Efficient Distributed Data Generation and Training of Graph Neural Network–Based Molecular Dynamics using Weighted Ensemble Sampling

by

Alexander Aghili

A thesis submitted in partial fulfillment of the requirements

for the degree of

Bachelor of Science in Computer Science

December 2025

University of California, Santa Cruz

Jack Baskin School of Engineering

Abstract

This work investigates how weighted ensemble (WE) simulation data can be used to train neural-network coarse-grained molecular dynamics models more effectively. WE sampling produces a broader range of molecular configurations, including transitions that standard simulations rarely observe. To make use of this data, a modified force-matching loss is introduced in which each training frame is scaled by its WE weight. This approach allows the learning process to approximate the behavior of an unbiased dataset while benefiting from the expanded coverage that WE provides.

Model performance is evaluated on a benchmark of eight fast-folding proteins, comparing training with and without WE weights. Across multiple systems, models trained with weighted data show improved long-term stability and better preservation of structural features such as helices, while unweighted models more frequently collapse into non-physical shapes. Although the weighted approach does not outperform the unweighted one on every metric, the overall results indicate that incorporating WE weights during training offers a practical and effective way to improve model robustness when using enhanced-sampling data.

Contents

1 Introduction **6**

2 Review **8**
2.1 Molecular Dynamics 8
2.1.1 Historical Context 8
2.1.2 Theory 10
2.1.3 Modern Methods 12
2.2 Weighted Ensemble Sampling 13
2.3 Machine Learning Methods for Molecular Dynamics 16

3 Benchmarking Neural Network Force Fields **19**
3.1 Ground-Truth Dataset 19
3.2 Benchmarking System 20
3.3 Metrics 20

4 Weighted Ensemble Data Generation **23**
4.1 Model Theory - Weighted Ensemble Data 23
4.2 Data Generation 26
4.3 Data Pipeline and Implementation Details 29
4.4 Results 32
4.5 Discussion 38
4.6 Future Work 39

5 Optimizations **40**
5.1 Distributed Training 40
5.2 Trajectory Packing 44
5.3 Future Optimizations 49

Bibliography **49**

6 Appendix **55**
6.1 Proof of Theorem 2 55
6.2 GPU Utilization Graphs 59

List of Figures

2.1 Classical force-field potential energy calculation . . . 11
2.2 Chignolin WESTPA vs Boltzmann Sampling in TICA Space with Equal Timescale 15
2.3 Homeodomain WESTPA vs Boltzmann Sampling in TICA Space with Equal Compute . . . 16

3.1 Architecture of the weighted ensemble (WE) benchmarking framework. . . . 21

4.1 Trp-cage protein with $C\alpha$ backbone . . . 30
4.2 Comparison of TICA Space Exploration of Unweighted and Weighted Model for Chignolin . . . 33
4.3 Comparison of TICA Space Exploration of Unweighted and Weighted Model for WWdomain . . . 34
4.4 Comparison of Unweighted and Weighted model PDF results for Homeodomain . . 35
4.5 Close up of single Homeodomain structure to compare unweighted and weighted model. . . . 36
4.6 Seven trajectories of Homeodomain used to compare weighted and unweighted models geometric structure. . . . 36
4.7 λ-repressor structure over time with weighted and unweighted models. . . . 37
4.8 Protein G Structure Comparison . . . 37

5.1 Total epochs evaluated against the number of nodes . . . 42
5.2 Average time per epoch evaluated against the number of nodes . . . 43
5.3 Trajectory Packing Tensor . . . 45
5.4 Time to compute a segment over block_size . . . 45
5.5 Time to compute a segment over block_size (16+) . . . 46
5.6 GPU Computations vs Block Size . . . 46
5.7 GPU utilization for GPUs 1-3 over time for a block_size of 1 . . . 47
5.8 GPU utilization for GPUs 1-3 over time for a block_size of 128 . . . 47
5.9 GPU utilization for GPUs 1-3 over time for a block_size of 1024 . . . 48

6.1 GPU Utilization Graph: Block Size = 1 . . . 60
6.2 GPU Utilization Graph: Block Size = 4 . . . 60
6.3 GPU Utilization Graph: Block Size = 8 . . . 61
6.4 GPU Utilization Graph: Block Size = 16 . . . 61
6.5 GPU Utilization Graph: Block Size = 32 . . . 62
6.6 GPU Utilization Graph: Block Size = 64 . . . 62
6.7 GPU Utilization Graph: Block Size = 128 . . . 63

6.8 GPU Utilization Graph: Block Size = 1024 63

List of Tables

3.1 Protein benchmarks with representative structures. 19

4.1 Simulation parameters for data generation 27
4.2 Simulation statistics for benchmark proteins of training data 28
4.3 Training and Data Pipeline Hyperparameters 32
4.4 [KL-Divergence Results Table]KL-Divergence – $D_{\mathrm{KL}}(\mathrm{GT} \,||\, \mathrm{Model})$ – for the weighted (W) and the unweighted (UW) model PDFs across TIC 0, TIC 1, and structural features: bond lengths, angles and dihedrals for all proteins in our benchmark. . . . 32

Chapter 1

Introduction

Molecular dynamics (MD) provides a mechanistic window into the motions of atoms and molecules by numerically integrating Newton's equations under a chosen potential energy function. Over the past several decades, MD has become indispensable for studying protein folding, conformational transitions, ligand binding, and materials behavior at atomic resolution. Classical all-atom force fields, combined with efficient integrators and long-range interaction schemes, now support microsecond-scale simulations on supercomputing hardware. Nonetheless, many biologically relevant processes occur on millisecond or longer timescales and involve rare transitions between metastable states, making them prohibitively expensive to sample with conventional Boltzmann-weighted dynamics.

Machine-learned potentials offer a promising way to mitigate this bottleneck by approximating complex energy landscapes directly from data. Recent graph neural network (GNN) and equivariant transformer architectures treat molecular systems as graphs and enforce the geometric symmetries required for physically consistent force predictions. In the coarse-grained (CG) setting, atomistic configurations and forces are mapped onto reduced representations, and a CG potential is learned through force matching. The effectiveness of these models, however, depends strongly on the diversity and statistical quality of the training data used to fit them.

Weighted ensemble (WE) sampling provides a complementary strategy by enhancing the ex-

ploration of conformational space. WE maintains an ensemble of weighted trajectories and periodically applies splitting and merging to focus computation on rarely visited regions while preserving unbiased estimates of observables. This makes WE an attractive source of training data, but its nonequilibrium sampling behavior means that uncorrected WE trajectories can bias the learning process and produce unstable or physically unrealistic CG models.

This thesis examines how WE sampling can be incorporated into the training of GNN-based CG molecular dynamics models in a statistically accurate and computationally scalable manner. A weighted force-matching loss is derived to properly account for WE trajectory weights, and a full preprocessing pipeline is used to convert WESTPA output into coarse-grained coordinates. Weighted and unweighted training are compared across eight fast-folding proteins using structural and kinetic benchmarks. Additional optimizations are introduced to improve computational efficiency. Together, these components demonstrate both the potential and limitations of WE-based data generation for machine-learned molecular dynamics and establish a framework for large-scale, enhanced sampling driven CG model development.

Chapter 2

Review

2.1 Molecular Dynamics

2.1.1 Historical Context

Long before the development of modern microscopic dynamical models, scientists sought to understand how forces produce motion across vastly different length scales. Isaac Newton's *Principia* (1687) provided a universal set of laws including this second law,

$$\mathbf{F} = m\mathbf{a},$$

that successfully described planetary trajectories and macroscopic mechanics. Yet when it came to the erratic movement of particles only a few micrometers in size, Newton's deterministic laws alone were insufficient.

A major observational breakthrough occurred in 1827, when botanist Robert Brown examined pollen grains of *Clarkia pulchella* under a microscope and noticed that the minute particles they released moved in a jittery manner. By repeating the experiment with finely powdered inorganic materials, Brown demonstrated that the motion was independent of biological processes.

During the second half of the 19th century, Ludwig Boltzmann and others advanced the ki-

netic theory of matter, proposing that gases and liquids consist of myriad molecules undergoing perpetual motion. This statistical viewpoint suggested that microscopic particles suspended in a fluid are incessantly buffeted by countless molecular impacts, but no quantitative link had yet been established between those unseen molecular motions and Brown's visible particle trajectories.

Such a connection was finally made in the early 20th century. In 1905, Albert Einstein published his groundbreaking analysis of Brownian motion, providing a statistical description that tied observable particle diffusion directly to molecular kinetics. Einstein clarified that the irregular movement of a suspended particle arises from the continual, unequal, and discrete collisions imparted by surrounding fluid molecules. Because these molecular impacts vary in direction and magnitude from moment to moment, the particle executes a randomized walk. By deriving explicit relations, Einstein showed that the chaotic trajectory of a micron-scale particle is a direct macroscopic signature of atomic-scale dynamics.

