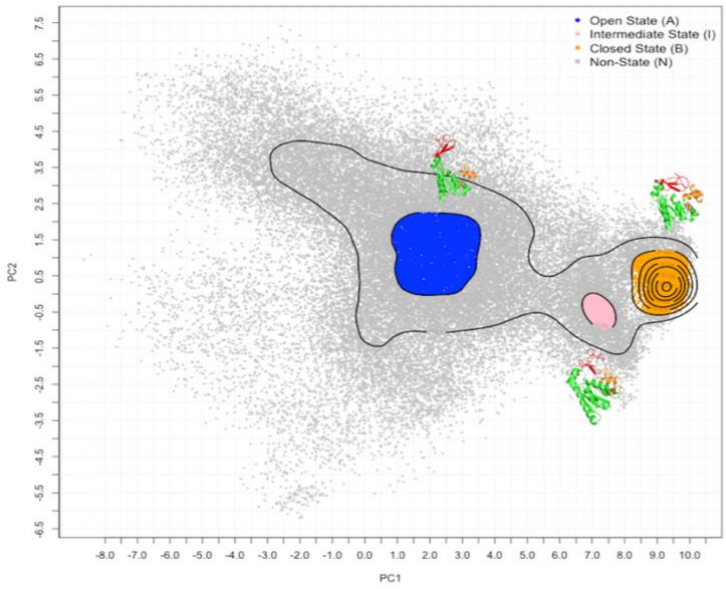
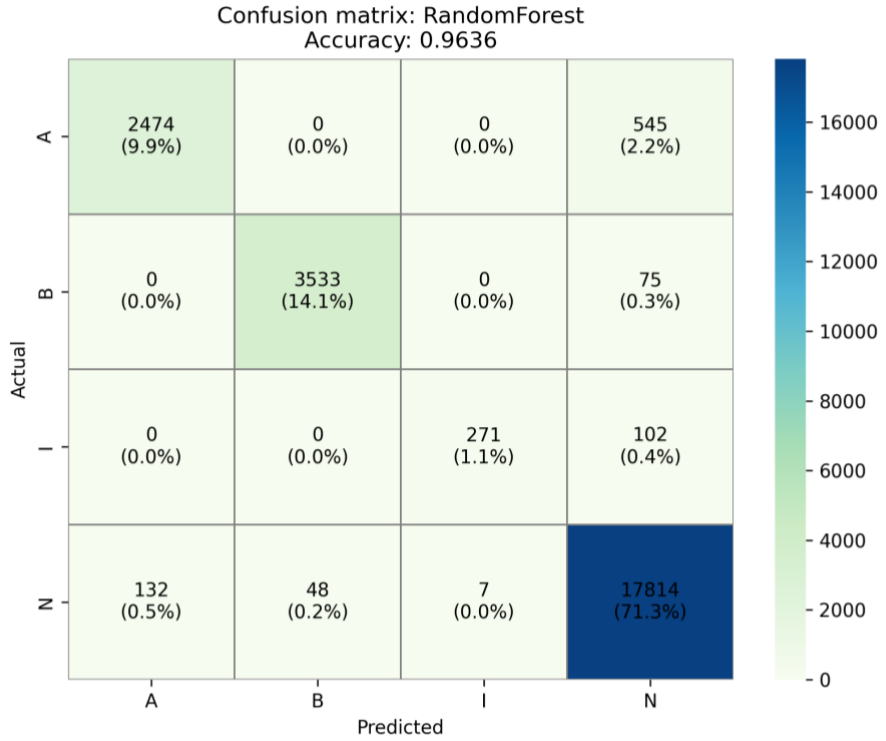
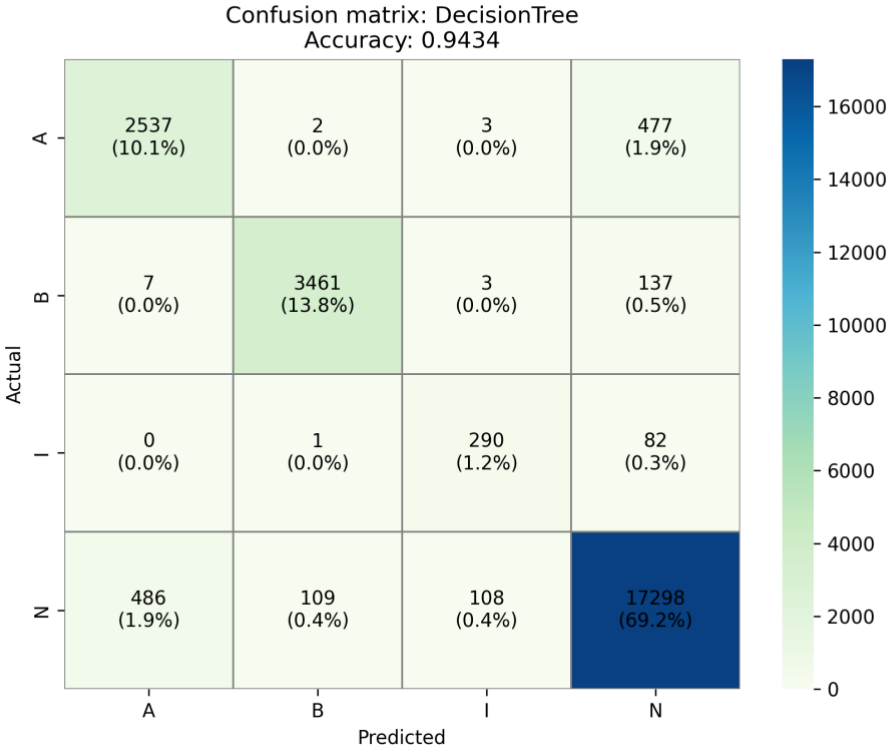


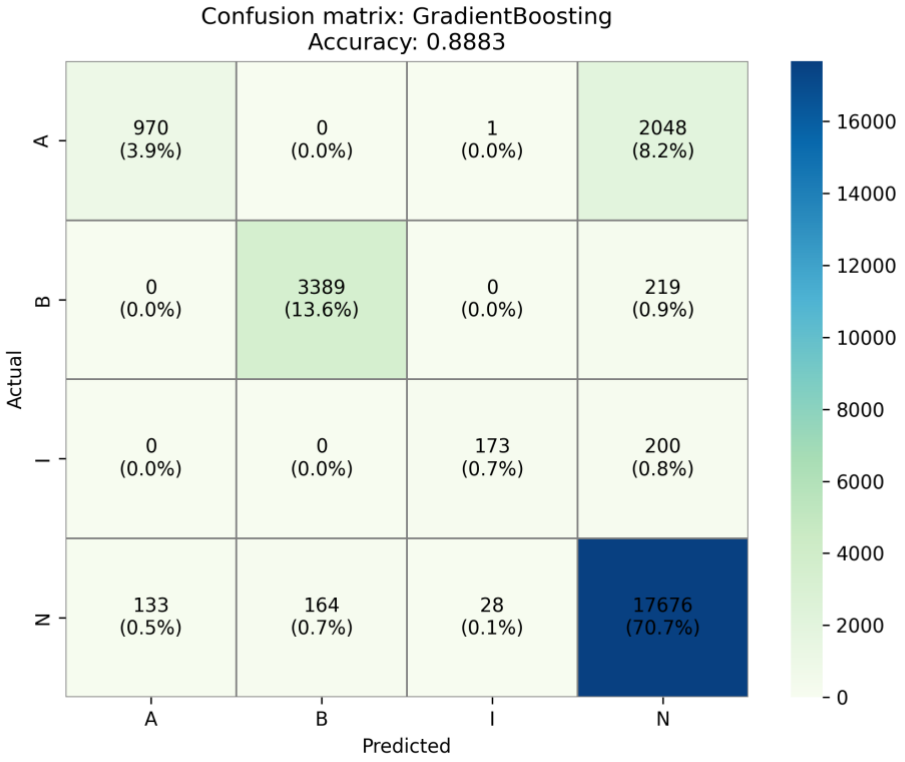
Figure 5.32 Approximate conformational space representation for ADK derived from the MD trajectory's PCA. PCA was used to identify high-density regions corresponding to free energy minima, representing the open (A), closed (B), and intermediate (I) states. Low-density regions were classified as non-states (N). Each point represents a sampled conformation, and structural snapshots illustrate the biological relevance of the identified states.

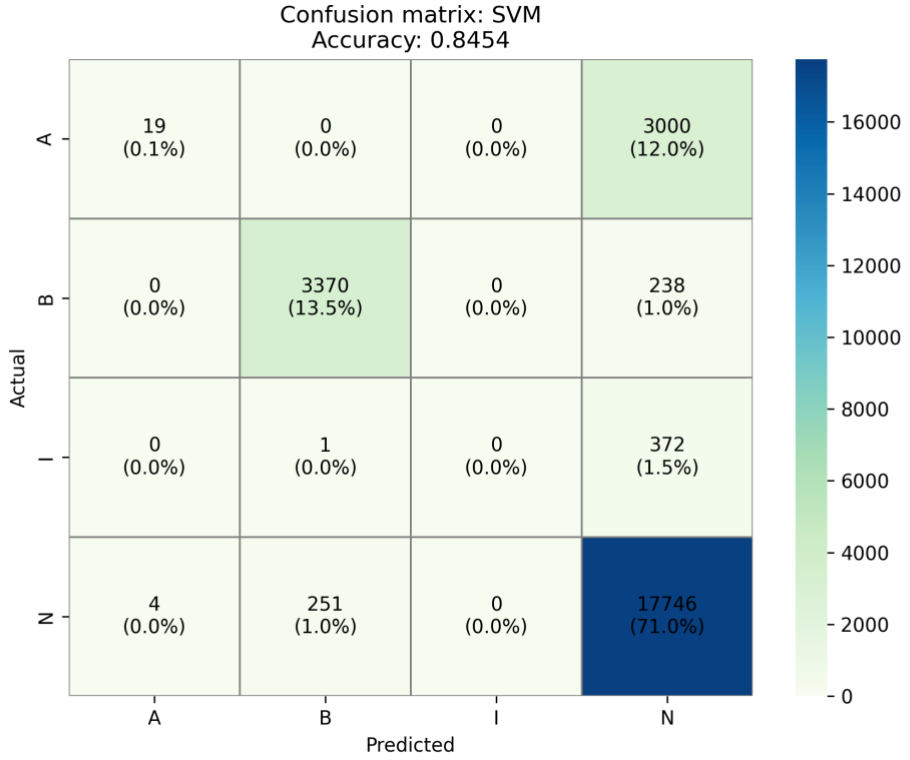
5.4.2 Machine Learning performance

Six machine learning models were tested to classify the labelled frames: Logistic Regression, Random Forest, SVM, Decision Tree, Gradient Boosting, and Multilayer Perceptron (MLP). The input dataset was split into training (70%) and testing (30%) sets to evaluate model performance. The models were compared based on accuracy, precision, recall, and F1-score metrics. Figure 5.34 shows the confusion matrices for all six models in a descending order of the best performance.


Confusion matrix: RandomForest
Accuracy: 0.9636
2474 (9.9%)
0 (0.0%)
0 (0.0%)
545 (2.2%)
0 (0.0%)
3533 (14.1%)
0 (0.0%)
75 (0.3%)
0 (0.0%)
0 (0.0%)
271 (1.1%)
102 (0.4%)
132 (0.5%)
48 (0.2%)
7 (0.0%)
17814 (71.3%)
Actual
A
B
I
N
Predicted
16000
14000
12000
10000
8000
6000
4000
2000
0


Confusion matrix: DecisionTree
Accuracy: 0.9434
2537 (10.1%)
2 (0.0%)
3 (0.0%)
477 (1.9%)
7 (0.0%)
3461 (13.8%)
3 (0.0%)
137 (0.5%)
0 (0.0%)
1 (0.0%)
290 (1.2%)
82 (0.3%)
486 (1.9%)
109 (0.4%)
108 (0.4%)
17298 (69.2%)
Actual
A
B
I
N
Predicted
16000
14000
12000
10000
8000
6000
4000
2000
0


Confusion matrix: GradientBoosting
Accuracy: 0.8883
970 (3.9%)
0 (0.0%)
1 (0.0%)
2048 (8.2%)
0 (0.0%)
3389 (13.6%)
0 (0.0%)
219 (0.9%)
0 (0.0%)
0 (0.0%)
173 (0.7%)
200 (0.8%)
133 (0.5%)
164 (0.7%)
28 (0.1%)
17676 (70.7%)
A
B
I
N
Actual
Predicted
16000
14000
12000
10000
8000
6000
4000
2000
0


Confusion matrix: SVM
Accuracy: 0.8454
19 (0.1%)
0 (0.0%)
0 (0.0%)
3000 (12.0%)
0 (0.0%)
3370 (13.5%)
0 (0.0%)
238 (1.0%)
0 (0.0%)
1 (0.0%)
0 (0.0%)
372 (1.5%)
4 (0.0%)
251 (1.0%)
0 (0.0%)
17746 (71.0%)
A
B
I
N
Actual
Predicted
16000
14000
12000
10000
8000
6000
4000
2000
0

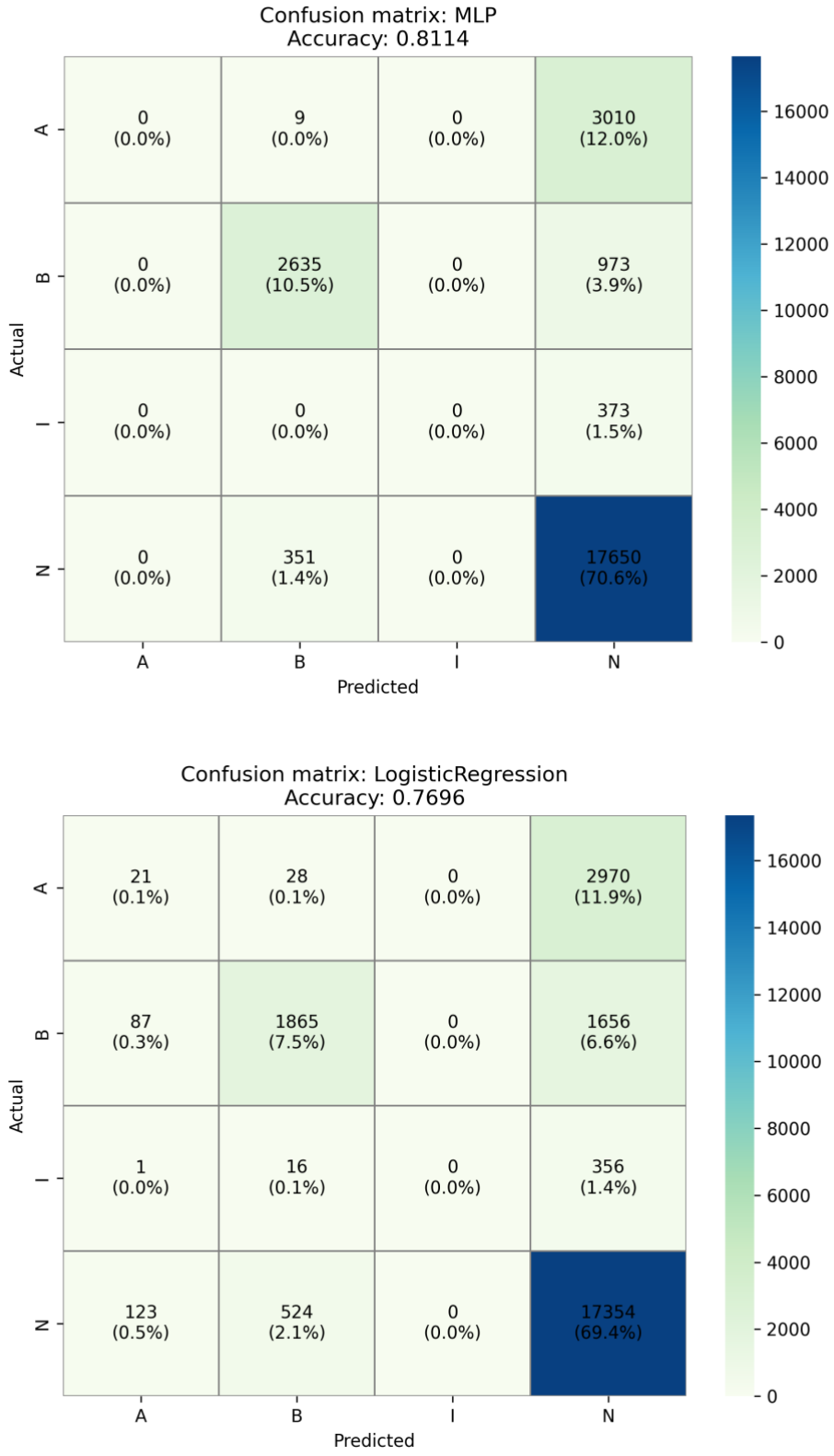


Figure 5.33 Confusion matrices for all machine learning models tested, including Random Forest, Decision Tree, Gradient Boosting, SVM, Logistic Regression, and MLP. Each matrix displays the classification performance across the four states: open (A), closed (B), intermediate (I), and non-states (N). Correct classifications are represented along the diagonal, while off-diagonal elements indicate misclassifications.

Among the models, the Random Forest classifier had the best performance with 96.36%. Its confusion matrix highlights the model's ability to accurately classify all four states with minimal misclassification across categories accurately. The Random Forest model successfully distinguished the intermediate state (I), which is often challenging due to its transitional nature, and showed strong performance for both dominant states (A and B) and the non-state (N). The weighted average F1-score for this model was 96%, making it the most reliable choice for this dataset. The Decision Tree model followed closely, with an accuracy of 94.34% Its confusion matrix revealed a comparable ability to classify the four states but with slightly higher misclassification rates for the intermediate and non-state regions, reflecting its limitations in handling less frequent categories.

The Gradient Boosting classifier reached an accuracy of 88.83%, demonstrating high precision for dominant states (A and B) but struggling to correctly identify the intermediate (I) and non-state (N) regions, as evident from its confusion matrix. The SVM achieved an accuracy of 84.54% as indicated by its confusion matrix, which showed challenges in classifying the non-state region and an inability to identify the intermediate state consistently.

The MLP and Logistic Regression models achieved accuracies of 81.14% and 76.96% respectively, with their confusion matrices reflecting difficulties in classifying the intermediate state and frequent misclassifications of the non-state region.

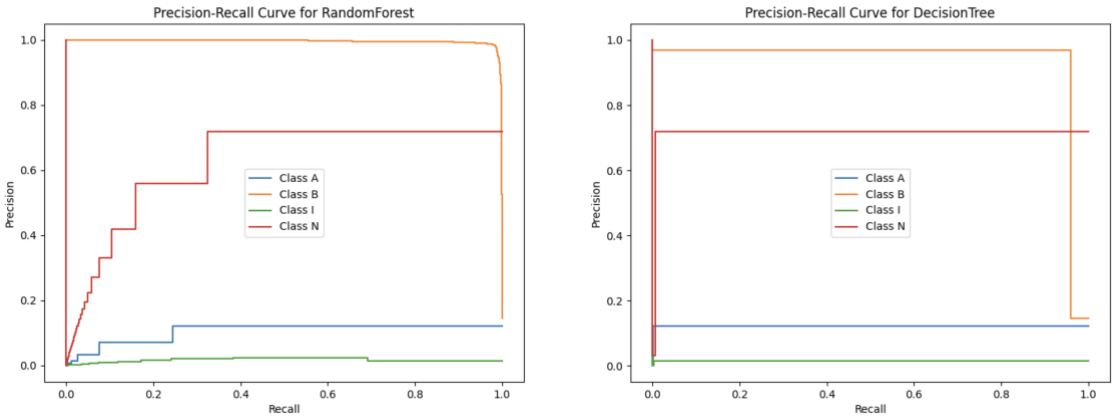


Figure 5.34 Precision-Recall (PR) curves for the Random Forest and Decision Tree models. The PR curves reflect the precision and recall trade-offs, with Random Forest maintaining higher precision and recall for all states, particularly the intermediate and non-states, compared to the Decision Tree. These results demonstrate the superior classification capabilities of the Random Forest model.