Only a few years later, in 1908, Paul Langevin provided a complementary and more dynamical formulation. Combining Newton's second law with a linear friction force and a rapidly fluctuating random force representing the instantaneous molecular impacts on the particle, Langevin arrived at what is now known as the Langevin equation. Langevin's framework laid the conceptual foundation for modeling systems where inertia, friction, and stochastic molecular forces interact, an essential perspective for describing the dynamics of particles, molecules, and condensed-phase systems at small scales.

The development of particle-motion theory laid the groundwork to enable computational methods for studying many-body systems. As electronic computers began to appear, researchers increasingly recognized the profound potential of these machines for performing large-scale numerical calculations that had previously been infeasible. In one of the earliest demonstrations, Alder and Wainwright (1957) employed computer simulations of hard-sphere gases to investigate phase transitions and transport phenomena [1]. Their work showed that the deterministic trajectories of individual particles are capable of reproducing macroscopic thermodynamic behavior, thereby providing direct evidence that microscopic dynamics could be computationally modeled

to yield insights into bulk physical properties. Building on this, Rahman (1964) conducted the first molecular-dynamics simulation of a realistic fluid by modeling liquid argon with a continuous Lennard–Jones potential [2]. His integration of Newton's equations of motion for such a system validated the feasibility of simulating interatomic interactions with higher fidelity. Over the next decades, numerical methods such as the Verlet algorithm [3], constraint algorithms [4] [5], and Particle Mesh Ewald (PME) summations [6] were developed for faster and more accurate molecular simulations.

Molecular dynamics began to turn into a tool for use in the biophysics community. McCammon, Gelin, and Karplus (1977) demonstrated atomic-level molecular dynamics to simulate how proteins fold [7]. Software packages and atomic force fields such as GROMACS [8], CHARMM [9], and OpenMM [10] were developed to provide scalable, centralized frameworks for conducting molecular simulations, which are commonly used today across universities, national laboratories, and in industry.

2.1.2 Theory

Molecular dynamics (MD) is a computational technique that simulates the behavior of atomic and molecular systems by applying Newton's second law to determine the forces acting on each particle. These forces are then numerical integrated to update particle velocities and positions over time, allowing the system's evolution to be tracked at atomic resolution. First, an initial structure with atomic coordinates R must be provided, either with experimental techniques like nuclear magnetic resonance (NMR) and X-ray crystallography, or with estimation techniques like Alphafold[11]. Then the potential energy is calculated.

For each atom i, the force $\mathbf{F}_i$ is obtained as the negative gradient of the potential

$$\mathbf{F}_i(t) = -\nabla_{r_i} U(R(t))$$

$$U(R) = \sum_{bonds} k_r(r - r_{eq})^2 + \sum_{angles} k_\theta(\theta - \theta_{eq})^2 + \sum_{dihedrals} k_\phi \left[1 + \cos(n\phi - \gamma)\right]$$
$$+ \sum_{i<j}^{atoms} \left[\frac{A_{ij}}{r_{ij}^{12}} - \frac{B_{ij}}{r_{ij}^{6}}\right] + \sum_{i<j}^{atoms} \frac{q_i q_j}{4\pi\varepsilon_0 r_{ij}}$$

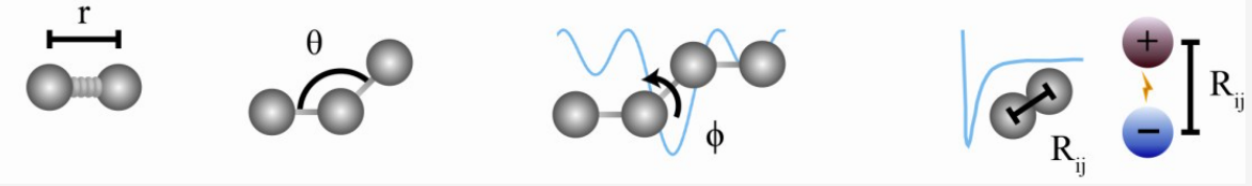

Figure 2.1: Classical force field calculation of potential energy. Includes the terms and associated diagrams for the bonds, angles, dihedrals, van der Waals forces, and electrostatic forces. Here, r is the bond length, θ is the bond angle, ϕ is the dihedral angle, with k_r, k_θ, and k_ϕ as force constants and r_{eq} and θ_{eq} as the equilibrium positions. r_{ij} is the distance between atoms.

Apply Newton's second law

$$\mathbf{a}_i(t) = \frac{\mathbf{F}_i(t)}{m_i} \tag{2.1}$$

The position $\mathbf{r}$ and velocities $\mathbf{v}$ can be obtained via integration,

$$\mathbf{v}_i(t) = \int \mathbf{a}_i(t)dt$$

and

$$\mathbf{r}_i(t) = \int \mathbf{v}_i(t)dt = \iint \mathbf{a}_i(t)dt$$

The integration must be done numerically, so an integration scheme is used. The Velocity-Verlet scheme is a common scheme which updates the positions and velocities as follows. Given known quantities at time t, $\mathbf{r}_i(t), \mathbf{v}_i(t)$, and $\mathbf{a}_i(t)$ (either from previous iteration or from the original structure), then first calculate the update position from the velocity and acceleration

$$\mathbf{r}_i(t + \Delta t) = \mathbf{r}_i(t) + \mathbf{v}_i(t)\Delta t + \frac{1}{2}\mathbf{a}_i(t)\Delta t^2$$

Recompute the forces the get new accelerations from (1) and then update the velocities

$$\mathbf{v}_i(t+\Delta t) = \mathbf{v}_i(t) + \frac{1}{2}\left[\mathbf{a}_i(t) + \mathbf{a}_i(t+\Delta t)\right]\Delta t$$

In order to maintain numerical stability and avoid large errors, Δt is often restricted to $1-4$ femtoseconds (fs). This highlights a significant limitation of MD, which is that it requires small timesteps, each of which require $O(N^2)$ or $O(N\log N)$ computations with N atoms. Because many biological processes occur on millisecond-to-second timescales and span whole-cell dimensions involving billions of atoms, they are typically far too large and slow to simulate with conventional molecular dynamics. As a result, several techniques have been developed to extend molecular dynamics to biologically relevant temporal and spatial scales which are discussed in section 2.1.3.

When it comes to simulating molecular systems, Langevin dynamics extends deterministic Newtonian motion by introducing an implicit heat bath. In addition to the Newtonian forces, each atom also experiences a frictional drag and stochastic random force representing thermal fluctuations from the surrounding solvent. The equation of motion becomes

$$m_i d\mathbf{v}_i(t) = \mathbf{F}_i(t)dt - \gamma_i m_i \mathbf{v}_i(t)dt + \sqrt{2m_i\gamma_i k_B T}\,d\mathbf{W}(t)$$

where γ_i is the friction coefficient, $-\gamma_i m_i \mathbf{v}_i(t)$ is a viscous damping term, T is the temperature, and $d\mathbf{W}(t)$ is a Wiener process introducing stochastic variation sampling from the canonical (NVT) ensemble. This is the formulation used in the molecular dynamics (OpenMM) simulations used in this thesis.

2.1.3 Modern Methods

A variety of strategies have been developed to overcome the timescale bottleneck in molecular dynamics (MD) simulations and thereby access biologically relevant motions. One class of approaches focuses on maximizing raw computational speed through specialized hardware. The most notable example is the work of D. E. Shaw Research which, in collaboration with the National Sci-

ence Foundation and the Pittsburgh Supercomputing Center, developed the *Anton*[12] series of supercomputers, using ASICs designed explicitly for MD simulations. Although these systems have demonstrated exceptional performance, their utility remains limited by their restricted availability.

A complementary line of research recognizes that many slow biomolecular processes correspond to rare events: systems tend to reside for long periods near minima on the free-energy landscape, with infrequent transitions between metastable states. This observation motivates the introduction of statistical biases to preferentially sample otherwise underrepresented regions of phase space. Methods such as umbrella sampling, metadynamics, and weighted ensembles accelerate sampling by altering the effective energy landscape to encourage transitions along selected collective variables.

2.2 Weighted Ensemble Sampling

Weighted ensemble (WE) sampling [13] [14] is a resampling-based path-ensemble method designed to enhance the sampling of rare events in stochastic dynamical systems. Consider a Markov process $(X_t)_{t\geq 0}$ on a state space Ω with transition kernel $P_{\Delta t}(x, dy)$. A WE simulation maintains an ensemble of N independent trajectories ("walkers"), each carrying a non-negative weight $w^{(i)}$ satisfying $\sum_{i=1}^{N} w^{(i)} = 1$. The essential idea is that walkers propagate according to the true dynamics,

$$X_{t+\Delta t}^{(i)} \sim P_{\Delta t}(X_t^{(i)}, \cdot),$$

and their weights remain unchanged during propagation. Only during a structured resampling step, performed at fixed time intervals τ, are weights redistributed via splitting and merging operations while preserving unbiased expectations for observables over path space.

In WE, the state space Ω is partitioned into a finite measurable set of bins $\{B_k\}_{k=1}^{K}$, and the resampling step is applied within each bin independently [15]. If I_k denotes the index set of walkers currently occupying bin B_k and $W_k = \sum_{i\in I_k} w^{(i)}$ the total weight of that bin, then WE attempts to

maintain a target number M_k of walkers in that bin. When a walker has weight significantly larger than W_k/M_k, it is split into

$$m_i = \left\lfloor \frac{w^{(i)}}{W_k/M_k} \right\rfloor$$

clones, each carrying equal weight $\tilde{w}^{(i)} = w^{(i)}/m_i$. Conversely, if too many low-weight walkers occur in a bin, WE performs stochastic merging, usually via a multinomial-like selection in which one survivor walker is chosen with probability proportional to its weight and assigned the entire merged weight W_k, while the remaining walkers in that merge group are removed. Both splitting and merging preserve the expected contribution of each trajectory: for any bounded observable g, one has

$$\mathbb{E}\Big[\sum_i \tilde{w}^{(i)} g(X_t^{(i)})\Big] = \sum_i w^{(i)} g(X_t^{(i)}),$$

because splitting is deterministic weight partitioning, and merging uses a selection rule satisfying $\mathbb{E}[\tilde{w}^{(i)}] = w^{(i)}$. As a result, for any path-dependent observable $f(\gamma)$, the WE estimator

$$\hat{f}_N = \sum_i w^{(i)} f(\gamma^{(i)})$$

remains unbiased, regardless of the choice of bins or target populations [16]. This property is a central theorem of the WE framework and distinguishes it from biased rare-event methods such as umbrella sampling or metadynamics.

In practice, however, the choice of bins strongly affects efficiency, and it is generally advantageous to adapt bin boundaries based on the evolving ensemble rather than fixing them a priori. A minimal adaptive binning (MAB) scheme [17] provides a simple, automated way to place bins along a chosen progress coordinate within WE. At each resampling time, MAB identifies the trailing and leading trajectories and one or more "bottleneck" trajectories where the cumulative weight has dropped most sharply along the progress coordinate. A fixed, relatively small number of bins is then laid down evenly between the trailing and leading trajectories, while each boundary and bottleneck trajectory is assigned its own bin, standard WE splitting/merging is subsequently applied

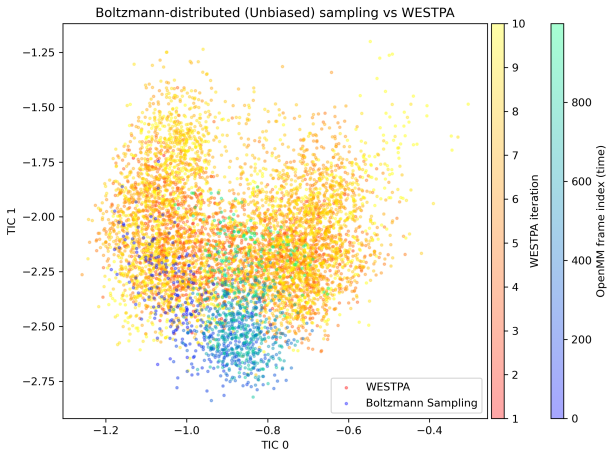

Figure 2.2: Chignolin Protein in Time-lagged Independent Component Analysis (TICA) Space (TIC 0 and TIC 1, TICA is a measure of conformational variance) across several timesteps. Regular OpenMM (Blue) simulation (Boltzmann Sampling) compared to WESTPA (Yellow) for the same timescale (the amount of simulated time the protein has run). As shown, there is greater deviation from the starting point, meaning that for the same timescale, WESTPA allows proteins to vary more in conformational space (as opposed to staying in the local region, as is common with Boltzmann statistics). However, note that WESTPA generates several parallel trajectories for the same timescale (eg. 0-100 fs have 4 walkers, while regular OpenMM only has 1 “walker"). Therefore, the compute required for WESTPA remains greater.

within these adaptively updated bins. This procedure focuses sampling near free-energy barriers and other bottlenecks, improves transition rates and path diversity relative to manual, fixed binning, and requires far fewer bins overall, all while preserving the unbiased nature of WE estimates.

Weighted ensemble sampling is also embarrassingly parallel. During each propagation interval $[t, t+\Delta t]$, all walkers evolve independently, requiring no communication between them. The resampling step is also naturally parallelizable because it is performed bin by bin, and operations within a bin depend only on walkers inside that bin. Thus, if one distributes walkers across processors such that each processor manages either a subset of bins or a subset of trajectories, then propagation requires zero communication, and resampling requires only a lightweight reduction operation to compute per-bin total weights W_k and a few random selections. Formally, if C_{prop} denotes the computational cost of propagating one walker for a time step and C_{comm} the cost of cross-bin communication, then WE typically achieves speedups proportional to the number of pro-

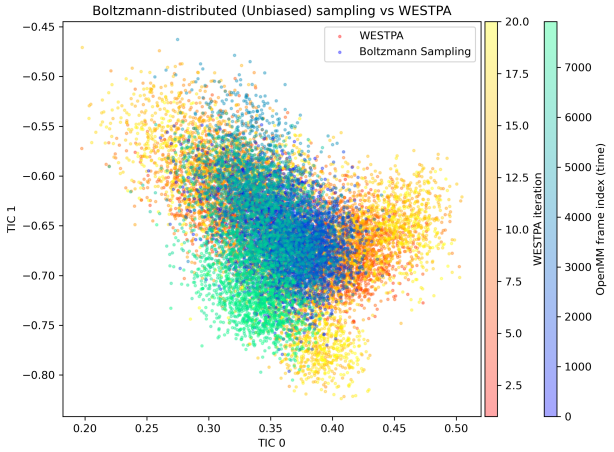

Figure 2.3: Homeodomain Protein in TICA Space. Regular OpenMM (Blue) simulation (Boltzmann Sampling) compared to WESTPA (Yellow) for the same compute. This means that the single OpenMM walker has a longer timescale. However, even with the increased timescale ($3\times$ as long), the explored TICA space is lesser than that of WESTPA. For this small example, WESTPA has greater conformational variance than a regular OpenMM trajectory.

cessors p because

$$C_{\text{total}}(p) \approx \frac{N C_{\text{prop}}}{p} + O(C_{\text{comm}}),$$

with C_{comm} negligible relative to propagation. Thus, WE scales linearly with computational resources.

2.3 Machine Learning Methods for Molecular Dynamics

As machine learning (ML) has grown in its ability, researchers attempt to use ML force fields to accelerate the speed of molecular dynamics since the speed of inference is much faster than computing a timestep in traditional MD. Such ML force fields increasingly rely on graph neural networks (GNNs) and Transformers to represent molecular configurations [18]. A molecular structure with coordinates $\mathbf{r} = (\mathbf{r}_1, \ldots, \mathbf{r}_n)$ is treated as a graph $G = (V, E)$, where nodes correspond to atoms (or coarse-grained beads) and edges connect spatial neighbors, typically defined by a cutoff radius. Each node i carries an embedding vector $\mathbf{x}_i^{(0)}$ derived from its atomic type, and edges (i, j) contain geometric features such as pairwise distances $r_{ij} = \|\mathbf{r}_i - \mathbf{r}_j\|$. GNNs update these

embeddings through a sequence of message passing steps of the form

$$\mathbf{m}_{ij}^{(t)} = \phi_{\text{msg}}\left(\mathbf{x}_i^{(t)}, \mathbf{x}_j^{(t)}, r_{ij}\right), \mathbf{x}_i^{(t+1)} = \phi_{\text{upd}}\left(\mathbf{x}_i^{(t)}, \sum_{j \in \mathcal{N}(i)} \mathbf{m}_{ij}^{(t)}\right),$$

where ϕ_{msg} and ϕ_{upd} are learnable functions. Because the messages depend only on relative coordinates, GNN-based force fields can be constructed to satisfy translation and rotation invariance as well as permutation symmetry among identical atoms.

The message-passing GNN is used in several ML force field architectures. SchNet [19] implements continuous-filter convolutions in which the interaction between atoms is controlled by a learned filter $W_\theta(r_{ij})$ that depends solely on interatomic distance, yielding the message

$$\mathbf{m}_{ij}^{(t)} = W_\theta(r_{ij}) \odot \mathbf{x}_j^{(t)}.$$

Because r_{ij} is invariant under rigid motions, the entire architecture is naturally rotation and translation invariant. TorchMD-Net [20][21] uses $E(3)$-equivariant transformer, allowing both scalar and vector features to transform correctly under rotations. This ensures that applying a rotation R to the coordinates yields

$$\Phi(R\mathbf{r}) = R\,\Phi(\mathbf{r}),$$

guaranteeing that forces and other vector outputs remain consistent with molecular geometry. CGSchNet [22] adapts SchNet to coarse-grained (CG) modeling by operating on a reduced graph in which nodes represent CG beads.