Figure 5.34 shows the Precision-Recall (PR) for the top two ML methods that achieved the highest performance: Random Forest and Decision Tree. The PR curve for the Random Forest shows its precision and recall for the dominant states (A and B). It also performs well in

identifying the intermediate state, showcasing the model's ability to handle rare and transitional conformations effectively. While slightly less effective, the PR curve for the Decision Tree model indicates robust precision for the dominant states. However, it shows slight drops in recall for the intermediate and non-states, consistent with its higher misclassification rates in these categories.

Furthermore, evaluation of the performance of the Random Forest model is achieved through the interpretation of the calibration curve plot shown in Figure 5.35. The plot shows that the model can produce well-calibrated probability predictions for each class. The diagonal dashed line represents the ideal calibration, where predicted probabilities align perfectly with the observed probabilities. The model demonstrates excellent calibration for the dominant states (Class A: Open and Class B: Closed), with predicted probabilities closely following the diagonal. Therefore, the model's confidence in these predictions is consistent with the actual values.

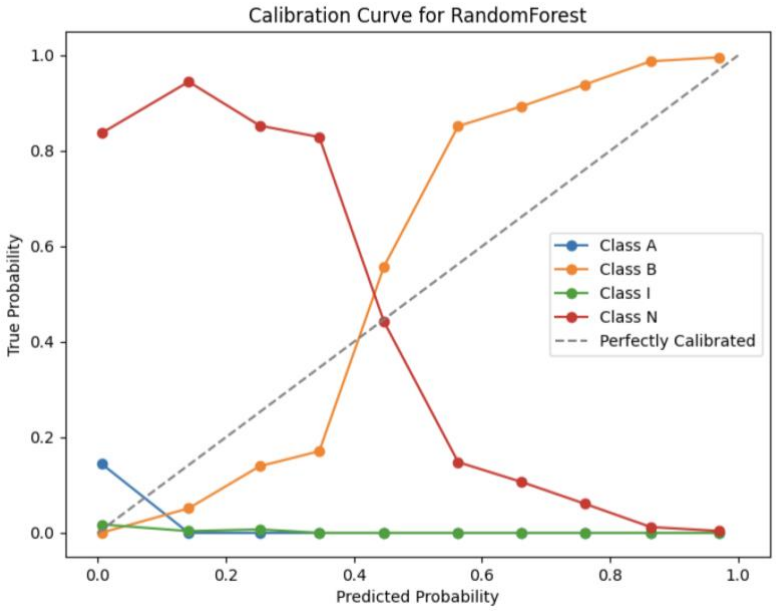


Figure 5.35 Calibration curve for the Random Forest model. The plot illustrates the relationship between predicted probabilities and observed outcomes for each class: open (A), closed (B), intermediate (I), and non-state (N). The curves for dominant states (A and B) align closely with the ideal diagonal line, indicating well-calibrated predictions. Slight deviations for the intermediate state (I) and non-state (N) reflect minor overconfidence in these predictions. Overall, the Random Forest model exhibits strong calibration across all states.

The calibration for the intermediate state (Class I) is slightly less consistent but still adequate as the predicted probabilities align reasonably well with the observed data. However, for the non-state (Class N), the curve deviates from the ideal line, indicating a slight overconfidence in predictions for this class. Despite this deviation, the model's overall calibration across all four states is superior, confirming its capability to provide reliable probability estimates.

In conclusion, this work demonstrated the utility of MDAP in constructing predictive models for state labels, building on the preprocessing capabilities of MDSS and the mutation analysis framework of MDAM. The MDSS toolkit's robust capabilities inspired the approach, and the findings emphasise its potential for broader integration with other computational methods in the field. Notably, the models struggled with I state label, a less represented class, which reflects the dual challenge of sparse data representation and the intricate nature of non-canonical intermediate states. These states require enhanced characterisation and refinement in their original labelling to support more robust predictions. Addressing class imbalance through targeted sampling strategies and advanced algorithms holds promise for improving performance in such challenging cases. This work lays the foundation for further exploration of automated tools and workflows in computational protein design, paving the way for more comprehensive and scalable analyses of dynamic biomolecular systems.

5.5 Summary

This chapter presented the results of the MDSS, MDAM, and MDAP toolkits, highlighting their ability to address key challenges in protein dynamics research. MDSS effectively reduced data complexity while preserving critical information, MDAM validated mutation impacts on protein dynamics, and MDAP demonstrated the potential for ML-based state classification. Together, these tools provide a cohesive automated framework for the redesign of protein dynamics.

6 Summary, conclusions, and further work

This chapter provides a summary of this thesis and concludes with the key outcomes achieved by developing an automated framework for redesigning protein dynamics. It highlights the contributions of the MDSubSampler, MDAutoMut, and MDAutoPredict tools and their integration into a unified workflow. Limitations of this research study are presented and discussed. Additionally, suggestions for future work are provided, outlining how the outcomes of this study can be expanded and applied to address challenges in protein engineering and computational biology.

6.1 Summary

This thesis presented a comprehensive framework for addressing some of the challenges in redesigning protein dynamics. The framework was designed by developing three novel computational tools — MDSubSampler, MDAutoMut, and MDAutoPredict — each designed to address a specific gap in the field.

The research followed a systematic approach to achieve the initial goal of this PhD: contributing novel methodologies for redesigning protein function. Considering the relationship between mutations, protein dynamics, and their functional outcomes, the study focused on redesigning protein dynamics associated with function. Given the lack of automated methods and unified strategies, the research aimed to develop a fully automated library with a flexible design to integrate customisable workflows.

The first step was the development of MDSubSampler, a framework designed to clean, prepare and reduce MD simulations by performing *a posteriori* subsampling of the data. It was built as a user-friendly toolkit applicable to various research problems. Validation was performed using simple example scenarios, demonstrating its adaptability and utility across different applications. MDSubSampler facilitates faster and more efficient workflows by providing cleaned input data, making it easier to deal with MD data when using automated workflows like MDAutoMut. MDSubSampler also enabled property comparison within MDAutoMut, supporting the assessment of whether a given mutation achieves the desired dynamic behaviour.

Following the development of MDSubSampler, MDAutoMut was designed to automate mutation analysis and directly record the effects of mutations on protein dynamics. Given the

computational cost when dealing with MD simulations and these fully automated workflows, deploying the tool on a high-performance computing (HPC) system became a critical step. Therefore, the toolkit was deployed and tested on ARCHER2. Validation of MDAutoMut involved a proof-of-concept approach where specific positions and mutations (known from a study that can change the dynamics of the example system used – ADK) were tested to determine whether the toolkit could identify them automatically. However, the complexity of searching the mutational space, even with just two positions, highlighted the need for a heuristic approach, for which a simple demonstrative example was developed and tested.

Finally, MDAutoPredict was implemented as an extension tool of MDAutoMut to integrate ML into the workflow. This tool was validated by benchmarking and testing various ML methods to predict protein states. A proof-of-concept example was created with a state label for training and testing. Designed with flexibility, MDAutoPredict can be adapted or extended to address other research problems beyond state prediction, making it a versatile addition to the automated library. It is foreseeable to use MDAutoPredict to generate the values of the target variable for MDAutoMut redesign workflows in cases where the calculation of this can be too expansive for an automated large-scale study (e.g. when a state label may require free energy reconstruction to be calculated).

Together, the tools developed in this research study tackle some of the critical challenges in protein design, forming a unified yet modular framework through a fully Python-based interface.

6.2 Conclusions

This thesis demonstrates the successful development and application of the components for a modular computational framework to redesign protein dynamics. The three tools—MDSubSampler, MDAutoMut, and MDAutoPredict —offer the possibility of building workflows for MD data preprocessing, automated mutation analysis, measurement of mutations' impact on dynamics, and prediction of protein states. Each tool addresses critical gaps in the computational design of protein dynamics.

MDSubSampler's ability to *a posteriori* extract representative subsets of trajectories has proven essential for reducing noise, retaining biologically relevant information, and formatting the MD data for ML/DL applications. This tool provides a flexible and general-purpose solution that complements existing clustering methods, enabling workflows applicable to different

research problems. Additionally, MDSubSampler's abilities extend to allow the evaluation of protein dynamics through statistical methods. Its integration into MDAutoMut highlights its importance as the foundational step in the full computational framework of this PhD research work.

The development and validation of MDAutoMut contribute to addressing some of the challenges of redesigning protein dynamics. One of the main challenges faced during the validation of MDAutoMut was undersampling, particularly when studying the dynamic behaviour of ADK. This system was chosen as an example for the toolkits' validation due to its literature-documented mutations that shift its structure towards the closed state. A clear example of redesigning protein dynamics that could be used to perform the following proof-of-concept: given the positions of two mutations on the ADK LID's domain, the MDAutoMut toolkit can identify the two mutations that have the desired impact on the system's dynamics.

While the double mutation identified in the literature was expected to induce closure consistently, it was observed that it only sampled the closed state more frequently than the WT but not consistently across all simulations. This variability presents another limitation in the tool's validation process.

To overcome this challenge, the following validation approach was used: Short simulations (50 ps-500 ps) were initially generated to test the functionality of MDAutoMut and its components. Then, longer simulations (300 ns, 500 ns, 1000 ns) were performed on HPC platforms, allowing the system to adequately sample the desired closed state. Finally, the full workflow of MDAutoMut was tested with these pre-calculated longer-timescale simulations. This testing strategy was critical to demonstrating that, given sufficient data where the closure is observed, MDAutoMut can effectively identify and analyse the relevant mutations.

The deployment of MDAutoMut on HPC, particularly ARCHER2, was essential to demonstrate the tool's scalability for larger and more complex systems. Given MD data's computational demands and the workflow's iterative nature, an HPC platform was the only feasible option. However, deploying MDAutoMut on ARCHER2 came with some challenges, mainly due to the need to install and configure multiple external libraries required by the workflow. An aspect of automated mutation scanning not explored was identifying the residue's position to change. The current implementation of MDAutoMut can scale up the number of sites but still requires identification of which sites to mutate from the user.

Building on the success of MDAutoMut, the extension into MDAutoPredict aimed to integrate ML capabilities into the framework, enabling the predictions of protein states. MDAutoPredict was validated with a simple example of ADK's state prediction. The labelling exercise of the target variable was done independently of the predictive model to prevent bias in the predictive analysis.

When implementing MDAutoPredict, certain limitations need to be considered. The quality and reliability of predictions are linked to the representativeness of the training data. Rare states, such as the closed conformation of ADK, may not appear frequently enough in the dataset, presenting challenges for generalisation. Therefore, the example data used for validation contained a sampling of both open and closed states of the system to ensure efficient tool testing.

In summary, this thesis presents a modular computational framework applicable to computational protein design. While the framework's validation highlighted challenges like undersampling and data representativeness, it also demonstrated the tools' potential to facilitate workflows of rational protein design of dynamics.

6.3 Current limitation and future development of this research study

The three toolkits' design, implementation and testing lay the foundation for an automated modular framework that can redesign protein dynamics. However, there is significant potential for their extension and refinement to address more complex challenges in computational protein design.

For MDSubSampler, future work could focus on extending its ability to evaluate also physicochemical or energetic properties, aside from geometric or incorporating energy-based criteria for selecting trajectory frames. This would improve its effectiveness in capturing transitions important to protein function. Given its integration with MDAutoPredict, MDSubSampler could be further refined to facilitate the preprocessing of trajectory data specifically for ML workflows. For instance, expanding MDSubSampler to include automated feature extraction tailored to ML models would reduce manual intervention and ensure optimal input quality for predictive tasks. Additionally, combining MDSubSampler with clustering approaches could offer a more robust framework for identifying and classifying conformational

states, supporting MDAutoPredict's predictive capabilities and creating automated pipelines for protein engineering applications.

For MDAutoMut, future work could focus on integrating advanced sampling strategies, such as adaptive or enhanced sampling techniques, to address the challenge of undersampling and improve the observation of rare but functionally important states. Refining the heuristic mutation scanning process to handle more complex space searches, including cases involving more than two mutations, would enhance the tool's versatility and applicability. Optimising the computational efficiency of the workflow would allow MDAutoMut to scale effectively, enabling its application to larger protein systems and broader mutational spaces. Finally, a significant development area would be the addition of methodology to scan putative position, possibly through preliminary analysis using directly *PyRosetta*'s static design components to identify candidate residue positions.

Future work on MDAutoPredict could expand its predictive capabilities to include tasks such as predicting protein stability, ligand binding affinity, or allosteric site activity, potentially through multi-task learning frameworks. A foreseeable extension of this framework would involve state prediction, where MDAutoPredict is integrated with MDAutoMut to test the ability of identified mutations to generate desired protein states. MDAutoPredict could refine the mutational selection process by annotating mutations with their state-prediction outcomes, enabling a more targeted approach to engineering protein dynamics and extending the application of MDAutoMut to cases where state labels can only be generated costly by free energy reconstruction. Additionally, incorporating enhanced sampling techniques and experimental data could improve the diversity and representativeness of training datasets.

Making these toolkits available as open-source software is key to enabling the proposed extensions. It allows collaboration, transparency, and adaptability, allowing the scientific community to refine and expand their capabilities to address evolving challenges in protein engineering. To this end, each tool is developed under GNU General Public Licence and has been and will be made available at the stage of manuscript submissions.

6.4 Addressing the research questions

The research questions established in Chapter 1 were centred on automating the analysis and redesign of protein dynamics, evaluating the role of mutation engineering on protein dynamics, and leveraging machine learning for predicting system states. These questions were directly

addressed through the development and implementation of the three toolkits introduced in this thesis:

- **RQ1: Processing MD simulation data:** *How can the volume of MD simulation data be effectively managed to enable its use in automated workflows without exceeding computational resource limitations? At the same time, how can the complexity of these data formats be addressed to ensure easy integration into ML/DL pipelines?*

  This question was addressed through the development of MDSubSampler, a toolkit designed for rational *a posteriori* subsampling of trajectory data. MDSubSampler implements statistical techniques to reduce the number of frames while preserving the dynamic diversity of the system. In addition to this, MDSubSampler offers format conversion and trajectory preparation functions compatible with downstream statistical analysis and machine learning workflows. The evaluation in Chapter 5 demonstrated that subsampled datasets maintained essential properties of the full trajectory, as shown by RMSD distribution and Bhattacharyya distance metric.

- **RQ2: Automating protein dynamics redesign:** *How can computational strategies be developed to fine-tune protein dynamics through targeted mutations, and how can these workflows be automated, integrated with existing computational libraries, and scaled effectively using high-performance computing (HPC) resources?*

  This was fulfilled by the development of MDAutoMut, a pipeline that automates mutation engineering, simulation preparation, execution, and trajectory comparison of dynamics. The tool integrates with GROMACS and PyRosetta and is tailored for high-throughput mutation scanning. By evaluating changes in geometric descriptors and PCA space, MDAutoMut enables systematic comparison of mutant behaviour against the wild-type system. Application to the ADK protein system showed that the pipeline could reproduce experimentally known mutation effects and identify new candidates that shift conformational behaviour.

- **RQ3: Predictive modelling for protein dynamics:** *How can the automated workflow be extended to perform predictions in combination with ML/DL pipelines, and which ML models are most appropriate for predictive tasks in protein dynamics?*

This question was explored through the implementation of MDAutoPredict, a toolkit for prediction tasks on MD data using ML. Specifically, MDAutoPredict enables supervised learning prediction of protein states. In the test case with ADK, ML classifiers such as Decision Trees and Random Forests were trained to distinguish between open and closed conformational states.

6.5 Lessons learned and future recommendations

Reflecting on the design and implementation of the automated pipeline presented in this thesis, several practical insights emerged that may inform future developments of computational workflows for protein dynamics redesign.

First, modularity proved to be a critical factor in enabling flexibility and scalability. The decision to separate the pipeline into distinct components for data preparation (MDSubSampler), dynamics redesign and MD simulation (MDAutoMut), and system state prediction (MDAutoPredict) allowed each tool to be developed, tested, and refined independently. This modular approach also facilitates integration with existing libraries and adaptation to new systems or workflows.

Second, the importance of preprocessing and standardisation became increasingly evident. A significant portion of development time was devoted to handling file format inconsistencies, trajectory cleaning, and feature extraction. These steps, although often underestimated, had a direct impact on the performance of MD analysis and ML models. Future pipelines would benefit from adopting more standardised data transformation procedures and shared libraries for input/output handling.

Third, while heuristic-based mutation analysis was effective in guiding exploration of dynamic behaviour, there is potential to expand this approach through learning-driven design strategies. For instance, reinforcement learning or optimisation algorithms could be employed in future work to prioritise mutations based on dynamic responses or predictive uncertainty.

Fourth, sampling limitations emerged as a constraint on the performance of ML models, particularly in distinguishing between states that were underrepresented in the simulation data. Incorporating enhanced sampling techniques or active learning approaches could improve coverage of conformational space and support more robust model training.

Finally, although the pipeline was designed to be automated, domain knowledge remained essential throughout the study. Expert input was necessary to define target variables, interpret ML outputs, and assess biological relevance. This suggests that future developments should aim not to remove human input entirely, but to structure it in a way that supports decision-making within the automated workflow.

In summary, these lessons highlight the need for future automated pipelines to balance efficiency and flexibility, leverage data-driven methods alongside domain expertise, and incorporate feedback mechanisms to improve design outcomes iteratively. The integration of such features may enable more generalisable and intelligent frameworks for protein dynamics analysis and design.

Bibliography

[1] Ł. Nierzwicki and G. Palermo, “Molecular Dynamics to Predict Cryo-EM: Capturing Transitions and Short-Lived Conformational States of Biomolecules,” 2018, doi: 10.3389/fmolb.2021.641208.

[2] M. J. Abraham *et al.*, “GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers,” *SoftwareX*, vol. 1–2, pp. 19–25, Sep. 2015, doi: 10.1016/J.SOFTX.2015.06.001.

[3] K. Zhou and B. Liu, *Molecular Dynamics Simulation: Fundamentals and Applications*. Elsevier, 2022. doi: 10.1016/B978-0-12-816419-8.00001-5.

[4] E. Prašnikar, M. Ljubič, A. Perdih, and J. Borišek, “Machine learning heralding a new development phase in molecular dynamics simulations,” *Artificial Intelligence Review 2024 57:4*, vol. 57, no. 4, pp. 1–36, Mar. 2024, doi: 10.1007/S10462-024-10731-4.

[5] “Press release: The Nobel Prize in Chemistry 2024 - NobelPrize.org.” Accessed: Jan. 31, 2025. [Online]. Available: https://www.nobelprize.org/prizes/chemistry/2024/press-release/

[6] S. Gu *et al.*, “Can molecular dynamics simulations improve predictions of protein-ligand binding affinity with machine learning?,” *Brief Bioinform*, vol. 24, no. 2, pp. 1–16, Mar. 2023, doi: 10.1093/BIB/BBAD008.