After T message-passing layers, the final embeddings $\mathbf{x}_i^{(T)}$ are mapped to atomic (or bead-wise) energy contributions, so that the total potential energy is modeled as

$$U(\mathbf{r};\mathbf{Q}) = \sum_{i=1}^{n} u_\theta\left(\mathbf{x}_i^{(T)}\right),$$

where $\mathbf{Q}$ contains all learnable parameters. Forces are obtained by differentiation as

$$\mathbf{F}(\mathbf{r}) = -\nabla_{\mathbf{r}} U(\mathbf{r};\mathbf{Q}),$$

making the model directly compatible with MD simulation. Training typically uses force matching. For coarse-grained learning, mapping operators Ξ and Ξ_F project atomistic coordinates and forces onto the CG representation. The corresponding force-matching loss is

$$\mathscr{L}_F(\mathbf{r};\mathbf{Q}) = \frac{1}{3Mn}\sum_{i=1}^{M} \left\| \Xi_F \mathbf{F}(\mathbf{r}_i) + \nabla U(\Xi \mathbf{r}_i, \mathbf{Q}) \right\|^2,$$

which penalizes discrepancies between reference forces and those implied by the learned potential. Models such as SchNet, TorchMD-Net, and CGSchNet minimize this loss to obtain stable ML force fields suitable for molecular dynamics simulations.

Chapter 3

Benchmarking Neural Network Force Fields

This chapter is based off, and provides a brief synopsis of the material from, the manuscript: **A Standardized Benchmark for Machine-Learned Molecular Dynamics using Weighted Ensemble Sampling** Alexander Aghili, Andy Bruce, Daniel Sabo, Sanya Murdeshwar, Kevin Bachelor, Ionut Mistreanu, Ashwin Lokapally, and Razvan Marinescu, *The Journal of Physical Chemistry B*

3.1 Ground-Truth Dataset

Eight fast-folding proteins from Lindorff-Larsen et al. [23] spanning diverse folds are selected as a representative base for biophysical simulations, as shown in Table 3.1.

Protein	**Residues**	**Fold**	**Description**	**PDB**
Chignolin	10	β-hairpin	Tests basic secondary structure formation	1UAO
Trp-cage	20	α-helix	Fast-folding with hydrophobic collapse	1L2Y
BBA	28	$\beta\beta\alpha$	Mixed secondary structure competition	1FME
Protein B	53	3-helix	Helix packing dynamics	1PRB
Protein G	56	α/β	Complex topology formation	1PGA
λ-repressor	224	5-helix	Tests scalability	1LMB
Homeodomain	54	3-helix bundle	DNA-binding domain with stable fold	1ENH
WW domain	37	β-sheet	Challenging β-sheet topology	1E0L

Table 3.1: Protein benchmarks with representative structures.

To generate reference data against which models will be compared, we ran MD simulations from a total of 9919 different starting points provided by Majewski et al. [24] (Prepared at 350K). From each starting point, we ran 1,000,000 steps at a 4 femtosecond (fs) timestep resulting in a total of 4 nanosecond (ns) per starting point at 300K. Since the starting points provide a detailed coverage of the conformation space, the simulation length of 1,000,000 steps per starting point was selected to ensure adequate sampling of dynamics across the conformational space for these well-characterized proteins.

3.2 Benchmarking System

An overview of the benchmark suite is given in Fig. 3.1. We first pre-process protein conformations, and then run a WESTPA-based weighted ensemble (WE) simulation [25], which is an open-source WE software package. This requires defining the progress coordinate and propagating walkers. We then compare the WE with ground truth data along several dimensions covering both global properties, such as Time-lagged Independent Component Analysis (TICA) [26] energy landscapes, contact map differences and distributions for the radius of gyration (RoG), bond lengths, angles and dihedrals. Finally, we also compute quantitative divergence metrics, mainly Wasserstein-1 and Kullback-Leibler divergences across all the relevant analyses.

3.3 Metrics

Nineteen metrics and plots are used to benchmark a molecular system, which capture both global slow, collective motions and kinetic movement. Time-lagged Independent Component Analysis (TICA) is used to identify the slowest motions in a protein by reducing high-dimensional structural data into a few ordered components. After learning the projection U from ground-truth data, we apply it to both ground-truth and model conformations using

$$z^\top(t) = r^\top(t)U = r^\top(t)W\Sigma^{-1}V.$$

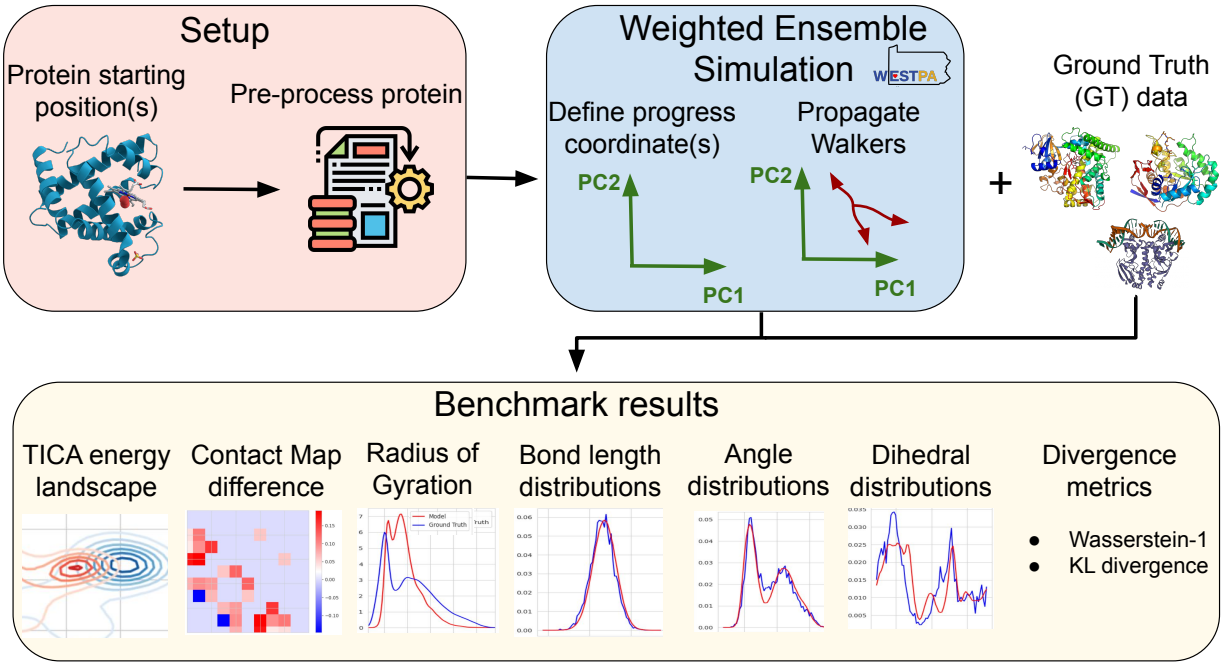

Figure 3.1: Architecture of the weighted ensemble (WE) benchmarking framework.

We then examine 2D plots of the first four TICA components and estimate a weighted KDE in the TICA 0/1 space, computed as

$$p(x) = \sum_i w_i K_h(x - x_i),$$

to compare how well each trajectory samples slow conformational changes. To quantify these differences, we compute the KL divergence,

$$D_{\mathrm{KL}}(P \parallel Q) = \int p(x) \log\left(\frac{p(x)}{q(x)}\right) dx,$$

and the Wasserstein-1 distance,

$$W(P,Q) = \inf_{\gamma \in \Gamma(P,Q)} \int |x - y| \, d\gamma(x,y),$$

which together indicate how similar the two sampled distributions are. Structural deviations are also assessed through contact-map differences, where for each residue pair

$$C_{ij} = \langle d_{ij} \rangle_M - \langle d_{ij} \rangle_{\mathrm{GT}},$$

so positive values mean residues are farther apart in the model and negative values mean they are closer. Global compaction is measured using the radius of gyration,

$$R_g = \sqrt{\frac{1}{N} \sum_{i=1}^{N} \|\mathbf{r}_i - \mathbf{r}_{\mathrm{com}}\|^2},$$

with weighted histograms showing whether the model captures realistic compact and extended states. Finally, local geometry is checked through bond lengths, angles, and dihedrals, where bond lengths are computed as

$$d_{ij} = \|\mathbf{r}_i - \mathbf{r}_j\|,$$

and deviations from expected distributions indicate violations of basic chemical structure.

Chapter 4

Weighted Ensemble Data Generation

4.1 Model Theory - Weighted Ensemble Data

Weighted-ensemble (WE) simulations enable the exploration of a broader conformational landscape within reduced computational time, albeit at the cost of generating raw trajectories that do not follow Boltzmann statistics. Using WE data to sample free-energy space more rapidly than conventional Boltzmann-weighted methods offers several advantages for training machine-learning models: it provides greater diversity of configurations, increases representation of slow conformational transitions, precisely the features the model is intended to learn, and reduces the computational resources required for data generation. However, because WE sampling departs from equilibrium distributions, using such data can introduce biases, preventing the model from learning physically accurate transition dynamics and potentially leading to instability or poor generalization. Consequently, to employ weighted-ensemble data effectively for model training, the training process should be modified to account for the statistical structure of WE sampling.

The force-matching loss function from CGSchNet, assuming a Boltzmann distribution from the training data, is defined as

$$\mathscr{L}_F(\mathbf{r};\mathbf{Q}) = \frac{1}{3Mn}\sum_{i=1}^{M} ||\Xi_F F(\mathbf{r}_i) + \nabla U(\Xi\mathbf{r}_i,\mathbf{Q})||^2$$

with M frames, n Cα atoms, the coarse-graining function Ξ, and the force-mapping function Ξ_F. Adjusting for the weighting, the updated loss function comes to

$$\mathscr{L}_F^{WE}(\mathbf{r};\mathbf{Q}) = \frac{1}{3n}\sum_{i=1}^{M} \hat{w}_i ||\Xi_F F(\mathbf{r}_i) + \nabla U(\Xi \mathbf{r}_i, \mathbf{Q})||^2, \hat{w}_i = \frac{w_i}{\sum_j w_j}$$

Proof. Consider the atomistic configuration $\mathbf{r} \in \mathbb{R}^{3N}$ mapped to the coarse-grained coordinates $\Xi\mathbf{r} \in \mathbb{R}^{3n}$ using the linear coarse-grained mapping operator Ξ. The corresponding mapped forces are $\Xi_F\mathbf{F}(\mathbf{r}) \in \mathbb{R}^{3n}$. For a coarse-grained potential $U(\Xi\mathbf{r};\mathbf{Q})$ with parameters $\mathbf{Q}$, the force matching loss is defined as the mean-squared deviation between the mapped atomistic forces and the coarse-grained forces:

$$\mathscr{L}_F(\mathbf{Q}) = \frac{1}{3n}\mathbb{E}_{r\sim\mu}||\Xi_F F(\mathbf{r}) + \nabla U(\Xi\mathbf{r},\mathbf{Q})||^2 \tag{4.1}$$

where μ is the probability distribution of atomistic configurations under the canonical ensemble (NVT). If unbiased samples $\mathbf{r}_i \sim \mu$ were available, then (4.1) would be approximated by the empirical average

$$\mathscr{L}_F(\mathbf{Q}) \approx \frac{1}{3Mn}\sum_{i=1}^{M} g(\mathbf{r}_i), g(\mathbf{r}) = ||\Xi_F F(\mathbf{r}) + \nabla U(\Xi\mathbf{r},\mathbf{Q})||^2$$

However, when using weighted ensembles, the samples are biased thus $\mathbf{r_i} \not\sim \mu$ but instead $\mathbf{r_i} \sim \nu$. The following theorems show how to alter the loss function to approximate Boltzmann sampling.

Theorem 1. *Importance Sampling Theorem*

Let $(\Omega, \mathscr{F})$ be a measurable space with μ be the target probability measure on Ω, ν be a proposal measure on Ω such that μ is absolutely continuous with respect to ν ($\mu \ll \nu$). Then by the Radon-Nikodym theorem, there exists a measurable function $w : \Omega \to [0,\infty)$ such that

$$\frac{d\mu}{d\nu}(x) = w(x)$$

For every integrable measurable function g,

$$\mathbb{E}_\mu[g(X)] = \int_\Omega g(x)\mu(dx)$$

Plugging in the Radon-Nikodym expression for $\mu(dx)$,

$$\int_\Omega g(x)\mu(dx) = \int_\Omega g(x)w(x)\nu(dx)$$

Therefore,

$$\mathbb{E}_\mu[g(X)] = \mathbb{E}_\nu[w(X)g(X)]$$

Corollary 1.1. *Self-Normalized Importance Sampling*

Let $X_1, \cdots, X_M$ be independent and identically distributed from ν. Define the unnormalized weights

$$\tilde{w} = w(X_i)$$

and normalized weights

$$\hat{w}_i = \frac{\tilde{w}_i}{\sum_{j=1}^M \tilde{w}_j}$$

Consider the self-normalized importance sampling estimator

$$\tilde{I}_M = \sum_{i=1}^M \hat{w}_i g(X_i)$$

Thus, as $M \to \infty$, $\tilde{I}_M \to \mathbb{E}_\mu[g(X)]$. This means as the samples increase to infinty, the bias goes to zero, and in practice is a sufficient approximator.

Corollary 1.2. *Weighted Ensemble Encoding The Weighted Ensemble method constructs a random particle system*

$$\{(X_i^t, W_i^t)\}_{i=1}^{N_t}$$

designed to approximate the true distribution μ_t of a stochastic process X_t. The associated empir-

ical measure is

$$\hat{\mu}_t^N = \sum_{i=1}^{N_t} W_i^t \delta_{X_i^t}$$

Theorem 2. *Unbiased Property For every bounded measurable function g,*

$$\mathbb{E}\left[\sum_{i=1}^{N_t} W_i^t g(X_i^t)\right] = \int g(x)\mu_t(dx)$$

This holds $\forall t$ and for any binning, even if the bins change dynamically [27].

Let ν be the sampling distribution implicitly induced by the WE genealogical process and binning. Further, let μ_t be the true underlying process distribution at time t (Boltzmann). Then, Theorem 2 states

$$W_i^t \propto \frac{d\mu_t}{d\nu}(X_i^t)$$

Any observable g satisfies

$$\mathbb{E}_\mu[g] = \mathbb{E}_{WE}\left[\sum_i W_i g(X_i)\right]$$

Thus, substituting $g(r)$ yields the WE estimator of the force-matching loss

$$\mathscr{L}_F^{WE}(\mathbf{Q}) = \frac{1}{3n}\sum_{i=1}^{M} \hat{w}_i ||\Xi_F F(\mathbf{r}_i) + \nabla U(\Xi \mathbf{r}_i, \mathbf{Q})||^2, \hat{w}_i = \frac{w_i}{\sum_j w_j}$$

□

4.2 Data Generation

All simulation data was generated using OpenMM for molecular dynamics propagation and WESTPA as the weighted ensemble toolkit. Each system began from a single all-atom conformation in explicit solvent. After every segment, WESTPA used a 2D TICA (TIC 0 and TIC 1) progress coordinate to adaptively redistribute trajectories. Trajectories were continued by restoring saved OpenMM states, while new ones were initialized from the starting structure. All configurations

and observables required for analysis were recorded during propagation, and the entire workflow was parallelized over GPUs using WESTPA's MPI work manager.

Table 4.1: Simulation parameters for data generation runs (explicit all-atom) with WESTPA

Quantity	**Value**
Simulation engine	OpenMM
Force field	AMBER14 all-atom (amber14-all.xml)
Water model	TIP3P-FB (amber14/tip3pfb.xml)
Solvent model	Explicit solvent (periodic box)
Temperature	300 K
Pressure	1 atm
Thermostat / integrator	Langevin Middle Integrator
Friction coefficient	1 ps^{-1}
MD timestep	4 fs
Hydrogen mass	1.5 amu (mass repartitioning)
Bond constraints	Standard bonds to H (OpenMM HBonds)
Constraint tolerance	10^{-6}
Barostat interval	25 steps
Progress-coordinate type	2D TICA (components 0 and 1)
Progress-coordinate input	Cα Cartesian coordinates (in Å)
Bin mapping scheme	Minimal adaptive binning (MAB)
Number of WE bins	9×9 (2D grid)
Target trajectories per bin	6
Steps per iteration (segment length)	1000 MD steps (4 ps)
Trajectory save interval	Every 100 steps (0.4 ps)
Number of nodes	4
MPI ranks per node	4 (16 total ranks)
Walltime limit	36 hours
WESTPA work manager	MPI

After completing the runs, the original data has the characteristics

Protein	Iterations	Total segments	Max timescale (ns)[1]	Estimated Size (TB)[2]
BBA	2665	886699	10.66	3.78
Chignolin	2593	972281	10.37	1.99
Homeodomain	3280	774246	13.12	5.41
λ-repressor	2399	638106	9.60	10.29
Protein B	3327	788771	13.31	5.30
Protein G	3094	766921	12.38	6.00
Trp-cage	2636	908161	10.54	2.69
WWdomain	2493	868669	9.97	4.14

Table 4.2: Simulation statistics for benchmark proteins of training data.
[1] Maximum timescale is a measure of the how much time has passed from the starting point at the final iteration. At 4 fs per step and 1,000 steps per iteration, each iteration corresponds to 4 picoseconds (0.004 ns). The maximum timescale is found by taking the total iterations for a protein and multiplying by 4 ps. Note that this is not a metric of compute required, since there maybe hundreds of segments per iteration that are parallel trajectories in time.
[2] Size estimated by taking 75 random segment directories, calculating the average size, and multiplying by total segments.