[7] F. Noé, G. De Fabritiis, and C. Clementi, “Machine learning for protein folding and dynamics,” *Curr Opin Struct Biol*, vol. 60, pp. 77–84, Feb. 2020, doi: 10.1016/J.SBI.2019.12.005.

[8] N. Michaud-Agrawal, E. J. Denning, T. B. Woolf, and O. Beckstein, “MDAnalysis: A toolkit for the analysis of molecular dynamics simulations,” *J Comput Chem*, vol. 32, no. 10, pp. 2319–2327, Jul. 2011, doi: 10.1002/JCC.21787.

[9] S. Chaudhury, S. Lyskov, and J. J. Gray, “PyRosetta: a script-based interface for implementing molecular modeling algorithms using Rosetta,” *Bioinformatics*, vol. 26, no. 5, pp. 689–691, Jan. 2010, doi: 10.1093/BIOINFORMATICS/BTQ007.

[10] D. L. Nelson, M. Cox, and A. A. Hoskins, “Lehninger Principes of biochemistry,” p. 868, 2021, Accessed: Dec. 21, 2024. [Online]. Available: https://www.bruna.nlhttps://www.bruna.nl/engelse-boeken/lehninger-principles-of-biochemistry-9781319381493

[11] J. Jumper *et al.*, “Highly accurate protein structure prediction with AlphaFold,” *Nature 2021 596:7873*, vol. 596, no. 7873, pp. 583–589, Jul. 2021, doi: 10.1038/s41586-021-03819-2.

[12] K. Henzler-Wildman and D. Kern, “Dynamic personalities of proteins,” *Nature*, vol. 450, no. 7172, pp. 964–972, Dec. 2007, doi: 10.1038/NATURE06522.

[13] A. V. Finkelstein, N. S. Bogatyreva, D. N. Ivankov, and S. O. Garbuzynskiy, “Protein folding problem: enigma, paradox, solution,” *Biophysical Reviews 2022 14:6*, vol. 14, no. 6, pp. 1255–1272, Oct. 2022, doi: 10.1007/S12551-022-01000-1.

[14] C. H. Rodrigues, D. E. Pires, D. B. Ascher, I. RenéRen, R. Rachou, and F. Oswaldo Cruz, “DynaMut: predicting the impact of mutations on protein conformation, flexibility and stability,” *Nucleic Acids Res*, vol. 46, no. W1, pp. W350–W355, Jul. 2018, doi: 10.1093/NAR/GKY300.

[15] T. P. J. Knowles, M. Vendruscolo, and C. M. Dobson, “The amyloid state and its association with protein misfolding diseases,” *Nature Reviews Molecular Cell Biology 2014 15:6*, vol. 15, no. 6, pp. 384–396, May 2014, doi: 10.1038/nrm3810.

[16] F. Chiti and C. M. Dobson, “Protein Misfolding, Amyloid Formation, and Human Disease: A Summary of Progress Over the Last Decade,” *Annu Rev Biochem*, vol. 86, pp. 27–68, Jun. 2017, doi: 10.1146/ANNUREV-BIOCHEM-061516-045115.

[17] D. D. Wang, L. Ou-Yang, H. Xie, M. Zhu, and H. Yan, “Predicting the Impacts of Mutations on Protein-Ligand Affinity Based on Molecular Dynamics Simulations and Machine Learning Methods,” *Comput Struct Biotechnol J*, vol. 18, pp. 439–454, Jan. 2020, doi: 10.1016/J.CSBJ.2020.02.007.

[18] Y. Peng, E. Alexov, and S. Basu, “Structural perspective on revealing and altering molecular functions of genetic variants linked with diseases,” *Int J Mol Sci*, vol. 20, no. 3, Feb. 2019, doi: 10.3390/IJMS20030548.

[19] D. Thirumalai, C. Hyeon, P. I. Zhuravlev, and G. H. Lorimer, “Symmetry, Rigidity, and Allosteric Signaling: From Monomeric Proteins to Molecular Machines,” *Chem Rev*, vol. 119, no. 12, pp. 6788–6821, Dec. 2018, doi: 10.1021/acs.chemrev.8b00760.

[20] P. E. Wright and H. J. Dyson, “Intrinsically disordered proteins in cellular signalling and regulation,” *Nat Rev Mol Cell Biol*, vol. 16, no. 1, pp. 18–29, Dec. 2015, doi: 10.1038/NRM3920.

[21] V. N. Uversky, “Intrinsically disordered proteins and their ‘Mysterious’ (meta)physics,” *Front Phys*, vol. 7, no. FEB, p. 416379, Feb. 2019, doi: 10.3389/FPHY.2019.00010/BIBTEX.

[22] A. F. Dishman and B. F. Volkman, “Design and discovery of metamorphic proteins,” *Curr Opin Struct Biol*, vol. 74, p. 102380, Jun. 2022, doi: 10.1016/J.SBI.2022.102380.

[23] C. I. Branden and J. Tooze, “Introduction to Protein Structure,” *Introduction to Protein Structure*, Mar. 2012, doi: 10.1201/9781136969898.

[24] K. A. Dill and J. L. MacCallum, “The protein-folding problem, 50 years on,” *Science*, vol. 338, no. 6110, pp. 1042–1046, Nov. 2012, doi: 10.1126/SCIENCE.1219021.

[25] “File:Main protein structure levels en.svg - Wikimedia Commons.” Accessed: Jan. 20, 2025. [Online]. Available: https://commons.wikimedia.org/wiki/File:Main_protein_structure_levels_en.svg

[26] O. Rivoire, “A role for conformational changes in enzyme catalysis,” *Biophys J*, vol. 123, no. 12, pp. 1563–1578, Jun. 2024, doi: 10.1016/J.BPJ.2024.04.030.

[27] D. Petrovic, V. A. Risso, S. C. L. Kamerlin, and J. M. Sanchez-Ruiz, “Conformational dynamics and enzyme evolution,” *J R Soc Interface*, vol. 15, no. 144, Jul. 2018, doi: 10.1098/RSIF.2018.0330.

[28] C. Moreira, A. R. Calixto, J. P. Richard, S. Caroline, and L. Kamerlin, “The role of ligand-gated conformational changes in enzyme catalysis,” 2019, doi: 10.1042/BST20190298.

[29] J. Kim, S. Moon, T. D. Romo, Y. Yang, E. Bae, and G. N. Phillips, “Conformational dynamics of adenylate kinase in crystals,” *Structural Dynamics*, vol. 11, no. 1, p. 014702, Jan. 2024, doi: 10.1063/4.0000205.

[30] S. M. Salehi, M. Pezzella, A. Willard, M. Meuwly, and M. Karplus, “Water dynamics around T0 vs R4 of hemoglobin from local hydrophobicity analysis,” *J Chem Phys*, vol. 158, no. 2, Jan. 2023, doi: 10.1063/5.0129990.

[31] M. D. Miller and G. N. Phillips, “Moving beyond static snapshots: Protein dynamics and the Protein Data Bank,” *Journal of Biological Chemistry*, vol. 296, Jan. 2021, doi: 10.1016/J.JBC.2021.100749/ASSET/21D8CC6D-3302-468E-8F8F-207E11235448/MAIN.ASSETS/GR4.JPG.

[32] R. Nussinov, Y. Liu, W. Zhang, and H. Jang, “Protein conformational ensembles in function: roles and mechanisms,” *RSC Chem Biol*, vol. 4, no. 11, pp. 850–864, Nov. 2023, doi: 10.1039/D3CB00114H.

[33] S. H. A. Raza, R. Zhong, X. Yu, G. Zhao, X. Wei, and H. Lei, “Advances of Predicting Allosteric Mechanisms Through Protein Contact in New Technologies and Their Application,” *Mol Biotechnol*, vol. 66, no. 12, pp. 3385–3397, Dec. 2023, doi: 10.1007/S12033-023-00951-4/METRICS.

[34] G. Hu, P. Doruker, H. Li, and E. Demet Akten, “Editorial: Understanding Protein Dynamics, Binding and Allostery for Drug Design,” *Front Mol Biosci*, vol. 8, p. 681364, Apr. 2021, doi: 10.3389/FMOLB.2021.681364/BIBTEX.

[35] M. B. Kubitzki, B. L. de Groot, and D. Seeliger, “Protein Dynamics : From Structure to Function,” *From Protein Structure to Function with Bioinformatics: Second Edition*, pp. 393–425, Apr. 2017, doi: 10.1007/978-94-024-1069-3_12.

[36] P. J. Heckmeier, J. Ruf, C. Rochereau, and P. Hamm, “A billion years of evolution manifest in nanosecond protein dynamics,” *Proc Natl Acad Sci U S A*, vol. 121, no. 10, Sep. 2023, doi: 10.1073/pnas.2318743121.

[37] T. Hett *et al.*, “Spatiotemporal Resolution of Conformational Changes in Biomolecules by Combining Pulsed Electron−Electron Double Resonance Spectroscopy with Microsecond Freeze-Hyperquenching,” *Cite This: J. Am. Chem. Soc*, vol. 143, pp. 6981–6989, 2021, doi: 10.1021/jacs.1c01081.

[38] D. Budday, S. Leyendecker, and H. Van Den Bedem, “Kinematic Flexibility Analysis: Hydrogen Bonding Patterns Impart a Spatial Hierarchy of Protein Motion,” *J Chem Inf Model*, vol. 58, no. 10, pp. 2108–2122, Feb. 2018, doi: 10.1021/acs.jcim.8b00267.

[39] C. Narayanan, D. N. Bernard, and N. Doucet, “Role of Conformational Motions in Enzyme Function: Selected Methodologies and Case Studies,” *Catalysts 2016, Vol. 6, Page 81*, vol. 6, no. 6, p. 81, May 2016, doi: 10.3390/CATAL6060081.

[40] G. Haran and H. Mazal, “How fast are the motions of tertiary-structure elements in proteins?,” *J Chem Phys*, vol. 153, no. 13, Oct. 2020, doi: 10.1063/5.0024972.

[41] O. Bozovic *et al.*, “Real-time observation of ligand-induced allosteric transitions in a PDZ domain,” *Proc Natl Acad Sci U S A*, vol. 117, no. 42, pp. 26031–26039, Oct. 2020, doi: 10.1073/PNAS.2012999117/-/DCSUPPLEMENTAL.

[42] A. A. A. I. Ali, A. Gulzar, S. Wolf, and G. Stock, “Nonequilibrium Modeling of the Elementary Step in PDZ3 Allosteric Communication,” *Journal of Physical Chemistry Letters*, vol. 13, no. 42, pp. 9862–9868, Oct. 2022, doi: 10.1021/ACS.JPCLETT.2C02821/SUPPL_FILE/JZ2C02821_SI_001.PDF.

[43] V. Muñoz and M. Cerminara, “When fast is better: protein folding fundamentals and mechanisms from ultrafast approaches,” *Biochemical Journal*, vol. 473, no. 17, pp. 2545–2559, Sep. 2016, doi: 10.1042/BCJ20160107.

[44] M. Gruebele, “Protein Dynamics in Simulation and Experiment,” *J Am Chem Soc*, vol. 136, no. 48, pp. 16695–16697, Dec. 2014, doi: 10.1021/JA510614S.

[45] S. Sasidharan, V. Gosu, T. Tripathi, and P. Saudagar, “Molecular Dynamics Simulation to Study Protein Conformation and Ligand Interaction,” *Protein Folding Dynamics and Stability: Experimental and Computational Methods*, pp. 107–127, Jan. 2023, doi: 10.1007/978-981-99-2079-2_6.

[46] A. Son *et al.*, “Utilizing Molecular Dynamics Simulations, Machine Learning, Cryo-EM, and NMR Spectroscopy to Predict and Validate Protein Dynamics,” *International Journal of Molecular Sciences 2024, Vol. 25, Page 9725*, vol. 25, no. 17, p. 9725, Sep. 2024, doi: 10.3390/IJMS25179725.

[47] V. Timofeev and V. Samygina, “Protein Crystallography: Achievements and Challenges,” *Crystals 2023, Vol. 13, Page 71*, vol. 13, no. 1, p. 71, Jan. 2023, doi: 10.3390/CRYST13010071.

[48] C. Charlier, S. F. Cousin, and F. Ferrage, “Protein dynamics from nuclear magnetic relaxation,” *Chem Soc Rev*, vol. 45, no. 9, pp. 2410–2422, May 2016, doi: 10.1039/C5CS00832H.

[49] J. A. Purslow, B. Khatiwada, M. J. Bayro, and V. Venditti, “NMR Methods for Structural Characterization of Protein-Protein Complexes,” *Front Mol Biosci*, vol. 7, no. 9, p. 502193, Jan. 2020, doi: 10.3389/FMOLB.2020.00009/BIBTEX.

[50] S. C. Chiliveri and M. V. Deshmukh, “Recent excitements in protein NMR: Large proteins and biologically relevant dynamics,” *Journal of Biosciences 2016 41:4*, vol. 41, no. 4, pp. 787–803, Oct. 2016, doi: 10.1007/S12038-016-9640-Y.

[51] T. Schlick *et al.*, “Annual Review of Biophysics Biomolecular Modeling and Simulation: A Prospering Multidisciplinary Field,” 2021, doi: 10.1146/annurev-biophys-091720.

[52] C. E. Tzeliou, M. A. Mermigki, and D. Tzeli, “molecules Review Review on the QM/MM Methodologies and Their Application to Metalloproteins,” 2022, doi: 10.3390/molecules27092660.

[53] G. A. Voth, “Coarse-Graining of Condensed Phase and Biomolecular Systems,” *Coarse-Graining of Condensed Phase and Biomolecular Systems*, Sep. 2008, doi: 10.1201/9781420059564.

[54] S. J. Marrink, H. J. Risselada, S. Yefimov, D. P. Tieleman, and A. H. De Vries, “The MARTINI force field: coarse grained model for biomolecular simulations,” *J Phys Chem B*, vol. 111, no. 27, pp. 7812–7824, Jul. 2007, doi: 10.1021/JP071097F.

[55] M. R. Machado, E. E. Barrera, F. Klein, M. Sónora, S. Silva, and S. Pantano, “The SIRAH 2.0 Force Field: Altius, Fortius, Citius,” *J Chem Theory Comput*, vol. 15, no. 4, pp. 2719–2733, Apr. 2019, doi: 10.1021/ACS.JCTC.9B00006.

[56] I. Bahar and A. J. Rader, “Coarse-grained normal mode analysis in structural biology,” *Curr Opin Struct Biol*, vol. 15, no. 5, pp. 586–592, Oct. 2005, doi: 10.1016/J.SBI.2005.08.007.

[57] N. Awasthi, R. Shukla, D. Kumar, A. K. Tiwari, and T. Tripathi, “Monte Carlo Approaches to Study Protein Conformation Ensembles,” *Protein Folding Dynamics and Stability: Experimental and Computational Methods*, pp. 129–146, Jan. 2023, doi: 10.1007/978-981-99-2079-2_7.

[58] J. Machta and R. S. Ellis, “Monte Carlo Methods for Rough Free Energy Landscapes: Population Annealing and Parallel Tempering,” *J Stat Phys*, vol. 144, no. 3, pp. 541–553, Aug. 2011, doi: 10.1007/S10955-011-0249-0/METRICS.

[59] J. H. M. Van Gils *et al.*, “Chapter 15 Monte Carlo for Protein Structures CHAPTER 15. MONTE CARLO FOR PROTEIN STRUCTURES 15 Monte Carlo for Protein Structures 1,” 2023.

[60] G. Kumar, R. R. Mishra, and A. Verma, “Introduction to Molecular Dynamics Simulations,” *Lecture Notes in Applied and Computational Mechanics*, vol. 99, pp. 1–19, 2022, doi: 10.1007/978-981-19-3092-8_1.

[61] W. D. Cornell *et al.*, “ A Second Generation Force Field for the Simulation of Proteins, Nucleic Acids, and Organic Molecules J . Am . Chem . Soc . 1995 , 117 , 5179−5197 ,” *J Am Chem Soc*, vol. 118, no. 9, pp. 2309–2309, Jan. 1996, doi: 10.1021/JA955032E.

[62] A. D. MacKerell, J. Wiórkiewicz-Kuczera, M. Karplus, and A. D. MacKerell, “An All-Atom Empirical Energy Function for the Simulation of Nucleic Acids,” *J Am Chem Soc*, vol. 117, no. 48, pp. 11946–11975, 1995, doi: 10.1021/JA00153A017/SUPPL_FILE/JA11946A.PDF.

[63] B. R. Brooks, R. E. Bruccoleri, B. D. Olafson, D. J. States, S. Swaminathan, and M. Karplus, “CHARMM: A program for macromolecular energy, minimization, and dynamics calculations,” *J Comput Chem*, vol. 4, no. 2, pp. 187–217, Jun. 1983, doi: 10.1002/JCC.540040211.

[64] W. L. Jorgensen, D. S. Maxwell, and J. Tirado-Rives, “Development and testing of the OPLS all-atom force field on conformational energetics and properties of organic liquids,” *J Am Chem Soc*, vol. 118, no. 45, pp. 11225–11236, Nov. 1996, doi: 10.1021/JA9621760/SUPPL_FILE/JA11225.PDF.

[65] W. F. Van Gunsteren and H. J. C. Berendsen, “The GROMOS Software for (Bio)Molecular Simulation GROMOS87 Groningen Molecular Simulation (GROMOS) Library Manual,” 1987.

[66] R. B. Best, “Atomistic Force Fields for Proteins,” *Methods in Molecular Biology*, vol. 2022, pp. 3–19, 2019, doi: 10.1007/978-1-4939-9608-7_1.

[67] Y. I. Yang, Q. Shao, J. Zhang, L. Yang, and Y. Q. Gao, “Enhanced sampling in molecular dynamics,” *Journal of Chemical Physics*, vol. 151, no. 7, p. 70902, Aug. 2019, doi: 10.1063/1.5109531/197966.

[68] M. S. Badar, S. Shamsi, J. Ahmed, and Md. A. Alam, “Molecular Dynamics Simulations: Concept, Methods, and Applications,” pp. 131–151, 2022, doi: 10.1007/978-3-030-94651-7_7.

[69] P. E. M. Lopes, O. Guvench, and A. D. Mackerell, “Current Status of Protein Force Fields for Molecular Dynamics Simulations,” *Methods in Molecular Biology*, vol. 1215, pp. 47–71, 2015, doi: 10.1007/978-1-4939-1465-4_3.

[70] K. Lindorff-Larsen *et al.*, “Improved side-chain torsion potentials for the Amber ff99SB protein force field,” *Proteins*, vol. 78, no. 8, pp. 1950–1958, Jun. 2010, doi: 10.1002/PROT.22711.

[71] V. Hornak, R. Abel, A. Okur, B. Strockbine, A. Roitberg, and C. Simmerling, “Comparison of multiple Amber force fields and development of improved protein backbone parameters,” *Proteins*, vol. 65, no. 3, pp. 712–725, Nov. 2006, doi: 10.1002/PROT.21123.

[72] J. A. Harrison, J. D. Schall, S. Maskey, P. T. Mikulski, M. T. Knippenberg, and B. H. Morrow, “Review of force fields and intermolecular potentials used in atomistic computational materials research,” *Appl Phys Rev*, vol. 5, no. 3, p. 31104, Sep. 2018, doi: 10.1063/1.5020808/123935.

[73] C. E. Cavender *et al.*, “Structure-Based Experimental Datasets for Benchmarking of Protein Simulation Force Fields,” 2023.

[74] C. R. A. Abreu and M. E. Tuckerman, “Multiple timescale molecular dynamics with very large time steps: avoidance of resonances,” *The European Physical Journal B 2021 94:11*, vol. 94, no. 11, pp. 1–13, Nov. 2021, doi: 10.1140/EPJB/S10051-021-00226-4.

[75] L. Verlet, “Computer ‘Experiments’ on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules,” *Physical Review*, vol. 159, no. 1, pp. 98–103, 1967, doi: 10.1103/PhysRev.159.98.

[76] W. C. Swope, H. C. Andersen, P. H. Berens, and K. R. Wilson, “A computer simulation method for the calculation of equilibrium constants for the formation of physical clusters of molecules: Application to small water clusters,” *J Chem Phys*, vol. 76, no. 1, pp. 637–649, Jan. 1982, doi: 10.1063/1.442716.

[77] W. F. Van Gunsteren and H. J. C. Berendsen, “A LEAP-FROG ALGORITHM FOR STOCHASTIC DYNAMICS,” *Mol Simul*, vol. 1, no. 3, pp. 173–185, 1988, doi: 10.1080/08927028808080941.

[78] R. Hockney, “The potential calculation and some applications,” 1970.

[79] J. P. Ryckaert, G. Ciccotti, and H. J. C. Berendsen, “Numerical integration of the cartesian equations of motion of a system with constraints: molecular dynamics of n-alkanes,” *J Comput Phys*, vol. 23, no. 3, pp. 327–341, Mar. 1977, doi: 10.1016/0021-9991(77)90098-5.

[80] M. Tuckerman, B. J. Berne, and G. J. Martyna, “Reversible multiple time scale molecular dynamics,” *J Chem Phys*, vol. 97, no. 3, pp. 1990–2001, Aug. 1992, doi: 10.1063/1.463137.

[81] B. Hess, H. Bekker, H. Berendsen, and J. Fraaije, “LINCS: A linear constraint solver for molecular simulations,” *J Comput Chem*, 1997, doi: 10.1002/(SICI)1096-987X(199709)18:12.

[82] T. Schneider and E. Stoll, “Molecular-dynamics study of a three-dimensional one-component model for distortive phase transitions,” *Phys Rev B*, vol. 17, no. 3, pp. 1302–1322, 1978, doi: 10.1103/PhysRevB.17.1302.

[83] H. J. C. Berendsen, J. P. M. Postma, W. F. Van Gunsteren, A. Dinola, and J. R. Haak, “Molecular dynamics with coupling to an external bath,” *J Chem Phys*, vol. 81, no. 8, pp. 3684–3690, Oct. 1984, doi: 10.1063/1.448118.

[84] G. Bussi, D. Donadio, and M. Parrinello, “Canonical sampling through velocity rescaling,” *J Chem Phys*, vol. 126, no. 1, 2007, doi: 10.1063/1.2408420.

[85] M. Parrinello and A. Rahman, “Polymorphic transitions in single crystals: A new molecular dynamics method,” *J Appl Phys*, vol. 52, no. 12, pp. 7182–7190, 1981, doi: 10.1063/1.328693.

[86] W. L. Jorgensen *et al.*, “Comparison of simple potential functions for simulating liquid water,” *JChPh*, vol. 79, no. 2, pp. 926–935, 1983, doi: 10.1063/1.445869.

[87] H. J. C. Berendsen, J. R. Grigera, and T. P. Straatsma, “The missing term in effective pair potentials,” *Journal of Physical Chemistry*, vol. 91, no. 24, pp. 6269–6271, 1987, doi: 10.1021/J100308A038/ASSET/J100308A038.FP.PNG_V03.

[88] J. Wang, R. M. Wolf, J. W. Caldwell, P. A. Kollman, and D. A. Case, “Development and testing of a general amber force field,” *J Comput Chem*, vol. 25, no. 9, pp. 1157–1174, Jul. 2004, doi: 10.1002/JCC.20035.

[89] W. Clark Still, A. Tempczyk, R. C. Hawley, and T. Hendrickson, “Semianalytical Treatment of Solvation for Molecular Mechanics and Dynamics,” *J Am Chem Soc*, vol. 112, no. 16, pp. 6127–6129, 1990, doi: 10.1021/JA00172A038/SUPPL_FILE/JA00172A038_SI_001.PDF.

[90] B. Honig and A. Nicholls, “Classical electrostatics in biology and chemistry,” *Science*, vol. 268, no. 5214, pp. 1144–1149, 1995, doi: 10.1126/SCIENCE.7761829.

[91] M. P. Allen and D. J. Tildesley, “Computer Simulation of Liquids (Oxford Science Publications) SE - Oxford science publications,” *Oxford University Press*, vol. 45, p. 408, 1989, Accessed: Jan. 17, 2025. [Online]. Available: https://books.google.com/books/about/Computer_Simulation_of_Liquids.html?id=O32VXB9e5P4C

[92] T. Cao, X. Ji, J. Wu, S. Zhang, and X. Yang, “Correction of diffusion calculations when using two types of non-rectangular simulation boxes in molecular simulations,” *J Mol*

Model, vol. 25, no. 1, pp. 1–10, Jan. 2019, doi: 10.1007/S00894-018-3910-6/FIGURES/7.

[93] A. R. Leach, “Empirical Force Field Models: Molecular Mechanics,” *Molecular Modelling: Principles and Applications*, pp. 165–252, 2001, Accessed: Dec. 21, 2024. [Online]. Available: https://books.google.com/books/about/Molecular_Modelling.html?id=kB7jsbV-uhkC

[94] T. Darden, D. York, and L. Pedersen, “Particle mesh Ewald: An N·log(N) method for Ewald sums in large systems,” *J Chem Phys*, vol. 98, no. 12, pp. 10089–10092, Jun. 1993, doi: 10.1063/1.464397.

[95] U. Essmann, L. Perera, M. L. Berkowitz, T. Darden, H. Lee, and L. G. Pedersen, “A smooth particle mesh Ewald method,” *J Chem Phys*, vol. 103, no. 19, pp. 8577–8593, Nov. 1995, doi: 10.1063/1.470117.

[96] A. Y. Toukmaji and J. A. Board, “Ewald summation techniques in perspective: a survey,” *Comput Phys Commun*, vol. 95, no. 2–3, pp. 73–92, Jun. 1996, doi: 10.1016/0010-4655(96)00016-1.

[97] S. Hayward, “A Retrospective on the Development of Methods for the Analysis of Protein Conformational Ensembles,” *Protein Journal*, vol. 42, no. 3, pp. 181–191, Jun. 2023, doi: 10.1007/S10930-023-10113-9/FIGURES/3.

[98] J. Machine, L.-E. Zheng, S. Barethiya, E. Nordquist, and J. Chen, “Citation: molecules Machine Learning Generation of Dynamic Protein Conformational Ensembles,” 2023, doi: 10.3390/molecules28104047.

[99] N. Oues, S. C. Dantu, R. J. Patel, and A. Pandini, “MDSubSampler: a posteriori sampling of important protein conformations from biomolecular simulations,” *Bioinformatics*, vol. 39, no. 7, Jul. 2023, doi: 10.1093/BIOINFORMATICS/BTAD427.

[100] S. Kullback and R. A. Leibler, “On Information and Sufficiency,” *The Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79–86, Mar. 1951, doi: 10.1214/aoms/1177729694.

[101] A. Bhattacharyya, “On a Measure of Divergence between Two Multinomial Populations,” *The Indian Journal of Statistics*, vol. 7, no. 4, pp. 401–406, 1933, Accessed: Apr. 08, 2023. [Online]. Available: https://www.jstor.org/stable/25047882

[102] K. Pearson, “VII. Note on regression and inheritance in the case of two parents,” *Proceedings of the Royal Society of London*, vol. 58, no. 347–352, pp. 240–242, Dec. 1895, doi: 10.1098/RSPL.1895.0041.

[103] R. Lazim, D. Suh, and S. Choi, “Advances in Molecular Dynamics Simulations and Enhanced Sampling Methods for the Study of Protein Systems,” *Int J Mol Sci*, vol. 21, no. 17, pp. 1–20, Sep. 2020, doi: 10.3390/IJMS21176339.

[104] J. Kästner, “Umbrella sampling,” *Wiley Interdiscip Rev Comput Mol Sci*, vol. 1, no. 6, pp. 932–942, Nov. 2011, doi: 10.1002/WCMS.66.

[105] R. Qi, G. Wei, B. Ma, and R. Nussinov, “Replica exchange molecular dynamics: A practical application protocol with solutions to common problems and a peptide aggregation and self-assembly example,” *Methods in Molecular Biology*, vol. 1777, pp. 101–119, 2018, doi: 10.1007/978-1-4939-7811-3_5.

[106] G. Bussi, A. Laio, and P. Tiwary, “Metadynamics: A Unified Framework for Accelerating Rare Events and Sampling Thermodynamics and Kinetics,” *Handbook of Materials Modeling*, pp. 1–31, 2018, doi: 10.1007/978-3-319-42913-7_49-1.

[107] A. Laio 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, Nov. 2008, doi: 10.1088/0034-4885/71/12/126601.

[108] Y. Wang, J. M. Lamim Ribeiro, and P. Tiwary, “Machine learning approaches for analyzing and enhancing molecular dynamics simulations,” *Curr Opin Struct Biol*, vol. 61, pp. 139–145, Apr. 2020, doi: 10.1016/J.SBI.2019.12.016.

[109] I. D. Mienye and T. G. Swart, “A Comprehensive Review of Deep Learning: Architectures, Recent Advances, and Applications,” *Information 2024, Vol. 15, Page 755*, vol. 15, no. 12, p. 755, Nov. 2024, doi: 10.3390/INFO15120755.

[110] L. Alzubaidi *et al.*, “Review of deep learning: concepts, CNN architectures, challenges, applications, future directions,” *Journal of Big Data 2021 8:1*, vol. 8, no. 1, pp. 1–74, Mar. 2021, doi: 10.1186/S40537-021-00444-8.

[111] D. D. Wang, W. Wu, and R. Wang, “Structure-based, deep-learning models for protein-ligand binding affinity prediction,” *J Cheminform*, vol. 16, no. 1, pp. 1–15, Dec. 2024, doi: 10.1186/S13321-023-00795-9/FIGURES/8.

[112] N. A. Gonzalez, B. A. Li, and M. E. McCully, “The stability and dynamics of computationally designed proteins,” *Protein Engineering, Design and Selection*, vol. 35, Feb. 2022, doi: 10.1093/PROTEIN/GZAC001.

[113] K. Nam and M. Wolf-Watz, “Protein dynamics: The future is bright and complicated!,” *Structural Dynamics*, vol. 10, no. 1, p. 14301, Jan. 2023, doi: 10.1063/4.0000179/2876350.

[114] O. Beckstein, 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,” *J Mol Biol*, vol. 394, no. 1, pp. 160–176, Nov. 2009, doi: 10.1016/J.JMB.2009.09.009.

[115] B. Kuhlman and P. Bradley, “Advances in protein structure prediction and design,” Nov. 01, 2019, *Nature Publishing Group*. doi: 10.1038/s41580-019-0163-x.

[116] M. Karimi and Y. Shen, “iCFN: an efficient exact algorithm for multistate protein design,” *Bioinformatics*, vol. 34, no. 17, pp. i811–i820, Sep. 2018, doi: 10.1093/BIOINFORMATICS/BTY564.

[117] J. Vucinic, D. Simoncini, M. Ruffini, S. Barbe, and T. Schiex, “Positive multistate protein design,” *Bioinformatics*, vol. 36, no. 1, pp. 122–130, Jan. 2020, doi: 10.1093/BIOINFORMATICS/BTZ497.

[118] S. L. Lisanza *et al.*, “Multistate and functional protein design using RoseTTAFold sequence space diffusion,” *Nature Biotechnology 2024*, pp. 1–11, Sep. 2024, doi: 10.1038/s41587-024-02395-w.

[119] A. Pillai *et al.*, “De novo design of allosterically switchable protein assemblies,” *Nature*, vol. 632, no. 8026, pp. 911–920, Aug. 2024, doi: 10.1038/S41586-024-07813-2.

[120] Y. Bouchiba, J. Cortés, T. Schiex, and S. Barbe, “Molecular flexibility in computational protein design: an algorithmic perspective,” *Protein Engineering, Design and Selection*, vol. 34, pp. 1–7, Feb. 2021, doi: 10.1093/PROTEIN/GZAB011.

[121] J. A. Davey and R. A. Chica, “Multistate approaches in computational protein design,” *Protein Sci*, vol. 21, no. 9, p. 1241, Sep. 2012, doi: 10.1002/PRO.2128.

[122] C. Ren, X. Wen, J. Mencius, and S. Quan, “Selection and screening strategies in directed evolution to improve protein stability,” *Bioresources and Bioprocessing 2019 6:1*, vol. 6, no. 1, pp. 1–14, Dec. 2019, doi: 10.1186/S40643-019-0288-Y.

[123] V. Parthiban, M. M. Gromiha, M. Abhinandan, and D. Schomburg, “Computational modeling of protein mutant stability: Analysis and optimization of statistical potentials and structural features reveal insights into prediction model development,” *BMC Struct Biol*, vol. 7, no. 1, pp. 1–9, Aug. 2007, doi: 10.1186/1472-6807-7-54/FIGURES/6.

[124] C. Lemay-St-Denis, N. Doucet, and J. N. Pelletier, “Integrating dynamics into enzyme engineering,” *Protein Engineering, Design and Selection*, vol. 35, pp. 1–11, Feb. 2022, doi: 10.1093/PROTEIN/GZAC015.

[125] F. Jiang *et al.*, “A general temperature-guided language model to design proteins of enhanced stability and activity,” *Sci Adv*, vol. 10, no. 48, p. eadr2641, Nov. 2024, doi: 10.1126/SCIADV.ADR2641.

[126] P. S. Huang, S. E. Boyken, and D. Baker, “The coming of age of de novo protein design,” *Nature*, vol. 537, no. 7620, pp. 320–327, Sep. 2016, doi: 10.1038/NATURE19946.

[127] J. Wang *et al.*, “Scaffolding protein functional sites using deep learning,” *Science (1979)*, vol. 377, no. 6604, pp. 387–394, Jul. 2022, doi: 10.1126/SCIENCE.ABN2100/SUPPL_FILE/SCIENCE.ABN2100_DATA_S1_AND_S2.ZIP.

[128] A. Leaver-Fay *et al.*, “Rosetta3: An Object-Oriented Software Suite for the Simulation and Design of Macromolecules,” *Methods Enzymol*, vol. 487, no. C, p. 545, 2011, doi: 10.1016/B978-0-12-381270-4.00019-6.

[129] R. Guerois, J. E. Nielsen, and L. Serrano, “Predicting changes in the stability of proteins and protein complexes: a study of more than 1000 mutations,” *J Mol Biol*, vol. 320, no. 2, pp. 369–387, 2002, doi: 10.1016/S0022-2836(02)00442-4.

[130] J. Cheng *et al.*, “Accurate proteome-wide missense variant effect prediction with AlphaMissense,” *Science (1979)*, vol. 381, no. 6664, Sep. 2023, doi: 10.1126/SCIENCE.ADG7492/SUPPL_FILE/SCIENCE.ADG7492_DATA_S1_TO_S9.ZIP.

[131] J. Schymkowitz, J. Borg, F. Stricher, R. Nys, F. Rousseau, and L. Serrano, “The FoldX web server: an online force field,” *Nucleic Acids Res*, vol. 33, no. Web Server issue, Jul. 2005, doi: 10.1093/NAR/GKI387.

[132] D. E. V. Pires, D. B. Ascher, and T. L. Blundell, “DUET: a server for predicting effects of mutations on protein stability using an integrated computational approach,” *Nucleic Acids Res*, vol. 42, no. W1, pp. W314–W319, Jul. 2014, doi: 10.1093/NAR/GKU411.

[133] V. Frappier and R. J. Najmanovich, “A coarse-grained elastic network atom contact model and its use in the simulation of protein dynamics and the prediction of the effect of mutations,” *PLoS Comput Biol*, vol. 10, no. 4, 2014, doi: 10.1371/JOURNAL.PCBI.1003569.

[134] J. Fang, “A critical review of five machine learning-based algorithms for predicting protein stability changes upon mutation,” *Brief Bioinform*, vol. 21, no. 4, pp. 1285–1292, Jul. 2020, doi: 10.1093/BIB/BBZ071.

[135] J. Cheng, A. Randall, and P. Baldi, “Prediction of protein stability changes for single-site mutations using support vector machines,” *Proteins*, vol. 62, no. 4, pp. 1125–1132, Mar. 2006, doi: 10.1002/PROT.20810.

[136] E. Capriotti, P. Fariselli, and R. Casadio, “A neural-network-based method for predicting protein stability changes upon single point mutations,” *Bioinformatics*, vol. 20 Suppl 1, no. SUPPL. 1, 2004, doi: 10.1093/BIOINFORMATICS/BTH928.

[137] N. Kumar and R. Srivastava, “Deep learning in structural bioinformatics: current applications and future perspectives,” *Brief Bioinform*, vol. 25, no. 3, Mar. 2024, doi: 10.1093/BIB/BBAE042.

[138] J. Kadupitiya, G. Fox, and V. Jadhao, “Simulating Molecular Dynamics with Large Timesteps using Recurrent Neural Networks,” *arXiv.org*, 2020.

[139] A. W. Senior *et al.*, “Improved protein structure prediction using potentials from deep learning,” *Nature 2020 577:7792*, vol. 577, no. 7792, pp. 706–710, Jan. 2020, doi: 10.1038/s41586-019-1923-7.

[140] K. Tunyasuvunakool *et al.*, “Highly accurate protein structure prediction for the human proteome,” *Nature 2021 596:7873*, vol. 596, no. 7873, pp. 590–596, Jul. 2021, doi: 10.1038/s41586-021-03828-1.

[141] J. Dauparas *et al.*, “Robust deep learning-based protein sequence design using ProteinMPNN,” *Science*, vol. 378, no. 6615, pp. 49–56, Oct. 2022, doi: 10.1126/SCIENCE.ADD2187.

[142] J. Abramson *et al.*, “Accurate structure prediction of biomolecular interactions with AlphaFold 3,” *Nature 2024 630:8016*, vol. 630, no. 8016, pp. 493–500, May 2024, doi: 10.1038/s41586-024-07487-w.

[143] M. Baek *et al.*, “Accurate prediction of protein structures and interactions using a three-track neural network,” *Science (1979)*, vol. 373, no. 6557, pp. 871–876, Aug. 2021, doi: 10.1126/SCIENCE.ABJ8754/SUPPL_FILE/ABJ8754_MDAR_REPRODUCIBILITY_CHECKLIST.PDF.

[144] I. Anishchenko *et al.*, “De novo protein design by deep network hallucination,” *Nature 2021 600:7889*, vol. 600, no. 7889, pp. 547–552, Dec. 2021, doi: 10.1038/s41586-021-04184-w.

[145] J. L. Watson *et al.*, “De novo design of protein structure and function with RFdiffusion.,” *Nature*, vol. 620, no. 7976, pp. 1089–1100, Jul. 2023, doi: 10.1038/S41586-023-06415-8.