4.3 Data Pipeline and Implementation Details

After WESTPA completes, its output is organized into a hierarchical structure in which each iteration contains multiple walker segments, and each segment stores a short MD trajectory together with auxiliary force data. A global HDF5 file records walker weights and iteration metadata. WESTPA's segmentation of configuration space is ideal for enhanced sampling but produces data that is fragmented. To make this data suitable for machine-learning–based coarse-grained (CG) modeling, we first convert it into a unified dataset that aggregates all segments into a single continuous coordinate and force array while preserving the WESTPA weights.

The conversion process reads the base topology, restricts it to protein atoms, and walks through the WESTPA run to identify all segments whose statistical weight exceeds a minimum threshold. Removing the low-probability segments maintains the important dynamics while decreasing the size and compute time for the conversion process. Each segment's trajectory is streamed into a single HDF5 file along with time metadata. In a second pass, the segment-level force arrays are read and appended to a unified forces dataset, and each frame is assigned the walker's weight. The resulting HDF5 file thus contains arrays

$$\mathbf{R}(t) \in \mathbb{R}^{3\times N_{\text{frames}}\times N_{\text{atoms}}}, \mathbf{F}(t) \in \mathbb{R}^{3\times N_{\text{frames}}\times N_{\text{atoms}}}, w(t) \in \mathbb{R}^{N_{\text{frames}}}$$

representing the protein-only coordinates, all-atom forces, and the WESTPA weights with N_{frames} be the post-filtering frames. Preprocessing then transforms this all-atom weighted dataset into coarse-grained inputs for training a CG force field. The mapped coordinates for frame t are computed as bead-center positions

$$\mathbf{r}_b(t) = \frac{1}{\sum_{a\in b} m_a} \sum_{a\in b} m_a \mathbf{R}_a(t)$$

Similarly, forces are mapped by projecting all-atom forces onto the coarse beads as a mass-

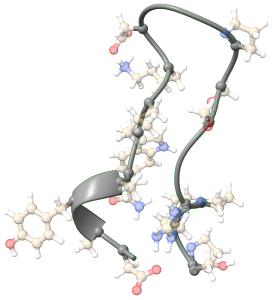

Figure 4.1: Trp-cage protein with $C\alpha$ backbone outlined in gray, with beads displayed as balls. The corresponding all-atom system is shown with some opacity. The mapping of the all-atom system to the $C\alpha$ backbone is coarse-graining.

weighted sum of the atomic forces within each bead

$$\mathbf{F}_b(t) = \frac{1}{\sum_{a\in b} m_a} \sum_{a\in b} m_a \mathbf{F}_a(t)$$

Many coarse-grained (CG) models incorporate a prior force field consisting of bonded, angular, dihedral, and repulsive terms. The preprocessing pipeline can either load a pre-defined prior or fit one directly from data. When fitting is required, structural statistics such as bond-length and dihedral distributions are accumulated across all systems. In the simplest case, a bonded prior might assume harmonic potentials,

$$U_{\text{bond}}(r) = k_r(r - r_{eq})^2,$$

while dihedral priors may use periodic expansions,

$$U_\phi(\phi) = \sum_{n=1}^{m} k_\phi \cos(n\phi - \gamma),$$

or more flexible representations depending on the selected prior. The fitting routines infer param-

eter sets $\{k_r, r_{eq}\}$, $\{k_\phi, \gamma\}$, etc., at a target temperature using the empirical structural distributions extracted from the mapped trajectories. Implementation-wise, cached histograms from Step 1 are merged, and maximum-likelihood or regularized optimization is performed to obtain the parameters.

The final stage of preprocessing computes the delta forces, which form the supervised learning targets for the machine-learning model. For each frame, the prior force field is evaluated on the CG coordinates, producing a predicted baseline force,

$$\mathbf{F}_{\text{prior}}(t).$$

The delta force is then defined as

$$\Delta\mathbf{F}(t) = \mathbf{F}_{\text{CG,MD}}(t) - \mathbf{F}_{\text{prior}}(t),$$

representing the component of the true force not captured by the analytic prior. These delta forces are stored alongside the CG coordinates, embeddings, and prior parameters, forming a complete coarse-grained dataset suitable for the subsequent training stage. At this point, preprocessing provides (1) CG geometries and embeddings, (2) mapped MD forces, (3) a fitted prior force field, and (4) delta-force learning targets, enabling construction of machine-learned CG potentials in the next step of the workflow.

The model is instantiated from a configurable architecture (TorchMD-Net) and optimized using AdamW with force losses. Training uses mini-batches over frames and proteins, optional learning-rate schedulers, and early stopping based on validation loss. To assess the trained model, we embed it into a new WESTPA run as the propagator for the coarse-grained dynamics, replacing the original all-atom MD engine. At each time step, the CG coordinates are advanced under the sum of prior forces and learned delta forces. This procedure is run through the weighted benchmark as described in Chapter 3. The result is the output diagrams and metrics to determine the success of the model.

4.4 Results

The experiments for all proteins were conducted under a unified set of hyperparameters, summarized in Table 4.3, with the only distinction between weighted and unweighted models being the inclusion or exclusion of force weights.

Table 4.3: Training and Data Pipeline Hyperparameters

Category	**Hyperparameter**
Epochs	50
Batch size	20
Configuration file	`config_cutoff2_seq6.yaml`
Weight cutoff (filtering)	1×10^{-5}
Coarse–graining	C_α
Preprocessing	No box
Temperature	300 K
APC	60,000
Prior	`CA_Majewski2022_v0`
Weighted model	Includes force weights
Unweighted model	Excludes force weights

We collect the KL-Divergence values for all the benchmark proteins in Table 4.4 and notable benchmark results.

Protein	**TIC 0**		**TIC 1**		**Bonds**		**Angles**		**Dihedrals**	
	W	UW	W	UW	W	UW	W	UW	W	UW
BBA	**0.109**	0.218	1.553	**1.470**	**0.005**	0.030	**0.169**	0.393	0.177	**0.167**
Chignolin	**1.401**	2.600	**0.227**	0.314	**0.006**	0.011	0.073	**0.050**	0.074	**0.058**
Homeodomain	2.490	**2.072**	**2.596**	2.651	**0.002**	0.083	**0.139**	0.306	**0.102**	0.218
λ-repressor	**1.790**	4.673	**2.657**	3.735	**0.001**	6.688	**0.059**	0.1668	**0.025**	0.082
Protein B	**6.540**	7.065	**0.613**	0.726	0.047	**0.039**	0.414	**0.328**	0.223	**0.186**
Protein G	**3.711**	9.614	**2.083**	9.119	**0.037**	12.451	**0.231**	0.745	**0.048**	0.270
Trp-cage	**4.281**	4.305	**0.926**	0.994	**0.001**	0.006	0.250	**0.211**	**0.159**	0.170
WW domain	4.182	**3.920**	1.564	**1.238**	**0.004**	0.008	**0.213**	0.243	0.041	**0.038**

Table 4.4: [KL-Divergence Results Table]KL-Divergence – $D_{\mathrm{KL}}(\mathrm{GT} \parallel \mathrm{Model})$ – for the weighted (W) and the unweighted (UW) model PDFs across TIC 0, TIC 1, and structural features: bond lengths, angles and dihedrals for all proteins in our benchmark.

Trajectories are visualized in ChimeraX to see how the physical structure progresses over time. A subset is shown with Homeodomain, λ-repressor, and Protein G.