[146] D. V. Laurents, “AlphaFold 2 and NMR Spectroscopy: Partners to Understand Protein Structure, Dynamics and Function,” *Front Mol Biosci*, vol. 9, p. 906437, May 2022, doi: 10.3389/FMOLB.2022.906437/BIBTEX.

[147] S. K. Niazi, Z. Mariam, and R. Z. Paracha, “Limitations of Protein Structure Prediction Algorithms in Therapeutic Protein Development,” 2024, doi: 10.3390/biomedinformatics4010007.

[148] J. Wang, J. L. Watson, and S. L. Lisanza, “Protein Design Using Structure-Prediction Networks: AlphaFold and RoseTTAFold as Protein Structure Foundation Models,” *Cold Spring Harb Perspect Biol*, vol. 16, no. 7, Jul. 2024, doi: 10.1101/CSHPERSPECT.A041472.

[149] R. F. Alford *et al.*, “The Rosetta All-Atom Energy Function for Macromolecular Modeling and Design,” *J Chem Theory Comput*, vol. 13, no. 6, pp. 3031–3048, Jun. 2017, doi: 10.1021/ACS.JCTC.7B00125.

[150] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, “Equation of State Calculations by Fast Computing Machines,” *J Chem Phys*, vol. 21, no. 6, pp. 1087–1092, Jun. 1953, doi: 10.1063/1.1699114.

[151] S. Reardon, “Five protein-design questions that still challenge AI,” *Nature*, vol. 635, no. 8037, pp. 246–248, Nov. 2024, doi: 10.1038/D41586-024-03595-9.

[152] T. Oliwa and Y. Shen, “cNMA: a framework of encounter complex-based normal mode analysis to model conformational changes in protein interactions,” *Bioinformatics*, vol. 31, no. 12, pp. i151–i160, Jun. 2015, doi: 10.1093/BIOINFORMATICS/BTV252.

[153] J. R. Allison, “Computational methods for exploring protein conformations,” *Biochem Soc Trans*, vol. 48, no. 4, pp. 1707–1724, Aug. 2020, doi: 10.1042/BST20200193.

[154] A. B. Guo, D. Akpinaroglu, M. J. S. Kelly, and T. Kortemme, “Deep learning guided design of dynamic proteins,” *bioRxiv*, Jul. 2024, doi: 10.1101/2024.07.17.603962.

[155] K. Fujisawa, “Regulation of Adenine Nucleotide Metabolism by Adenylate Kinase Isozymes: Physiological Roles and Diseases,” 2023, doi: 10.3390/ijms24065561.

[156] J. A. Brom, S. Samsri, R. G. Petrikis, S. Parnham, and G. J. Pielak, “1H, 13C, 15N backbone resonance assignment of Escherichia coli adenylate kinase,” *Biomol NMR Assign*, vol. 17, no. 2, pp. 235–238, Dec. 2023, doi: 10.1007/S12104-023-10147-1/METRICS.

[157] P. Dzeja and A. Terzic, “Adenylate Kinase and AMP Signaling Networks: Metabolic Monitoring, Signal Communication and Body Energy Sensing,” *Int J Mol Sci*, vol. 10, no. 4, p. 1729, Apr. 2009, doi: 10.3390/IJMS10041729.

[158] J. Kim, S. Moon, T. D. Romo, Y. Yang, E. Bae, and G. N. Phillips, “Conformational dynamics of adenylate kinase in crystals,” *Structural Dynamics*, vol. 11, no. 1, Jan. 2024, doi: 10.1063/4.0000205/3266828.

[159] V. V. H. Giri Rao and S. Gosavi, “In the Multi-domain Protein Adenylate Kinase, Domain Insertion Facilitates Cooperative Folding while Accommodating Function at Domain Interfaces,” *PLoS Comput Biol*, vol. 10, no. 11, p. e1003938, Nov. 2014, doi: 10.1371/JOURNAL.PCBI.1003938.

[160] Schrödinger LLC, “The PyMOL Molecular Graphics System, Version~1.8,” Nov. 2015.

[161] P. C. Whitford, O. Miyashita, Y. Levy, and J. N. Onuchic, “Conformational transitions of Adenylate Kinase: switching by cracking,” *J Mol Biol*, vol. 366, no. 5, p. 1661, Mar. 2007, doi: 10.1016/J.JMB.2006.11.085.

[162] U. Olsson and M. Wolf-Watz, “Overlap between folding and functional energy landscapes for adenylate kinase conformational change,” *Nature Communications 2010 1:1*, vol. 1, no. 1, pp. 1–8, Nov. 2010, doi: 10.1038/ncomms1106.

[163] H. D. Song and F. Zhu, “Conformational Dynamics of a Ligand-Free Adenylate Kinase,” *PLoS One*, vol. 8, no. 7, p. e68023, Jul. 2013, doi: 10.1371/JOURNAL.PONE.0068023.

[164] E. Formoso, V. Limongelli, and M. Parrinello, “Energetics and Structural Characterization of the large-scale Functional Motion of Adenylate Kinase,” *Scientific Reports 2015 5:1*, vol. 5, no. 1, pp. 1–8, Feb. 2015, doi: 10.1038/srep08425.

[165] H. Song, Y. Wutthinitikornkit, X. Zhou, and J. Li, “Impacts of mutations on dynamic allostery of adenylate kinase,” *J Chem Phys*, vol. 155, no. 3, Jul. 2021, doi: 10.1063/5.0053715.

[166] Z. lu Li, C. Mattos, and M. Buck, “Computational studies of the principle of dynamic-change-driven protein interactions,” *Structure*, vol. 30, no. 6, pp. 909-916.e2, Jun. 2022, doi: 10.1016/J.STR.2022.03.008.

[167] D. A. Case *et al.*, “The Amber Biomolecular Simulation Programs,” *J Comput Chem*, vol. 26, no. 16, p. 1668, Dec. 2005, doi: 10.1002/JCC.20290.

[168] H. Berman, K. Henrick, and H. Nakamura, “Announcing the worldwide Protein Data Bank,” *Nat Struct Biol*, vol. 10, no. 12, p. 980, Dec. 2003, doi: 10.1038/NSB1203-980.

[169] R. J. Loncharich, B. R. Brooks, and R. W. Pastor, “Langevin dynamics of peptides: the frictional dependence of isomerization rates of N-acetylalanyl-N’-methylamide,” *Biopolymers*, vol. 32, no. 5, pp. 523–535, 1992, doi: 10.1002/BIP.360320508.

[170] D. R. Koes and J. K. Vries, “Evaluating Amber force fields using computed NMR chemical shifts”, doi: 10.1002/prot.25350.

[171] P. Robustelli, S. Piana, and D. E. Shaw, “Developing a molecular dynamics force field for both folded and disordered protein states,” *Proc Natl Acad Sci U S A*, vol. 115, no. 21, pp. E4758–E4766, May 2018, doi: 10.1073/PNAS.1800690115/-/DCSUPPLEMENTAL.

[172] H. Lutz, V. Jaeger, T. Weidner, and B. L. De Groot, “Interpretation of Interfacial Protein Spectra with Enhanced Molecular Simulation Ensembles,” 2018, doi: 10.1021/acs.jctc.8b00840.

[173] “JADE HPC UK.” Accessed: Dec. 29, 2024. [Online]. Available: https://www.jade.ac.uk/

[174] “ARCHER2 Hardware & Software.” Accessed: Dec. 29, 2024. [Online]. Available: https://www.archer2.ac.uk/about/hardware.html?utm_source=chatgpt.com

[175] W. Humphrey, A. Dalke, and K. Schulten, “VMD: Visual molecular dynamics,” *J Mol Graph*, vol. 14, no. 1, pp. 33–38, Feb. 1996, doi: 10.1016/0263-7855(96)00018-5.

[176] “R: The R Project for Statistical Computing.” Accessed: Jan. 18, 2025. [Online]. Available: https://www.r-project.org/

[177] M. E. Irrgang, J. M. Hays, and P. M. Kasson, “gmxapi: a high-level interface for advanced control and extension of molecular dynamics simulations,” *Bioinformatics*, vol. 34, no. 22, pp. 3945–3947, Nov. 2018, doi: 10.1093/BIOINFORMATICS/BTY484.

[178] MerkelDirk, “Docker,” *Linux Journal*, Mar. 2014, doi: 10.5555/2600239.2600241.

[179] “Poetry - Python dependency management and packaging made easy.” Accessed: Jan. 18, 2025. [Online]. Available: https://python-poetry.org/

[180] S. Kullback and R. A. Leibler, “On Information and Sufficiency,” *The Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79–86, Mar. 1951, doi: 10.1214/aoms/1177729694.

[181] “Visual Studio Code - Code Editing. Redefined.” Accessed: Jan. 26, 2025. [Online]. Available: https://code.visualstudio.com/

[182] W. Evangelista Falcon, S. R. Ellingson, J. C. Smith, and J. Baudry, “Ensemble Docking in Drug Discovery: How Many Protein Configurations from Molecular Dynamics Simulations are Needed To Reproduce Known Ligand Binding?,” *J Phys Chem B*, vol. 123, no. 25, pp. 5189–5195, Jun. 2019, doi: 10.1021/ACS.JPCB.8B11491.

Appendix I

This research was reviewed by the College of Engineering, Design and Physical Sciences Research Ethics Committee, which confirmed that ethical review was not required. The confirmation letter is included in this appendix.

College of Engineering, Design and Physical Sciences Research Ethics Committee
Brunel University London
Kingston Lane
Uxbridge
UB8 3PH
United Kingdom

www.brunel.ac.uk

21 January 2022

LETTER OF CONFIRMATION

Applicant: Miss Namir Oues

Project Title: Development of machine learning methods for automated design of new biological functions in bacterial proteins

Reference: 35179-NER-Jan/2022- 37432-1

Dear Miss Namir Oues

The Research Ethics Committee has considered the above application recently submitted by you.

The Chair, acting under delegated authority has confirmed that, according to the information provided in your application, your project does not require ethical review.

Please note that:

- **You are not permitted to conduct research involving human participants, their tissue and/or their data. If you wish to conduct such research, you must contact the Research Ethics Committee to seek approval prior to engaging with any participants or working with data for which you do not have approval.**
- The Research Ethics Committee reserves the right to sample and review documentation relevant to the study.
- If during the course of the study, you would like to carry out research activities that concern a human participant, their tissue and/or their data, you must inform the Committee by submitting an appropriate Research Ethics Application. Research activity includes the recruitment of participants, undertaking consent procedures and collection of data. Breach of this requirement constitutes research misconduct and is a disciplinary offence.

Good luck with your research!

Kind regards,

Professor Simon Taylor

Chair of the College of Engineering, Design and Physical Sciences Research Ethics Committee

Brunel University London

Appendix II

In this section the three submission scripts used on ARCHER2 to run MDAutoMut library are provided. All three workflows (`mutation_workflow.py`, `mdprep_workflow.py` and `mdautomut_workflow.py`) were tested and validated.

`mutation_workflow.py`

```
#!/bin/bash --login
#SBATCH --job-name=python_test
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1
#SBATCH --time=24:00:00
#SBATCH --account=e280-Pandini
#SBATCH --partition=standard
#SBATCH --qos=standard

# Load the Python module, ...
module load cray-python

# PYTHONPATH
WORK=/mnt/lustre/a2fs-nvme/work/e280/e280/$USER
MDAM=MDAutoMut
MDSS=MDSubSampler
export
PYTHONPATH=$WORK/$MDAM:$WORK/$MDAM/mdam:$WORK/$MDSS:$WORK/$MDSS/mdss:$PYT
HONPATH
echo $PYTHONPATH

# activate virtual environment
source /mnt/lustre/a2fs-
nvme/work/e280/e280/$USER/pyenvs/mddev/bin/activate

# Run your Python program
# prefix="$1"

python mutation_workflow.py \
    --traj="/mnt/lustre/a2fs-
nvme/work/e280/e280/namir_oues/MDAutoMut/data/MD_4AKEA_protein.xtc" \
    --top="/mnt/lustre/a2fs-
nvme/work/e280/e280/namir_oues/MDAutoMut/data/MD_4AKEA_protein.gro" \
    --pdb="/mnt/lustre/a2fs-
nvme/work/e280/e280/namir_oues/MDAutoMut/data/MD_4AKEA_protein.pdb" \
    --frame-number="1" \
    --mutation-file="/mnt/lustre/a2fs-
nvme/work/e280/e280/namir_oues/MDAutoMut/data/mutations_135.rtf" \
    --output-folder="/mnt/lustre/a2fs-
nvme/work/e280/e280/namir_oues/MDAutoMut/mutation_results/" \
```

```
    --output-mode="single" \
    --prefix="1001"
```

mdprep_workflow.py

```
#!/bin/bash --login
#SBATCH --job-name=mdprep_workflow
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=128
#SBATCH --cpus-per-task=1
#SBATCH --time=24:00:00
#SBATCH --account=e280-Pandini
#SBATCH --partition=standard
#SBATCH --qos=standard

# Load the Python module, ...
module load cray-python
module load gromacs

# PYTHONPATH
PREFIX="1001"
RESULTS_PREFIX="${PREFIX}_${SLURM_JOB_ID}"
WORK=/mnt/lustre/a2fs-nvme/work/e280/e280/$USER
MDAM=MDAutoMut
MDSS=MDSubSampler
OUTPUT_DIRECTORY=$WORK/MDAutoMut/system_prep_results
SYSTEM_NAME=4AKE
MUTATION_PATH=$WORK/MDAutoMut/mutation_results
MDP_PATH=$WORK/MDAutoMut/mdp_spc_files

export
PYTHONPATH=$WORK/$MDAM:$WORK/$MDAM/mdam:$WORK/$MDSS:$WORK/$MDSS/mdss:$PYT
HONPATH
export GMXLIB=/mnt/lustre/a2fs-
nvme/work/e280/e280/namir_oues/Programs/gromacs-2022
export MPLCONFIGDIR=$WORK/config/matplotlib

echo $PYTHONPATH
echo $GMXLIB
echo $MPLCONFIGDIR

# Capture the Slurm Job ID
JOB_ID=$SLURM_JOB_ID
echo "Job ID: $JOB_ID"

# activate virtual environment
source /mnt/lustre/a2fs-
nvme/work/e280/e280/namir_oues/pyenv/mddev/bin/activate

# Run your Python program
python mdprep_workflow.py \
```

```
    --prefix="$RESULTS_PREFIX" \
    --system-name="$SYSTEM_NAME" \
    --mutation="V135K" \
    --outpath-directory="$OUTPUT_DIRECTORY" \
    --pdb="$MUTATION_PATH/1001_V142K.pdb" \
    --spc216-gro="$MDP_PATH/spc216.gro" \
    --ions-mdp="$MDP_PATH/ions.mdp" \
    --em-1-mdp="$MDP_PATH/em_sd_posre_1.mdp" \
    --em-2-mdp="$MDP_PATH/em_sd_2.mdp" \
    --em-3-mdp="$MDP_PATH/em_cg_3.mdp" \
    --nvt-1-mdp="$MDP_PATH/eqv_posre2000_T200_1.mdp" \
    --nvt-2-mdp="$MDP_PATH/eqv_posre1000_T250_2.mdp" \
    --nvt-3-mdp="$MDP_PATH/eqv_posre1000_T300_3.mdp" \
    --nvt-4-mdp="$MDP_PATH/eqv_posre500_T300_4.mdp" \
    --nvt-5-mdp="$MDP_PATH/eqv_posre250_T300_5.mdp" \
    --nvt-6-mdp="$MDP_PATH/eqv_T300_6.mdp" \
    --npt-1-mdp="$MDP_PATH/eqp_T300_1.mdp" \
    --npt-2-mdp="$MDP_PATH/eqpadv_T300_2.mdp" \
    --prod-mdp="$MDP_PATH/prod_T300.mdp"
```

mdautomut_workflow.py

```
#!/bin/bash --login
#SBATCH --job-name=mdautomut_workflow
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=128
#SBATCH --cpus-per-task=1
#SBATCH --time=48:00:00
#SBATCH --account=e280-Pandini
#SBATCH --partition=standard
#SBATCH --qos=long

module load cray-python
module load gromacs

# Environment setup
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
export MPICH_MAX_THREAD_SAFETY=multiple

export GMX_MAXBACKUP=-1
export GMX_ALLOW_CPT=1

# PYTHONPATH
RESULTS_PREFIX="0801"
PREFIX_SLURM="${RESULTS_PREFIX}_${SLURM_JOB_ID}"
WORK=/mnt/lustre/a2fs-nvme/work/e280/e280/namir_oues
MDAM=MDAutoMut
MDSS=MDSubSampler
OUTPUT_DIRECTORY=$WORK/$MDAM/mdautomut_results
RESULTS_SUBFOLDER_SLURM="$WORK/$MDAM/mdautomut_workflow_results/$PREFIX_S
LURM"
RESULTS_SUBFOLDER="$WORK/$MDAM/mdautomut_workflow_results"
```

```
MDP_PATH=$WORK/$MDAM/mdp_spc_files
MUTATION_FILE_PATH=$WORK/$MDAM/data/mutations_135_142.rtf

mkdir -p "$RESULTS_SUBFOLDER_SLURM"
mkdir -p "$RESULTS_SUBFOLDER"

export
PYTHONPATH=$WORK/$MDAM:$WORK/$MDAM/mdam:$WORK/$MDSS:$WORK/$MDSS/mdss:$PYT
HONPATH
export GMXLIB=/mnt/lustre/a2fs-
nvme/work/e280/e280/namir_oues/Programs/gromacs-2022
export MPLCONFIGDIR=$WORK/config/matplotlib

echo $PYTHONPATH
echo $GMXLIB
echo $MPLCONFIGDIR

# activate virtual environment
source /mnt/lustre/a2fs-
nvme/work/e280/e280/namir_oues/pyenv/mddev/bin/activate

# Capture the Slurm Job ID
JOB_ID=$SLURM_JOB_ID
echo "Job ID: $JOB_ID"

# Run your Python program
python mdautomut_workflow.py \
--prefix="0801" \
--check-existing \
--output-mode="multiple" \
--system-name="4AKE" \
--mutation-file="$MUTATION_FILE_PATH" \
--pdb-file="$RESULTS_SUBFOLDER/WT/0801_4AKE_WT_prod.pdb" \
--trajectory-file="$RESULTS_SUBFOLDER/WT/0801_4AKE_WT_prod.xtc" \
--topology-file="$RESULTS_SUBFOLDER/WT/0801_4AKE_WT_prod.gro" \
--outpath-dir="$RESULTS_SUBFOLDER" \
--mutation="WT" \
--mutation-subfolder="WT" \
--dissimilarity-threshold="0.05" \
--frame-number="1" \
--spc216-gro="$MDP_PATH/spc216.gro" \
--ions-mdp="$MDP_PATH/ions.mdp" \
--em1-mdp="$MDP_PATH/em_sd_posre_1.mdp" \
--em2-mdp="$MDP_PATH/em_sd_2.mdp" \
--em3-mdp="$MDP_PATH/em_cg_3.mdp" \
--nvt1-mdp="$MDP_PATH/eqv_posre2000_T200_1.mdp" \
--nvt2-mdp="$MDP_PATH/eqv_posre1000_T250_2.mdp" \
--nvt3-mdp="$MDP_PATH/eqv_posre1000_T300_3.mdp" \
--nvt4-mdp="$MDP_PATH/eqv_posre500_T300_4.mdp" \
--nvt5-mdp="$MDP_PATH/eqv_posre250_T300_5.mdp" \
--nvt6-mdp="$MDP_PATH/eqv_T300_6.mdp" \
```

```
--npt1-mdp="$MDP_PATH/eqp_T300_1.mdp" \
--npt2-mdp="$MDP_PATH/eqpadv_T300_2.mdp" \
--prod-mdp="$MDP_PATH/prod_T300.mdp"
```

Appendix III

In this section the submission script used to generate data on ARCHER2 is provided below. Due to time limitation of 24h on HPC queues a restart script was required to restart the simulation multiple times.

archer_prod.sh

```
#!/bin/bash
#SBATCH --mail-user=namir.oues@brunel.ac.uk
#SBATCH --mail-type=ALL
#SBATCH --job-name=mdrun_test
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=128
#SBATCH --cpus-per-task=1
#SBATCH --time=24:00:00
#SBATCH --account=e280-Pandini
#SBATCH --partition=standard
#SBATCH --qos=standard

label=`basename $PWD`
mol=$label

runtyp="prod_T300"
job=$mol"_"$runtyp

# Setup the environment
module load gromacs

# Ensure the cpus-per-task option is propagated to srun commands
export SRUN_CPUS_PER_TASK=$SLURM_CPUS_PER_TASK

export OMP_NUM_THREADS=1
srun --distribution=block:block --hint=nomultithread gmx_mpi mdrun -s
$job.tpr -o $job.trr -g $job.log -e $job.edr -c $job.gro -cpo $job.cpt -x
$job.xtc >& mdrun_$job.log
```

archer_prod_restart.sh

```
#!/bin/bash
#SBATCH --mail-user=namir.oues@brunel.ac.uk
#SBATCH --mail-type=ALL
#SBATCH --job-name=mdrun_test
#SBATCH --nodes=1
```

```
#SBATCH --ntasks-per-node=128
#SBATCH --cpus-per-task=1
#SBATCH --time=24:00:00
#SBATCH --account=e280-Pandini
#SBATCH --partition=standard
#SBATCH --qos=standard

label=`basename $PWD`
mol=$label

runtyp="prod_T300"
job=$mol"_"$runtyp
job_restart="restart_"$mol"_"$runtyp

# Setup the environment
module load gromacs

# Ensure the cpus-per-task option is propagated to srun commands
export SRUN_CPUS_PER_TASK=$SLURM_CPUS_PER_TASK

export OMP_NUM_THREADS=1
srun --distribution=block:block --hint=nomultithread gmx_mpi mdrun -s
$job.tpr -o $job.trr -g $job.log -e $job.edr -c $job.gro -cpi $job.cpt -
cpo $job.cpt -x $job.xtc -noappend -nice 0 >& mdrun_$job_restart.log
```

Appendix IV

In this section the python code for the MDAutoPredict library is presented. The code was built in a modular way; hence all classes/modules are presented in separate blocks.

`learner.py`

```
from log_setup import log
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
from methods import machine_learning_models
import joblib
import numpy as np

class MDTrajLearner:
    """
    Class for managing machine learning tasks on protein trajectory data
using ProteinData.

    Attributes
    ----------
    protein_data : ProteinData
        An instance of the ProteinData class.
    performance : dict
```

```
        A dictionary to store the performance metrics of trained models.
    output_path : str
        Path where the trained model object is saved.
    """

    def __init__(self, protein_data):
        """
        Initialize the MDTrajLearner class.

        Parameters
        ----------
        protein_data : ProteinData
            An instance of the ProteinData class.
        """
        self.protein_data = protein_data
        self.performance = {}
        self.output_path = None

    def generate_and_save_matrix(
        self, xtc_output_path, matrix_output_path, unit="nanometer"
    ):
        """
        Generate a matrix representation for learning and save it.

        Parameters
        ----------
        xtc_output_path : str
            Path to save the intermediate trajectory in numpy format.
        matrix_output_path : str
            Path to save the final matrix representation.
        unit : str, optional
            Unit for coordinate conversion. Default is 'nanometer'.

        Returns
        -------
        numpy.ndarray
            The generated matrix representation.
        """
        trajectory_numpy = self.protein_data.cast_output_traj_to_numpy(
            outfilepath=xtc_output_path,
            subsampled_traj=self.protein_data.trajectory_data.trajectory,
            unit=unit,
        )
        matrix_representation = self.protein_data.convert_numpy_to_2D(
            trajectory_numpy, outfilepath=matrix_output_path
        )
        np.save(matrix_output_path, matrix_representation)
        log.info(
            "{:<15s} {:<80s}".format(
                "STEPS",
                f"The matrix representation of the trajectory is saved at
{matrix_output_path}",
            )
        )
        return matrix_representation

    def train_and_test(
        self,
        ml_input_path,
```

```
        target_path,
        model_name,
        model_params,
        output_model_path,
    ):
        """
        Train and test a machine learning model on the provided data.

        Parameters
        ----------
        ml_input_path : str
            Path to the numpy file containing the ML input data.
        target_path : str
            Path to the file containing target variable data.
        model_name : str
            Name of the machine learning method.
        model_params : dict
            Parameters for initializing the machine learning model.
        output_model_path : str
            Path to save the trained model as a binary file.

        Returns
        -------
        dict
            Performance metrics of the trained model.
        """
        # Step 1: Load the input matrix and target labels
        x = np.load(ml_input_path)
        target = np.genfromtxt(target_path, dtype=str, delimiter=",",
skip_header=0)

        log.info(
            "{:<15s} {:<80s}".format(
                "VALIDATION",
                f"Input matrix size: {x.shape}, Target size:
{len(target)}",
            )
        )

        # Step 2: Encode string labels into numerical values
        label_encoder = LabelEncoder()
        y = label_encoder.fit_transform(target)

        # Ensure alignment between input and target sizes
        if x.shape[0] != len(y):
            log.error(
                "{:<15s} {:<80s}".format(
                    "ERROR",
                    "Mismatch between input matrix and target sizes.",
                )
            )
            raise ValueError("Mismatch between input matrix and target
sizes.")

        # Step 3: Split data into training and testing sets
        x_train, x_test, y_train, y_test = train_test_split(
            x, y, test_size=0.3, random_state=25
        )
        log.info(
```

```
            "{:<15s} {:<80s}".format(
                "STEPS",
                "Splitting the data into training and testing is done..",
            )
        )
        log.info(
            "{:<15s} {:<80s}".format(
                "VALIDATION",
                f"Training set size: {x_train.shape}, Testing set size:
{x_test.shape}",
            )
        )

        # Step 4: Initialize and train the model
        log.info(
            "{:<15s} {:<80s}".format(
                "STEPS",
                "Training the model has started.",
            )
        )
        ModelClass = machine_learning_models[model_name]
        model = ModelClass(**model_params)
        model.fit(x_train, y_train)
        log.info(
            "{:<15s} {:<80s}".format(
                "RESULT",
                "Training the model is completed.",
            )
        )

        # Save the trained model
        joblib.dump(model, output_model_path)
        self.output_path = output_model_path
        log.info(
            "{:<15s} {:<80s}".format(
                "STEPS",
                f"The model is saved at {output_model_path}",
            )
        )
        # Step 5: Evaluate the model
        log.info(
            "{:<15s} {:<80s}".format(
                "STEPS",
                "The model evaluation has started.",
            )
        )
        y_pred = model.predict(x_test)
        accuracy = accuracy_score(y_test, y_pred)
        conf_mat = confusion_matrix(y_test, y_pred)
        report = classification_report(
            y_test, y_pred, target_names=label_encoder.classes_,
output_dict=True
        )
        log.info(
            "{:<15s} {:<80s}".format(
                "STEPS",
                "The model evaluation is completed.",
            )
        )
```

```
        log.info(
            "{:<15s} {:<80s}".format(
                "RESULT",
                f"Accuracy of the model: {accuracy:.4f}",
            )
        )
        log.info(
            "{:<15s} {:<80s}".format(
                "RESULT",
                "Confusion Matrix:",
            )
        )
        for i, row in enumerate(conf_mat):
            log.info(
                "{:<15s} {:<80s}".format(
                    f"ROW {i}",
                    f"{row.tolist()}",
                )
            )

        # Store performance metrics
        self.performance = {
            "accuracy": accuracy,
            "confusion_matrix": conf_mat.tolist(),
            "classification_report": report,
        }
        return self.performance

    def test_model(self, ml_input_path, target_path, model_path):
        """
        Test a pre-trained machine learning model and compute performance
metrics.

        Parameters:
            ml_input_path (str): Path to the numpy file containing the
test input data.
            target_path (str): Path to the file containing target
variable data.
            model_path (str): Path to the pre-trained model file.

        Returns:
            dict: Performance metrics of the tested model, including
y_true and y_pred.
        """
        # Step 1: Load test data and pre-trained model
        x_test = np.load(ml_input_path)
        target = np.genfromtxt(target_path, dtype=str, delimiter=",",
skip_header=0)
        model = joblib.load(model_path)

        # Encode target labels if necessary
        label_encoder = LabelEncoder()
        y_true = label_encoder.fit_transform(target)  # Ensure numerical
encoding
        y_pred = model.predict(x_test)  # Predictions should match the
encoding

        try:
            y_scores = (
```

```
                model.predict_proba(x_test)[:, 1]
                if hasattr(model, "predict_proba")
                else None
            )
        except AttributeError:
            y_scores = None

        # Calculate performance metrics
        accuracy = accuracy_score(y_true, y_pred)
        conf_matrix = confusion_matrix(y_true, y_pred)
        class_report = classification_report(y_true, y_pred,
zero_division=0)

        # Decode labels back to original for human readability if needed
        decoded_y_true = label_encoder.inverse_transform(y_true)
        decoded_y_pred = label_encoder.inverse_transform(y_pred)

        return {
            "y_true": decoded_y_true,
            "y_pred": decoded_y_pred,
            "y_scores": y_scores,
            "accuracy": accuracy,
            "confusion_matrix": conf_matrix,
            "classification_report": class_report,
        }
```

evaluator.py

```
import os
import pandas as pd
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
from sklearn.preprocessing import label_binarize
from plot import PerformancePlotting as p

class Evaluator:
    def __init__(self):
        self.accuracy_scores = {}
        self.classification_reports = {}

    def evaluate(self, model_name, y_test, y_pred, class_labels,
output_dir):
        """
        Evaluate model performance and generate confusion matrix plot.
        """
        # Calculate accuracy
        accuracy = accuracy_score(y_test, y_pred)
        print(f"{model_name} Accuracy: {accuracy:.4f}")

        # Generate classification report and confusion matrix
        report = classification_report(
            y_test, y_pred, target_names=class_labels, output_dict=True
        )
        cm = confusion_matrix(y_test, y_pred)

        # Ensure the output directory exists
        os.makedirs(output_dir, exist_ok=True)

        # Save classification report to file
```

```
        report_path = os.path.join(
            output_dir, f"{model_name}_classification_report.txt"
        )
        with open(report_path, "w") as f:
            f.write(classification_report(y_test, y_pred,
target_names=class_labels))
        print(f"Classification report saved at: {report_path}")

        # Plot and save the confusion matrix in the output directory
        p.plot_confusion_matrix(
            cm,
            class_labels,
            model_name,
            f"Confusion matrix: {model_name}",
            accuracy,
            output_dir,
        )
        # Save confusion matrix as a CSV file
        self.save_confusion_matrix_to_csv(cm, class_labels, output_dir,
model_name)

    def save_confusion_matrix_to_csv(self, cm, class_labels, output_dir,
model_name):
        """
        Save confusion matrix to a CSV file.
        """
        os.makedirs(output_dir, exist_ok=True)
        cm_df = pd.DataFrame(cm, index=class_labels,
columns=class_labels)
        output_path = os.path.join(output_dir,
f"{model_name}_confusion_matrix.csv")
        cm_df.to_csv(output_path)
        print(f"Confusion matrix data saved at: {output_path}")

    def save_results(self):
        """
        Save evaluation results.
        """
        os.makedirs("results", exist_ok=True)
        accuracy_df = pd.DataFrame(
            list(self.accuracy_scores.items()), columns=["Model",
"Accuracy"]
        )
        accuracy_df.to_csv(os.path.join("results",
"accuracy_scores.csv"), index=False)
        print("Accuracy scores saved to 'results/accuracy_scores.csv'")

        classification_reports_df = pd.concat(
            {
                k: pd.DataFrame(v).transpose()
                for k, v in self.classification_reports.items()
            },
            axis=0,
        )
        classification_reports_df.to_csv(
            os.path.join("results", "classification_reports.csv")
        )
        print("Classification reports saved to
'results/classification_reports.csv'")
```

```
    def load_labels(self, file_path, label_mapping=None):
        """
        Load labels from a file. Optionally map numeric labels to string
labels.

        Parameters:
            file_path (str): Path to the file containing labels.
            label_mapping (dict, optional): Mapping of numeric labels to
string labels.

        Returns:
            list: Loaded and optionally mapped labels.
        """
        try:
            # Load labels as a list
            with open(file_path, "r") as f:
                labels = [line.strip() for line in f]

            if not labels:
                raise ValueError(f"The file {file_path} is empty.")

            # If mapping is provided, map numeric labels to string labels
            if label_mapping:
                labels = [
                    label_mapping[label] for label in labels if label in
label_mapping
                ]

            return labels
        except Exception as e:
            raise ValueError(f"Error reading or processing {file_path}:
{e}")
```

property.py

```
from log_setup import log
from mdss.property import ProteinProperty

class MLProperty(ProteinProperty):
    """
    MLProperty class for calculating and labeling properties using
machine learning predictions.

    Attributes
    ----------
    model : object
        A trained machine learning model for making predictions.
    frame_labels : dict
        A dictionary mapping frame indices to predicted labels.
    predictions : list
        A list of predictions for all frames.
    """

    def __init__(self, protein_data, model, atom_selection="name CA"):
        """
        Initialize MLProperty with a trained machine learning model.
```

```
        Parameters
        ----------
        protein_data : ProteinData
            An instance of the ProteinData class.
        model : object
            A trained machine learning model capable of making
predictions.
        atom_selection : str, optional
            Atom selection for property calculation. Default is 'name
CA'.
        """
        super().__init__(protein_data, atom_selection)
        self.model = model
        self.frame_labels = {}
        self.predictions = []

    def calculate_property(self, input_matrix=None):
        """
        Use the ML model to calculate a property for all frames.

        Parameters
        ----------
        input_matrix : numpy.ndarray, optional
            Input data matrix where each row corresponds to a frame.
            If None, input_matrix is generated from protein_data.

        Returns
        -------
        list
            A list of predictions corresponding to each frame.
        """
        if self.model is None:
            log.error(
                "{:<15s} {:<80s}".format(
                    "ERROR",
                    "No model is loaded for predictions. Please use a
valid model.",
                )
            )
            raise ValueError(
                "No model is loaded for predictions. Please use model as
input."
            )

        # Make predictions
        self.predictions = self.model.predict(input_matrix)
        self.property_vector = self.predictions
        self.frame_indices = list(range(len(self.predictions)))
        self._property_statistics()
        return self.predictions

    def label_frames(self, labels=None):
        """
        Assign labels to frames based on the model predictions.

        Parameters
        ----------
        labels : list, optional
```

```
            List of labels to assign based on the predictions.
            If None, use the unique predictions as labels.

        Returns
        -------
        dict
            A dictionary mapping frame indices to their assigned labels.
        """
        if not self.predictions:
            log.error(
                "{:<15s} {:<80s}".format(
                    "ERROR",
                    "No predictions available. Run calculate_property()
first.",
                )
            )
            raise ValueError(
                "No predictions available. Run calculate_property()
first."
            )

        unique_predictions = sorted(set(self.predictions))
        if labels is None:
            labels = unique_predictions
        else:
            if len(labels) != len(unique_predictions):
                log.error(
                    "{:<15s} {:<80s}".format(
                        "ERROR",
                        "Number of labels does not match the unique
predictions.",
                    )
                )
                raise ValueError(
                    "The number of labels does not match the number of
unique predictions."
                )

        label_map = {pred: label for pred, label in
zip(unique_predictions, labels)}

        self.frame_labels = {
            idx: label_map[pred]
            for idx, pred in zip(self.frame_indices, self.predictions)
        }
        log.info(
            "{:<15s} {:<80s}".format(
                "RESULT",
                "Frame labels assigned based on predictions.",
            )
        )
        return self.frame_labels

    def write_frame_labels(self, outfilepath):
        """
        Write the frame labels to a file.

        Parameters
        ----------
```

```
        outfilepath : str
            Path to save the frame labels.
        """
        if not self.frame_labels:
            log.error(
                "{:<15s} {:<80s}".format(
                    "ERROR",
                    "No frame labels available. Run label_frames()
first.",
                )
            )
            raise ValueError("No frame labels available. Run
label_frames() first.")

        with open(outfilepath, "w") as f:
            for frame_idx, label in self.frame_labels.items():
                f.write(f"{frame_idx} {label}\n")
        log.info(
            "{:<15s} {:<80s}".format(
                "RESULT",
                f"Frame labels written to {outfilepath}",
            )
        )
```