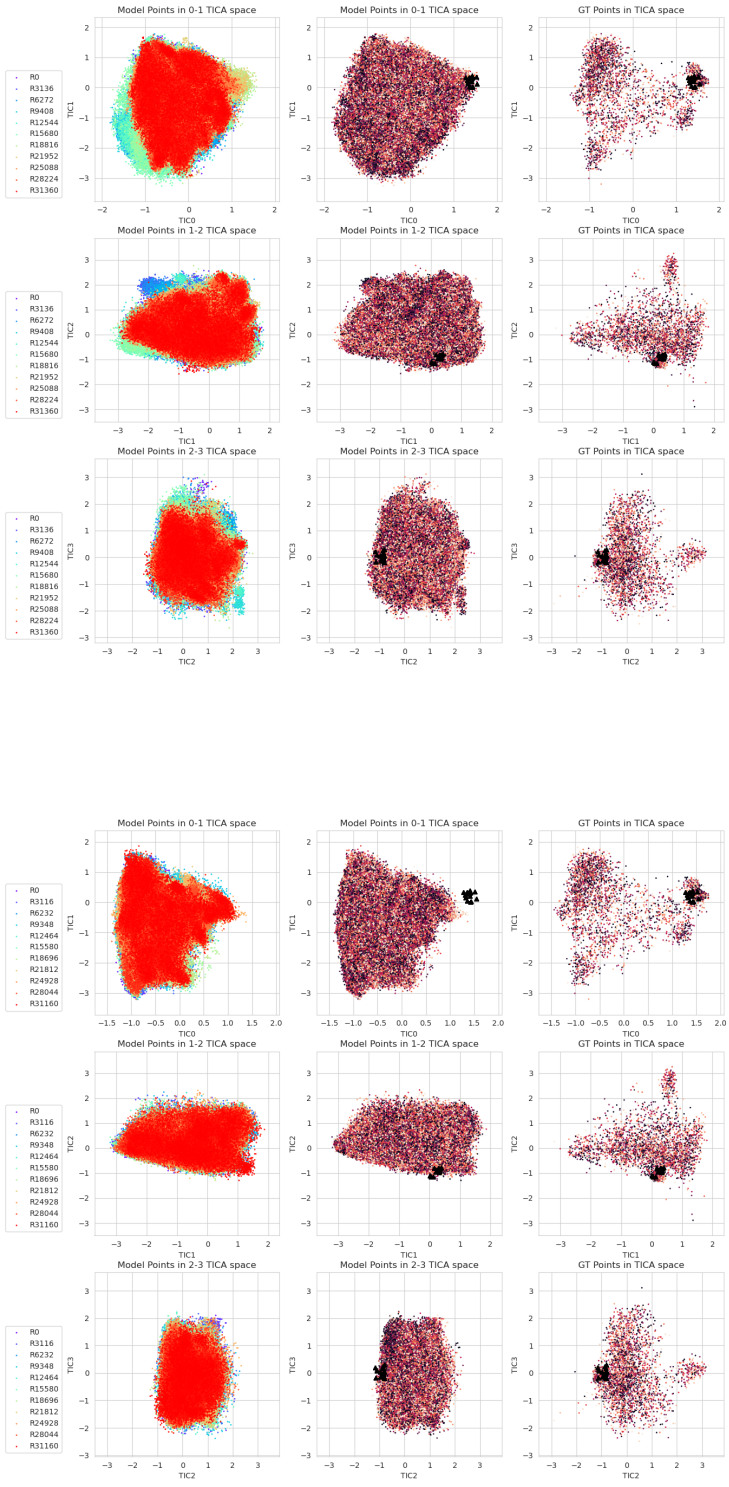

Figure 4.2: This comparison of the explored TICA space between the Unweighted (bottom) and Weighted (top) model for Chignolin shows how a weighted model can explore more of the free-energy space and be a closer match to the ground truth dataset.

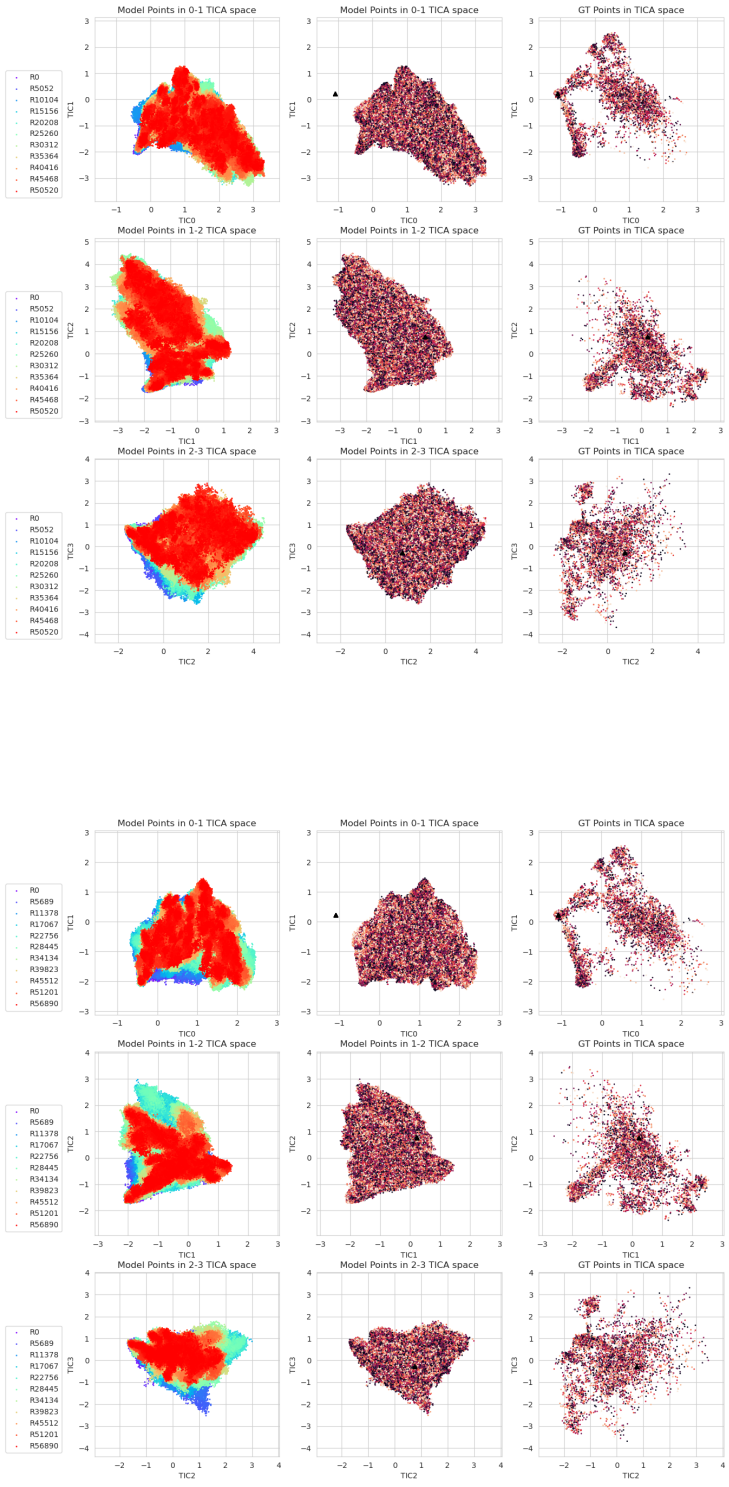

Figure 4.3: As in Figure 4.2, we have the explored TICA space between the Unweighted (bottom) and Weighted (top) model for WWdomain. However, as shown, the Unweighted model is able to explore more of the TICA space.