plot.py

```
import matplotlib.pyplot as plt
import seaborn as sns
import os
import numpy as np
from sklearn.preprocessing import label_binarize
from sklearn.metrics import PrecisionRecallDisplay
from sklearn.preprocessing import label_binarize
from sklearn.calibration import calibration_curve
from sklearn.metrics import PrecisionRecallDisplay
from sklearn.metrics import (
    confusion_matrix,
    precision_recall_curve,
    roc_curve,
)
from log_setup import log

class PerformancePlotting:
    @staticmethod
    def plot_confusion_matrix(
        cm, class_labels, model_name, title, accuracy, output_dir
    ):
        """
        Plot and save the confusion matrix with counts and percentages in
all cells.

        Parameters:
            cm (array-like): Confusion matrix.
            class_labels (list): List of class labels.
            title (str): Title of the plot.
            accuracy (float): Model accuracy to display in the title.
            output_dir (str): Directory to save the plot.
        """
```

```
        # Ensure cm dimensions match class_labels
        assert (
            cm.shape[0] == cm.shape[1] == len(class_labels)
        ), f"Confusion matrix dimensions {cm.shape} and class labels
{len(class_labels)} do not match."

        # Calculate total samples for global normalization
        total = cm.sum()

        # Create annotation labels with counts and percentages for all
cells
        labels = np.empty_like(cm, dtype=object)
        for i in range(cm.shape[0]):
            for j in range(cm.shape[1]):
                count = cm[i, j]
                percentage = (count / total) * 100 if total > 0 else 0
                labels[i, j] = f"{count}\n({percentage:.1f}%)"

        # Plot the heatmap
        plt.figure(figsize=(8, 6))
        sns.heatmap(
            cm,
            annot=False,  # Turn off default annotation
            fmt="",
            cmap="GnBu",
            xticklabels=class_labels,
            yticklabels=class_labels,
            cbar=True,
            linewidths=0.5,
            linecolor="gray",
        )

        # Manually add text annotations
        for i in range(cm.shape[0]):
            for j in range(cm.shape[1]):
                count = cm[i, j]
                percentage = (count / cm.sum()) * 100 if cm.sum() > 0
else 0
                text = f"{count}\n({percentage:.1f}%)"
                plt.text(
                    j + 0.5,  # x-coordinate (column)
                    i + 0.5,  # y-coordinate (row)
                    text,
                    ha="center",
                    va="center",
                    color="black",
                )

        # Add title and labels
        plt.title(f"{title}\nAccuracy: {accuracy:.4f}")
        plt.xlabel("Predicted")
        plt.ylabel("Actual")

        # Save the plot
        os.makedirs(output_dir, exist_ok=True)
        output_path = os.path.join(output_dir, f" {model_name}
_confusion_matrix.png")
        plt.savefig(output_path, dpi=300)
        plt.close()
```

```
    @staticmethod
    def plot_precision_recall(y_true, y_scores, model_name, output_dir,
prefix=None):
        """
        Precision-Recall Curve: Handle binary and multiclass
classification.

        Parameters:
            y_true (array): True labels.
            y_scores (array or None): Predicted probabilities or decision
scores.
            model_name (str): Name of the model.
            output_dir (str): Directory to save the plot.
        """
        # Skip plotting if y_scores is None
        if y_scores is None:
            log.info(
                "{:<15s} {:<80s}".format(
                    "STEPS",
                    f"Skipping Precision-Recall Curve for {model_name}:
No probability scores provided..",
                )
            )

            return

        # Binarize labels for multiclass
        classes = np.unique(y_true)
        y_true_binarized = label_binarize(y_true, classes=classes)

        plt.figure(figsize=(8, 6))
        for i, class_name in enumerate(classes):
            if y_scores.ndim > 1:
                y_class_scores = y_scores[:, i]
            else:
                y_class_scores = y_scores

            precision, recall, _ = precision_recall_curve(
                y_true_binarized[:, i], y_class_scores
            )
            display = PrecisionRecallDisplay(precision=precision,
recall=recall)
            display.plot(ax=plt.gca(), name=f"Class {class_name}")

        plt.title(f"Precision-Recall Curve for {model_name}")
        plt.xlabel("Recall")
        plt.ylabel("Precision")
        plt.legend(loc="best")
        file_name = (
            f"{prefix}_{model_name}_precision_recall_curve.png"
            if prefix
            else f"{model_name}_precision_recall_curve.png"
        )
        plt.savefig(os.path.join(output_dir, file_name))
        plt.close()

    @staticmethod
```

```
    def plot_roc_curve(y_true, y_scores, model_name, output_dir,
prefix=None):
        """
        ROC Curve: Handle binary and multiclass classification.

        Parameters:
            y_true (array): True labels.
            y_scores (array or None): Predicted probabilities or decision
scores.
            model_name (str): Name of the model.
            output_dir (str): Directory to save the plot.
        """
        from sklearn.preprocessing import label_binarize
        from sklearn.metrics import RocCurveDisplay

        # Skip plotting if y_scores is None
        if y_scores is None:
            log.info(
                "{:<15s} {:<80s}".format(
                    "STEPS",
                    f"Skipping ROC Curve for {model_name}: No probability
or decision scores provided.",
                )
            )
            return

        # Binarize labels for multiclass
        classes = np.unique(y_true)
        y_true_binarized = label_binarize(y_true, classes=classes)

        plt.figure(figsize=(8, 6))
        for i, class_name in enumerate(classes):
            if y_scores.ndim > 1:
                y_class_scores = y_scores[:, i]
            else:
                y_class_scores = y_scores

            fpr, tpr, _ = roc_curve(y_true_binarized[:, i],
y_class_scores)
            RocCurveDisplay(fpr=fpr, tpr=tpr).plot(
                ax=plt.gca(), name=f"Class {class_name}"
            )

        plt.title(f"ROC Curve for {model_name}")
        plt.xlabel("False Positive Rate")
        plt.ylabel("True Positive Rate")
        plt.legend(loc="best")
        file_name = (
            f"{prefix}_{model_name}_roc_curve.png"
            if prefix
            else f"{model_name}_roc_curve.png"
        )
        plt.savefig(os.path.join(output_dir, file_name))
        plt.close()

    @staticmethod
    def plot_calibration_curve(y_true, y_prob, model_name, output_dir,
prefix=None):
        """
```

```
        Calibration Curve: Handle binary and multiclass classification.

        Parameters:
            y_true (array): True labels.
            y_prob (array): Predicted probabilities for each class.
            model_name (str): Name of the model.
            output_dir (str): Directory to save the plot.
        """
        from sklearn.preprocessing import label_binarize

        # Binarize labels for multiclass
        classes = np.unique(y_true)
        y_true_binarized = label_binarize(y_true, classes=classes)

        plt.figure(figsize=(8, 6))
        for i, class_name in enumerate(classes):
            if y_prob.ndim > 1:
                y_class_prob = y_prob[:, i]
            else:
                y_class_prob = y_prob

            prob_true, prob_pred = calibration_curve(
                y_true_binarized[:, i], y_class_prob, n_bins=10
            )
            plt.plot(prob_pred, prob_true, marker="o", label=f"Class
{class_name}")

        plt.plot(
            [0, 1], [0, 1], linestyle="--", color="gray",
label="Perfectly Calibrated"
        )
        plt.title(f"Calibration Curve for {model_name}")
        plt.xlabel("Predicted Probability")
        plt.ylabel("True Probability")
        plt.legend(loc="best")
        file_name = (
            f"{prefix}_{model_name}_calibration_curve.png"
            if prefix
            else f"{model_name}_calibration_curve.png"
        )
        plt.savefig(os.path.join(output_dir, file_name))
        plt.close()

    @staticmethod
    def generate_model_plots(y_true, y_pred, y_scores, model_name,
output_dir):
        """
        Generate and save all plots for a given model.

        Parameters:
            y_true (array): Ground truth labels.
            y_pred (array): Predicted labels by the model.
            y_scores (array): Predicted probabilities or decision scores.
            model_name (str): Name of the machine learning model.
            output_dir (str): Directory to save the plots.
        """
        os.makedirs(output_dir, exist_ok=True)
        PerformancePlotting.plot_confusion_matrix(
            y_true, y_pred, model_name, output_dir
```

```
        )
        PerformancePlotting.plot_precision_recall(
            y_true, y_scores, model_name, output_dir
        )
        PerformancePlotting.plot_roc_curve(y_true, y_scores, model_name,
output_dir)
        if y_scores is not None and len(np.unique(y_scores)) > 2:
            PerformancePlotting.plot_calibration_curve(
                y_true, y_scores, model_name, output_dir
            )

    @staticmethod
    def plot_model_comparison(models, accuracies, output_dir,
prefix="comparison"):
        """
        Generates and saves a bar plot comparing model accuracies.

        Parameters:
            models (list): List of model names.
            accuracies (list): List of corresponding accuracies for the
models.
            output_dir (str): Directory to save the plot.
            prefix (str): Prefix for the output file name.
        """
        import os

        # Ensure output directory exists
        os.makedirs(output_dir, exist_ok=True)

        # Create and save the plot
        plt.figure(figsize=(10, 6))
        plt.bar(models, accuracies)
        plt.xlabel("Model")
        plt.ylabel("Accuracy")
        plt.title("Model Performance Comparison")
        plt.xticks(rotation=45)

        # Save the plot in the results folder
        comparison_plot_path = os.path.join(
            output_dir, f"{prefix}_models_accuracy_comparison.png"
        )
        plt.savefig(comparison_plot_path)
        plt.close()  # Close the plot to avoid displaying it
        log.info(
            "{:<15s} {:<80s}".format(
                "OUTPUT",
                f"Model performance comparison plot saved to
{comparison_plot_path}",
            )
        )
```

methods.py

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.neural_network import MLPClassifier
```

```
machine_learning_models = {
    "LogisticRegression": LogisticRegression,
    "RandomForest": RandomForestClassifier,
    "SVM": SVC,
    "DecisionTree": DecisionTreeClassifier,
    "GradientBoosting": GradientBoostingClassifier,
    "MLP": MLPClassifier,
}
```

log_setup.py

```
import logging
import os
from datetime import datetime

try:
    if not hasattr(logging, "configured"):
        here = os.path.abspath(os.path.dirname(__file__))
        log_dir = os.path.join(here, "logs")

        if not os.path.exists(log_dir):
            os.makedirs(log_dir)

        filename = datetime.now().strftime("log_%Y_%m_%d_%H_%M_%S.txt")
        filepath = os.path.join(log_dir, filename)
        print(f"Log file path: {filepath}")

        logging.basicConfig(
            filename=filepath,
            level=logging.INFO,
            format="%(asctime)s %(levelname)s %(message)s",
            filemode="w",
        )
        logging.configured = True
except Exception as e:
    print(f"Error during logging setup: {e}")

log = logging.getLogger(__name__)
```

workflow.py

```
from log_setup import log
from mdss.protein_data import ProteinData
from learner import MDTrajLearner
from property import MLProperty
from evaluation import Evaluator
import numpy as np
import joblib
import os

prefix = "1912"

log.info(
```

```
    "{:<15s} {:<80s}".format(
        "STEPS",
        f"Starting the machine learning workflow with prefix '{prefix}'",
    )
)

# Input files
trajectory_path = "data/R02_4AKE_DM_prod_T300_protein_dt40.xtc"
topology_path = "data/R02_4AKE_DM_prod_T300_protein.gro"
target_path = "data/R02_4AKE_DM_prod_T300_dt40_target_var.dat"
trajectory_to_predict = "data/R02_4AKE_DM_prod_T300_protein_dt200.xtc"

# Output files
results_dir = "results"
if not os.path.exists(results_dir):
    os.makedirs(results_dir)

matrix_output_path = os.path.join(results_dir, f"{prefix}_matrix.npy")
trajectory_to_predict_matrix = os.path.join(
    results_dir, f"{prefix}_matrix_to_predict.npy"
)

# Step 1: Initialize ProteinData and MDTrajLearners
protein_data = ProteinData(trajectory_path, topology_path)
learner = MDTrajLearner(protein_data)

# Step 2: Generate and save matrix representation
log.info(
    "{:<15s} {:<80s}".format(
        "STEPS",
        "Generating matrix representation of the trajectory.",
    )
)
learner.generate_and_save_matrix(
    xtc_output_path=trajectory_path,
matrix_output_path=matrix_output_path
)

# Initialize the Evaluator
evaluator = Evaluator()

# Step 3: Train and test all machine learning models
models_to_test = [
    "LogisticRegression",
    "RandomForest",
    "SVM",
    "DecisionTree",
    "GradientBoosting",
    "MLP",
]
results = {}
label_mapping = {"0": "A", "1": "B", "2": "I", "3": "N"}

# Load true labels
y_test = evaluator.load_labels(target_path)

for model_name in models_to_test:
    model_output_dir = os.path.join(results_dir,
f"{prefix}_{model_name}")
```

```
    os.makedirs(model_output_dir, exist_ok=True)
    model_output_path = os.path.join(model_output_dir, "model.joblib")

    if os.path.exists(model_output_path):
        log.info(
            "{:<15s} {:<80s}".format(
                "INPUT",
                f"Trained model for {model_name} already exists at
{model_output_path}. Skipping training.",
            )
        )
        # Load existing model
        loaded_model = joblib.load(model_output_path)
        y_pred = loaded_model.predict(np.load(matrix_output_path))

        # Evaluate and generate confusion matrix
        evaluator.evaluate(
            model_name=model_name,
            y_test=y_test,
            y_pred=y_pred,
            class_labels=list(label_mapping.values()),
            output_dir=model_output_dir,
        )
        continue

    log.info(
        "{:<15s} {:<80s}".format(
            "STEPS",
            f"Training and testing with {model_name}.",
        )
    )

    # Define model-specific parameters
    model_params = {}
    if model_name == "RandomForest":
        model_params = {"n_estimators": 100, "random_state": 42,
"n_jobs": -1}
    elif model_name == "GradientBoosting":
        model_params = {"n_estimators": 100, "learning_rate": 0.1}
    elif model_name == "SVM":
        model_params = {"kernel": "rbf", "C": 1, "gamma": "scale"}
    elif model_name == "MLP":
        model_params = {"hidden_layer_sizes": (100,), "max_iter": 300}

    # Train and test the model
    performance = learner.train_and_test(
        ml_input_path=matrix_output_path,
        target_path=target_path,
        model_name=model_name,
        model_params=model_params,
        output_model_path=model_output_path,
    )

    # Use predicted labels for evaluation
    y_pred = performance["predictions"]  # Assuming 'predictions' holds
y_pred
    evaluator.evaluate(
        model_name=model_name,
        y_test=y_test,
```

```
        y_pred=y_pred,
        class_labels=list(label_mapping.values()),
        output_dir=model_output_dir,
    )

log.info(f"Workflow completed with prefix '{prefix}'")
```

Appendix V

The following R code was used to generate the plots for the figures presented in the thesis.

Kde2d_PV_replicas.R

```
# parameters
nbin = 200
ncol = 200

# input details
simprefix = "R01-R05_4AKE"
prefix = paste(simprefix, "_XX_prod_T300_", sep = '')
APC = 'PC1'
BPC = 'PC2'

# functions
read_proj_xvg <- function(filename){
    filelines <- readLines(filename)
    filelines <- filelines[c(1:(length(filelines)-1))]
    df <- read.table(text = filelines, comment.char = "@", header = F)$V2
    return(df)
}

plot_image <- function(k_PC, col_fun, main_title, n_colours = ncol){
    col_vector = col_fun(n_colours)
    image(
        k_PC,
        col = col_vector,
        xlab = APC,
        ylab = BPC,
        xaxt = 'n',
        yaxt = 'n',
        main = main_title
    )
    axis(1, seq(-10,10,2))
    axis(2, seq(-10,10,2))
    abline(h = seq(-10,10,0.5), lty = 3, col = 'lightgrey')
    abline(v = seq(-10,10,0.5), lty = 3, col = 'lightgrey')
    image(
        k_PC,
        col = col_vector,
        xlab = APC,
        ylab = BPC,
        main = simprefix,
        add = T
    )
    contour(
        k_PC,
```

```
        col = 'grey10',
        add = T
    )
}

# read values
APC_values <- read_proj_xvg(paste(prefix, APC, '_proj.xvg', sep =''))
BPC_values <- read_proj_xvg(paste(prefix, BPC, '_proj.xvg', sep =''))

# calculate density
k_PC <- MASS::kde2d(APC_values, BPC_values, n = nbin)

# sim names
sim_names <- c(
    paste("R0", c(1:5), "_4AKE_WT" , sep = ''),
    paste("R0", c(1:5), "_4AKE_DM" , sep = '')
)
sim_col <- c(
  rep("blue", 5),
  rep("magenta", 5)
)

# plot
orange_palette <- colorRampPalette(c(rgb(1,1,1,0.2),"lightgoldenrod",
"goldenrod", "darkgoldenrod"), alpha = T)
par(mfrow=c(1,1))
plot_image(k_PC, orange_palette, simprefix)

# multiplot
#par(mfrow=c(2,5))
for(i in c(1:10)){
  png(paste(sim_names[i], '_PC1-PC2.png',sep = ''), width = 1350, height
= 1050)
  par(cex = 2)
  i_vector <- c(1:50000) + ((i-1)*50000)
  plot_image(k_PC, orange_palette, sim_names[i])
  points(
    APC_values[i_vector],
    BPC_values[i_vector],
    pch = '.',
    col = sim_col[i]
  )
  dev.off()
}
```

FA

target_distribution.R

```
library(ggplot2)
library(mclust)

set.seed(42)

peak1 <- rnorm(500000, mean = 2.4, sd = 0.1)
peak2 <- rnorm(4500005, mean = 3.55, sd = 0.4)

WT <- read.csv("R01_R05_4AKE_WT_prod_T300_COM.xvg", header = FALSE,
sep='')
target <- data.frame(Value = c(peak1, peak2), Group = "Target")
wt <- data.frame(Value = WT$V2, Group = "WT")
```

```
target_vs_WT <-rbind(target, wt)
target_vs_WT$Value <- target_vs_WT$Value * 10

create_target_plot <- function(data, x_label, plot_title, file_name) {
  p <- ggplot(data, aes(x = Value, fill = Group)) +
    geom_density(alpha = 0.5, colour =NA) +
    labs(title = plot_title, x = x_label, y = "Density", fill =
"Structure") +
    theme_minimal() +
    theme(plot.title = element_text(hjust = 0.5, size = 16, face =
"bold")) +
    scale_fill_manual(values = c("blue", "green"))

  ggsave(file_name, plot = p, width = 8, height = 6, dpi = 300)

}

create_target_plot(
  target_vs_WT,
  "COMDistance (Å)",
  "Target and WT for COMDistance",
  "target_distribution.png"
)
```

density_plots.py

```
library(ggplot2)
library(zoo)


setwd("~/Documents/PhD/ADK/xvg_plots_MD/jade/V135G/R01_4AKE_V135G/xvg")
R01_V135G_rmsd <- read.table("R01_4AKE_V135G_prod_T300_rmsd.xvg",
skip=18, col.names = c("FRAME", "RMSD"))
R01_V135G_rmsf <- read.table("R01_4AKE_V135G_prod_T300_rmsf.xvg",
skip=27, col.names = c("res", "RMSF"))
R01_V135G_Rg <- read.table("R01_4AKE_V135G_prod_T300_Rg.xvg", skip=27)


setwd("~/Documents/PhD/ADK/xvg_plots_MD/jade/V142G/R01_4AKE_V142G/xvg")
R01_V142G_Rg <- read.table("R01_4AKE_V142G_prod_T300_Rg.xvg", skip=27)
R01_V142G_rmsf <- read.table("R01_4AKE_V142G_prod_T300_rmsf.xvg",
skip=27, col.names = c("res", "RMSF"))
R01_V142G_rmsd <- read.table("R01_4AKE_V142G_prod_T300_rmsd.xvg",
skip=18, col.names = c("FRAME", "RMSD"))


setwd("~/Documents/PhD/ADK/xvg_plots_MD/jade/WT/R01_4AKE_WT/xvg")
R01_WT_Rg <- read.table("R01_4AKE_WT_prod_T300_Rg.xvg", skip=27)
R01_WT_rmsf <- read.table("R01_4AKE_WT_prod_T300_rmsf.xvg", skip=27,
col.names = c("res", "RMSF"))
```

```
R01_WT_rmsd <- read.table("R01_4AKE_WT_prod_T300_rmsd.xvg", skip=18,
col.names = c("FRAME", "RMSD"))

setwd("~/Documents/PhD/ADK/xvg_plots_MD/jade/DM/R02_4AKE_DM/xvg")
R02_DM_rmsf <- read.table("R02_4AKE_DM_prod_T300_rmsf.xvg", skip=27,
col.names = c("res", "RMSF"))
R02_DM_rmsd <- read.table("R02_4AKE_DM_prod_T300_rmsd.xvg", skip=18,
col.names = c("FRAME", "RMSD"))
R02_DM_Rg <- read.table("R02_4AKE_DM_prod_T300_Rg.xvg", skip=27)

R01_WT_rmsd$Condition    <- "WT"
R01_V135G_rmsd$Condition <- "V135G"
R01_V142G_rmsd$Condition <- "V142G"
R02_DM_rmsd$Condition    <- "DM"

combined_rmsd <- rbind(
  data.frame(FRAME = R01_WT_rmsd$FRAME, RMSD = R01_WT_rmsd$RMSD,
Condition = "WT"),
  data.frame(FRAME = R01_V135G_rmsd$FRAME, RMSD = R01_V135G_rmsd$RMSD,
Condition = "V135G"),
  data.frame(FRAME = R01_V142G_rmsd$FRAME, RMSD = R01_V142G_rmsd$RMSD,
Condition = "V142G"),
  data.frame(FRAME = R02_DM_rmsd$FRAME, RMSD = R02_DM_rmsd$RMSD,
Condition = "DM")
)

rmsd_plot <- ggplot(combined_rmsd, aes(x = FRAME, y = RMSD, color =
Condition)) +
  geom_line(size = 0.8) +
  labs(title = "RMSD",
       x = "Frame",
       y = "RMSD (nm)",
       color = "Structures") +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5, size = 14, face = "bold"))
ggsave("rmsd_overlayed_plot.png", plot = rmsd_plot, width = 8, height =
6, dpi = 300)

#################### smooth running average RMSD ######################
```

```
# Define a function to plot RMSD with consistent x-axis labels and
smoothed averages
plot_rmsd_with_consistent_x <- function(data, smooth_window = 100,
file_name = "rmsd_consistent_xaxis_plot.png") {

# Convert RMSD to Ångstroms (10x nanometers)
  data$RMSD <- data$RMSD * 10

  conditions <- unique(data$Condition)
  colors <- c("red", "blue", "green", "purple")
  png(file_name, width = 1200, height = 800, res = 150)
  plot(
    data$FRAME, data$RMSD, type = "n",  # Empty plot
    xlab = "Frame", ylab = "RMSD (Å)",
    main = "RMSD",
    ylim = range(data$RMSD, na.rm = TRUE),
    xlim = range(data$FRAME, na.rm = TRUE),
    xaxt = "n"  # Suppress default x-axis labels
  )

  for (i in seq_along(conditions)) {
    condition_data <- subset(data, Condition == conditions[i])

  Plot raw RMSD data as faint dashed lines
    lines(condition_data$FRAME, condition_data$RMSD, col = colors[i], lwd
= 0.5, lty = 3)

    # Calculate and plot smoothed RMSD data
    smoothed_rmsd <- rollmean(condition_data$RMSD, smooth_window, fill =
NA)
    lines(condition_data$FRAME, smoothed_rmsd, col = colors[i], lwd = 2)
  }
  axis(
    side = 1,  # Bottom axis
    at = seq(0, max(data$FRAME), by = 250000),  # Tick positions
    labels = format(seq(0, max(data$FRAME), by = 250000), scientific =
FALSE)  # Consistent number format
  )
   legend(
```

```
    "topright", legend = conditions, col = colors, lty = 1, lwd = 2,
    title = "Structure", bg = "white"
  )
   dev.off()
}

# Call the function to create the plot
plot_rmsd_with_consistent_x(
  combined_rmsd,
  smooth_window = 1000,  # Adjust the window for smoothing as needed
  file_name = "rmsd_smooth_running_average.png"
)

#################### Radis of Gyration#######################

plot_rg_with_consistent_x <- function(data, smooth_window = 100,
file_name = "rg_consistent_xaxis_plot.png") {
  data$Rg <- data$Rg * 10

  conditions <- unique(data$Condition)
  colors <- c("red", "blue", "green", "purple")

  png(file_name, width = 1200, height = 800, res = 150)

  plot(
    data$FRAME, data$Rg, type = "n",  # Empty plot
    xlab = "Frame", ylab = "Radius of Gyration (Å)",
    main = "Radius of Gyration",
    ylim = range(data$Rg, na.rm = TRUE),
    xlim = range(data$FRAME, na.rm = TRUE),
    xaxt = "n"  # Suppress default x-axis labels
  )

  for (i in seq_along(conditions)) {
    condition_data <- subset(data, Condition == conditions[i])

    lines(condition_data$FRAME, condition_data$Rg, col = colors[i], lwd =
0.5, lty = 3)
```

```
    smoothed_rg <- rollmean(condition_data$Rg, smooth_window, fill = NA)
    lines(condition_data$FRAME, smoothed_rg, col = colors[i], lwd = 2)
  }

  axis(
    side = 1,  # Bottom axis
    at = seq(0, max(data$FRAME), by = 250000),  # Tick positions
    labels = format(seq(0, max(data$FRAME), by = 250000), scientific =
FALSE)  # Consistent number format
  )

  legend(
    "topright", legend = conditions, col = colors, lty = 1, lwd = 2,
    title = "Structure", bg = "white"
  )

  dev.off()
}

plot_rg_with_consistent_x(
  combined_Rg,
  smooth_window = 1000,
  file_name = "rg_smooth_running_average.png"
)



####################  RMSF ######################

R01_WT_rmsf$Condition    <- "WT"
R01_V135G_rmsf$Condition <- "V135G"
R01_V142G_rmsf$Condition <- "V142G"
R02_DM_rmsf$Condition    <- "DM"

combined_rmsf <- rbind(
  data.frame(Residue = R01_V135G_rmsf$res, RMSF = R01_V135G_rmsf$RMSF,
Condition = "V135G"),
  data.frame(Residue = R01_V142G_rmsf$res, RMSF = R01_V142G_rmsf$RMSF,
Condition = "V142G"),
```

```
  data.frame(Residue = R01_WT_rmsf$res, RMSF = R01_WT_rmsf$RMSF,
Condition = "WT"),
  data.frame(Residue = R02_DM_rmsf$res, RMSF = R02_DM_rmsf$RMSF,
Condition = "DM")
)

combined_rmsf$RMSF <- combined_rmsf$RMSF * 10

rmsf_plot <- ggplot(combined_rmsf, aes(x = Residue, y = RMSF, color =
Condition)) +
  geom_line(size = 0.8) +
  labs(title = "RMSF",
       x = "Residue",
       y = "RMSF (Å)",
       color = "Structures") +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5, size = 14, face = "bold"))

ggsave("rmsf_overlayed_plot.png", plot = rmsf_plot, width = 8, height =
6, dpi = 300)

#################### Rg density plot ##################

R01_V135G_Rg$Condition <- "V135G"
R01_V142G_Rg$Condition <- "V142G"
R01_WT_Rg$Condition    <- "WT"
R02_DM_Rg$Condition    <- "DM"

combined_Rg <- rbind(
  data.frame(FRAME = R01_V135G_Rg$V1, Rg = R01_V135G_Rg$V2, Condition =
"V135G"),
  data.frame(FRAME = R01_V142G_Rg$V1, Rg = R01_V142G_Rg$V2, Condition =
"V142G"),
  data.frame(FRAME = R01_WT_Rg$V1, Rg = R01_WT_Rg$V2, Condition = "WT"),
  data.frame(FRAME = R02_DM_Rg$V1, Rg = R02_DM_Rg$V2, Condition = "DM")
)

Rg_plot <- ggplot(combined_Rg, aes(x = FRAME, y = Rg, color = Condition))
+
```

```
  geom_line(size = 0.8) +
  labs(title = "Radius of gyration",
       x = "Frame",
       y = "Radius of Gyration (nm)",
       color = "Structures") +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5, size = 14, face = "bold"))

ggsave("Rg_overlayed_plot.png", plot = Rg_plot, width = 8, height = 6,
dpi = 300)

###################### COMDistance density####################
setwd("~/Dropbox/Thesis draft/figures/xvg_WT_DM_COM Distance")
R02_DM_COMDistance <- read.table("R02_4AKE_DM_prod_T300_COM.xvg", skip
=25)
R01_WT_COMDistance <- read.table("R01_4AKE_WT_prod_T300_COM.xvg",
skip=25)

R01_WT_COMDistance$Structure <- "WT"
R02_DM_COMDistance$Structure <- "DM"

  combined_COMDistance <- rbind(
    data.frame(COMDistance = R01_WT_COMDistance$V2, Structure =
R01_WT_COMDistance$Structure),
    data.frame(COMDistance = R02_DM_COMDistance$V2, Structure =
R02_DM_COMDistance$Structure)
  )

  combined_COMDistance$COMDistance <- combined_COMDistance$COMDistance *
10

density_plot <- ggplot(combined_COMDistance, aes(x = COMDistance, fill =
Structure)) +
  geom_density(alpha = 0.5, adjust = 1) +  # Adjust alpha for
transparency
  labs(title = "COMDistance: R02_WT_vs_R02_DM",
       x = "COMDistance (Å)",
       y = "Density",
       fill = "Structure") +
```

```
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5, size = 14, face = "bold"))

ggsave("COMDistance_angstrom.png", plot = density_plot, width = 8, height
= 6, dpi = 300)

ggplot(DM_COM, aes(x = value)) +
  geom_density(fill = "blue", alpha = 0.5) +  # Adjust fill color and
transparency
  labs(title = "Density Plot of Value",
       x = "Value",
       y = "Density") +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5, size = 14, face = "bold"))

##########################density plots#############################
setwd("~/Documents/PhD/ADK/xvg_plots_MD/jade/DM/R01_R05_4AKE_DM/xvg")
R01_R05_DM_rmsd <- read.table("R01_R05_4AKE_DM_prod_T300_rmsd.xvg",
skip=1)
R01_R05_DM_Rg <- read.table("R01_R05_4AKE_DM_prod_T300_Rg.xvg", skip=1)
R05_DM_G55_P127 <- read.table("R01_R05_4AKE_DM_prod_T300_G55-P127.xvg",
skip=1)

setwd("~/Documents/PhD/ADK/xvg_plots_MD/jade/WT/R01_R05_4AKE_WT/xvg")
R01_R05_WT_rmsd <- read.table("R01_R05_4AKE_WT_prod_T300_rmsd.xvg",
skip=1)
R01_R05_WT_Rg <- read.table("R01_R05_4AKE_WT_prod_T300_Rg.xvg", skip=1)
R05_WT_G55_P127 <- read.table("R01_R05_4AKE_WT_prod_T300_G55-P127.xvg",
skip=1)

R01_R05_DM_rmsd$Structure <- "DM"
R01_R05_WT_rmsd$Structure <- "WT"
R01_R05_DM_Rg$Structure <- "DM"
R01_R05_WT_Rg$Structure <- "WT"
R05_DM_G55_P127$Structure <- "DM"
R05_WT_G55_P127$Structure <- "WT"

# Combine datasets for each property
combined_rmsd <- rbind(
```

```
  data.frame(Time = R01_R05_DM_rmsd$V1, Value = R01_R05_DM_rmsd$V2,
Structure = R01_R05_DM_rmsd$Structure),
  data.frame(Time = R01_R05_WT_rmsd$V1, Value = R01_R05_WT_rmsd$V2,
Structure = R01_R05_WT_rmsd$Structure)
)

combined_Rg <- rbind(
  data.frame(Time = R01_R05_DM_Rg$V1, Value = R01_R05_DM_Rg$V2, Structure
= R01_R05_DM_Rg$Structure),
  data.frame(Time = R01_R05_WT_Rg$V1, Value = R01_R05_WT_Rg$V2, Structure
= R01_R05_WT_Rg$Structure)
)

combined_G55_P127 <- rbind(
  data.frame(Time = R05_DM_G55_P127$V1, Value = R05_DM_G55_P127$V2,
Structure = R05_DM_G55_P127$Structure),
  data.frame(Time = R05_WT_G55_P127$V1, Value = R05_WT_G55_P127$V2,
Structure = R05_WT_G55_P127$Structure)
)

combined_rmsd$Value <- combined_rmsd$Value * 10
combined_Rg$Value <- combined_Rg$Value * 10
combined_G55_P127$Value <- combined_G55_P127$Value * 10

# Function to create and save density plots
create_density_plot <- function(data, x_label, plot_title, file_name) {
  p <- ggplot(data, aes(x = Value, fill = Structure)) +
    geom_density(alpha = 0.5, colour =NA) +
    labs(title = plot_title, x = x_label, y = "Density", fill =
"Structure") +
    theme_minimal() +
    theme(plot.title = element_text(hjust = 0.5, size = 14, face =
"bold"))

  ggsave(file_name, plot = p, width = 8, height = 6, dpi = 300)
}

# RMSD Density Plot
create_density_plot(
```

```
  combined_rmsd,
  "RMSD (Å)",
  "RMSD : R01_R05_WT_DM",
  "RMSD_density_comparison_angstrom.png"
)

# Rg Density Plot
create_density_plot(combined_Rg, "Rg (Å)", "Rg : R01_R05_WT_DM",
"Rg_density_comparison_angstrom.png")

# G55-P127 Distance Density Plot
create_density_plot(combined_G55_P127, "Distance between G55 and P127
(Å)", "dist_G55_P127 : R01_R05_WT_DM",
"G55_P127_distance_density_comparison_angstrom.png")
```

Density_Rg_Distance_target_labels.R

```
---
title: "R Notebook"
output: html_notebook
editor_options:
  chunk_output_type: console
---

### 0. Load packages and set paramters

```{r}
library(MASS)
library(DescTools)
library(grDevices)
library(sf)
library(sp)

data_file <- "MD05_ADK_protein_large_Rg-dG55P127.csv"
out_file <- "MD05_ADK_protein_large_Rg-dG55P127_output.csv"
target_var_file <- "MD05_ADK_protein_large_Rg-dG55P127_target_var.dat"
png_file <- "MD05_ADK_protein_large_Rg-dG55P127.png"

y_cutoff <- 1.765
x_idx <- 2
```
```

````
y_idx <- 3
```


### 1. Load data

```{r}
d <- read.csv(data_file)
```


### 2. Calculate density

```{r}
d_kde <- kde2d(d[,x_idx], d[,y_idx], n = 500)
```


### 3. Create contour

```{r}
cnt_d_kde <- contourLines(d_kde)
```


### 4. Annotate contour to data

```{r}
n_cnt <- length(cnt_d_kde)
cnt_levels <- sapply(cnt_d_kde, function(x){x$level})
ant <- sapply(c(1:n_cnt), function(i){
    i_cnt <- cnt_d_kde[[i]]
    point.in.polygon(d[,x_idx], d[,y_idx], i_cnt$x, i_cnt$y)*i
  }
)
d <- cbind(d, ant)
d$highest <- apply(ant, 1, max)
d$level <- sapply(d$highest, function(x){
  if(x == 0){
    0
    }else{
    cnt_levels[x]}
  }
````

```
)
```

5. Annotate counter table

```{r}
cnt_table <- data.frame(cnt_id = NA, level = cnt_levels, basin_id = NA)
for(i in c(1:length(cnt_levels))){
  ave_y <- mean(d[d$highest == i, 2])
  if(ave_y >= y_cutoff){
    cnt_table[i, 'basin_id'] = 'A'
    cnt_table[i, 'cnt_id'] = paste('A',cnt_table[i,'level'], sep = '_')
    }else{
    cnt_table[i, 'basin_id'] = 'B'
    cnt_table[i, 'cnt_id'] = paste('B',cnt_table[i,'level'], sep = '_')
  }
}


```

6. Annotate data table with basin

```{r}
d$basin <- 'N'
d[d$highest %in% which(cnt_table$basin_id == 'A') ,'basin'] <- 'A'
d[d$highest %in% which(cnt_table$basin_id == 'B') ,'basin'] <- 'B'
```

7. Plot contour lines

```{r}
rgb.palette <-
colorRampPalette(c("yellow","gold","orange","red","darkred"))
 plot(d[,c(x_idx,y_idx)], pch = '.', col = 'grey')

for(i in c(1:n_cnt)){
  if(cnt_table[i, 'basin_id'] == 'A'){
```

```
    points(d[d$highest == i ,c(x_idx,y_idx)], col
=rgb.palette(9)[cnt_table[i,'level']])
  }
}
for(i in c(1:n_cnt)){
  if(cnt_table[i, 'basin_id'] == 'B'){
    points(d[d$highest == i ,c(x_idx,y_idx)], col
=rgb.palette(9)[cnt_table[i,'level']])
  }
}
contour(d_kde, col = 'grey', add = T)

```

8. Set possible target variables

8.1 Label points by basin

```{r}
d$target_01 <- d$basin
```

8.2 Label points by basin with depth level greater than 4

```{r}
d$target_02 <- 0
d[d$highest > 4, 'target_02'] <- 1
```

8.3 Label points by two deeper region in each basin

```{r}
d$target_03 <- 'N'
A_deeper <- max(which(cnt_table$basin_id == 'A'))
B_deeper <- max(which(cnt_table$basin_id == 'B'))
d[d$highest == A_deeper, 'target_03'] <- 'A'
d[d$highest == B_deeper, 'target_03'] <- 'B'
```

````
#### 8.4 Label points by basin with depth level greater than 4 and
different basin label

```{r}
d$target_04 <- 'N'
d[d$highest > 4 & d$basin == 'A', 'target_04'] <- 'A'
d[d$highest > 4 & d$basin == 'B', 'target_04'] <- 'B'
```

### 9. Save data

```{r}
write.csv(file = out_file, d, quote = F, row.names = F)

write.table(file = target_var_file, d[, c('Time', "target_04")],
row.names = F, quote = F)
```



### 10. Graphs

```{r}
par(las = 1)
plot(
  d[,c(x_idx,y_idx)],
  pch = '.',
  col = 'grey',
  type = 'n',
  xlim = c(0.5,4.5),
  ylim = c(1.55,2.25),
  xlab = 'Gly 55 - Pro 127',
  ylab = expression(R[g]),
  xaxt = 'n',
  yaxt = 'n',
  bty = 'l'
)
axis(1, seq(0, 5, 0.25))
axis(2, seq(1.5, 2.5, 0.05))
abline(v = seq(0, 5, 0.25), col = 'lightgray')
````
````

```
abline(h = seq(1.5, 2.5, 0.05), col = 'lightgray')
points(d[,c(x_idx,y_idx)],  pch = '.',  col = 'grey50' )
contour(d_kde, col = SetAlpha('red', 1.5), lwd = 1.5, add = T, drawlabels
= FALSE)
```

```{r}
png(file = png_file, width = 1000, height = 1200)
par(las = 1)
par(cex = 2)
plot(
  d[,c(x_idx,y_idx)],
  pch = '.',
  col = 'grey',
  type = 'n',
  xlim = c(0.5,4.5),
  ylim = c(1.55,2.25),
  xlab = 'Gly 55 - Pro 127 / nm',
  ylab = expression(R[g] / nm),
  xaxt = 'n',
  yaxt = 'n',
  bty = 'l'
)
axis(1, seq(0, 5, 0.5))
axis(2, seq(1.5, 2.5, 0.05))
abline(v = seq(0, 5, 0.5), col = 'lightgray')
abline(h = seq(1.5, 2.5, 0.05), col = 'lightgray')
points(d[,c(x_idx,y_idx)],  pch = '.',  col = 'grey50' )
points(d[d$highest > 4,c(x_idx,y_idx)],  pch = '.',  col = 'grey20' )
contour(d_kde, col = SetAlpha('red', 1.5), lwd = 1.5, add = T, drawlabels
= FALSE)
dev.off()
```

```{r}
# rgb.palette <- colorRampPalette(
#   c("yellow","gold","orange","red","darkred"),
#   alpha = T
```

```
# )
# color_vector <-
as.vector(SetAlpha(rgb.palette(max(cnt_table[,'level'])), 0.3))

# for(i in c(1:n_cnt)){
#   if(cnt_table[i, 'basin_id'] == 'A'){
#     if (cnt_table[i,'level'] > 4){
#       points(d[d$highest == i ,c(x_idx,y_idx)], col
=color_vector[cnt_table[i,'level']])
#     }
#   }
# }
# for(i in c(1:n_cnt)){
#   if(cnt_table[i, 'basin_id'] == 'B'){
#     if (cnt_table[i,'level'] > 4){
#       points(d[d$highest == i ,c(x_idx,y_idx)], col
=color_vector[cnt_table[i,'level']])
#     }
#   }
# }
```