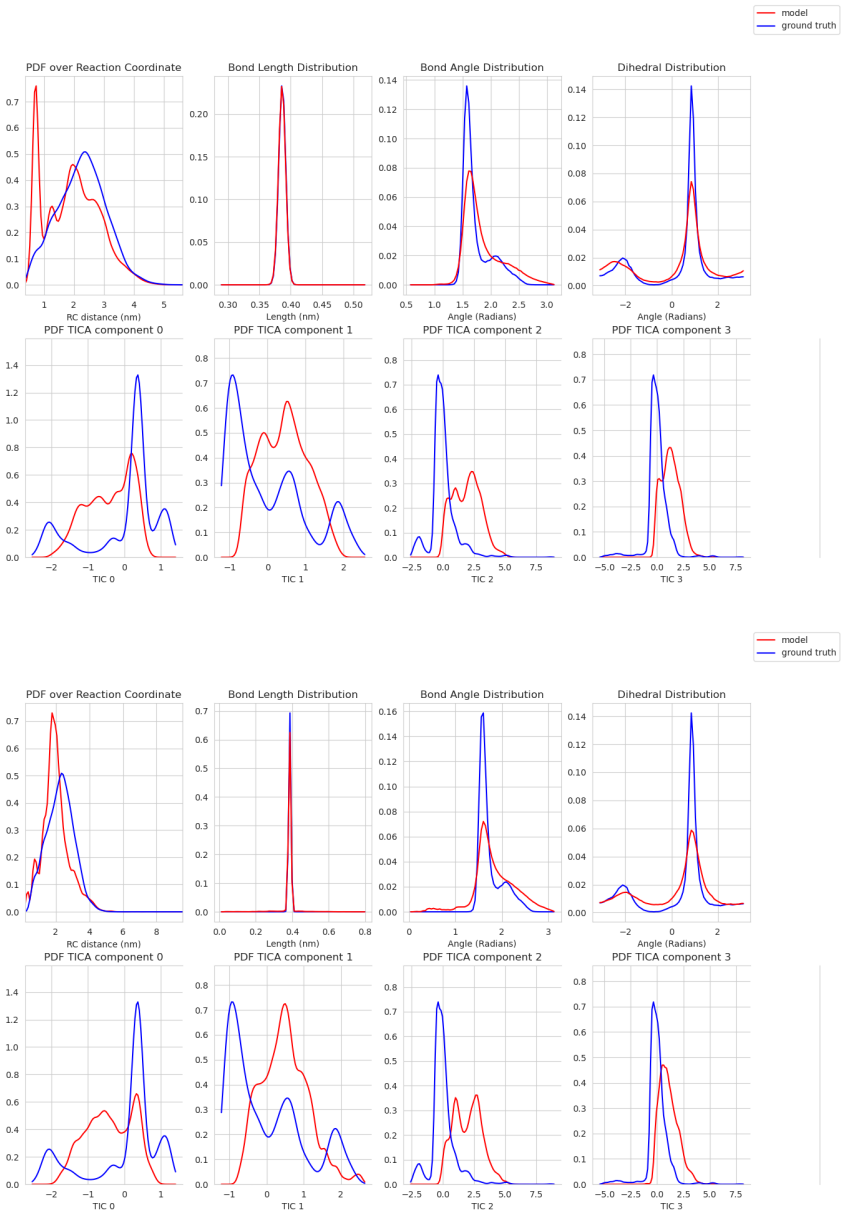

Figure 4.4: Comparing the Unweighted (bottom) and Weighted (top) model PDF results for Homeodomain, both models are fairly close in their results, with the TIC PDFs and BAD metrics all aligning similarly. Many models shared similar results, through further investigation into model structure revealed subtleties in model behavior when at extended timescales, namely collapse of models is not captured in the benchmark.

Figure 4.5: A close up of a single trajectory of Homeodomain, with the original structure (left), the weighted model (middle), and the unweighted model (right). The weighted model maintains geometric structure like the α-helix, and overall geometric accuracy. The unweighted model has lost geometric structures like the α-helix's and is in an unnatural state.

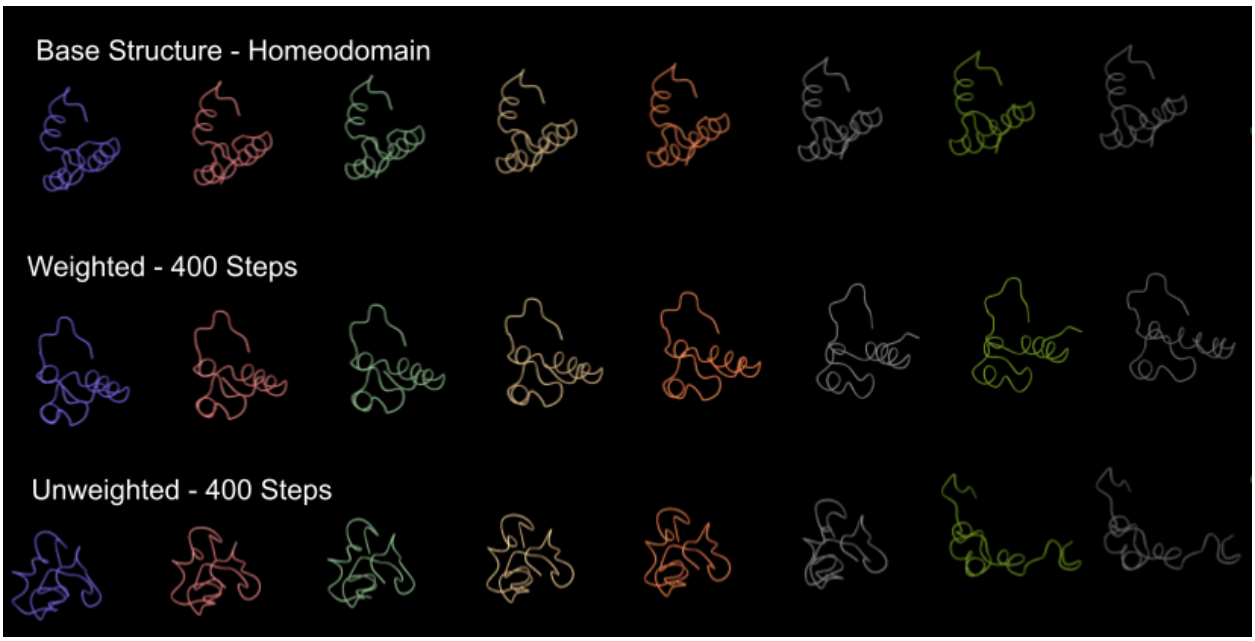

Figure 4.6: A sample of seven trajectories of Homeodomain, with the base structure (top) and weighted (middle) and unweighted (bottom) model after 400 steps for each trajectory.

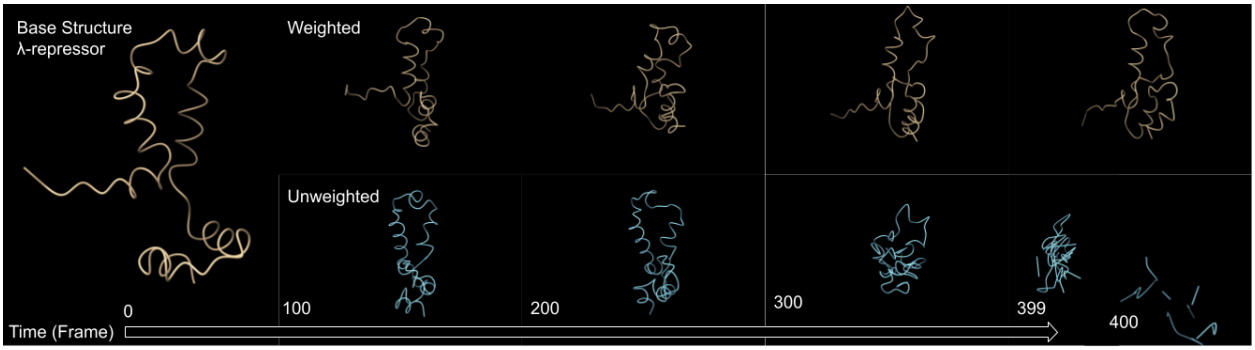

Figure 4.7: λ-repressor structure changing over time from the base structure (in intervals of 100 steps) with both the unweighted and weighted models. The weighted model maintains stability through 400 steps, while the unweighted model begins non-physical behavior at around 300 steps and fully collapses at 400 steps.

Figure 4.8: Protein G structure with the base structure (left), the weighted model (middle), and the unweighted model(right). The weighted model loses geometric structures such as the α-helix at the bottom of the base structure, entering a physically unrealistic state. The unweighted model (right) has completely collapsed into a non-physical state.

4.5 Discussion

The results presented here indicate the potential for using weighted ensemble sampling as a source of training data for coarse-grained neural network models in molecular dynamics. Across multiple figures, models trained with weighted trajectories exhibit improved long-term stability and more accurate conformational behavior. For instance, Figure 4.7 demonstrates that the unweighted model ultimately loses stability and collapses after approximately 400 steps, whereas the weighted model preserves both stability and relevant structural features over the same duration.

Despite this improvement, the weighted model is not without limitations. As illustrated in Figure 4.8, the weighted dataset can still exhibit structural failures: the model intermittently loses core stability, with geometric structure such as the α-helix structure deteriorating and transitioning into non-physical conformations. Even in these failure cases, the weighted model outperforms the unweighted counterpart, which undergoes complete structural collapse under equivalent conditions. The complete collapse of the unweighted model is likely due to the model being unable to compensate for non-Boltzmann behavior in the weights that is not accounted for. This suggests that weighted ensemble sampling provides a meaningful advantage, though additional methodological refinement is required to maintain geometric fidelity throughout extended simulations.

It is likely that weighted ensemble–based models, with further methodological and algorithmic enhancements, have the potential to serve as substantially more efficient sources of training data for machine-learning–driven molecular dynamics than traditional Boltzmann sampling approaches.

However, Table 4.4 indicates that the weighted dataset does not universally outperform the unweighted dataset, with performance faltering on key metrics like Dihedral consistency and TIC 1 matching. Continued investigation into the conditions under which weighting yields the greatest benefit is needed to understand how and why model performance varies across proteins.

4.6 Future Work

There remains much additional work to do. Future work should focus on expanding the experimental validation of WE-trained CG models to more thoroughly assess dynamical fidelity and physical realism. While the present study demonstrates improvements in structural observables and TICA-space consistency, additional benchmarks such as folding/unfolding kinetics, transition-path statistics, and long-timescale stability tests are needed to confirm that reweighted force matching yields genuinely improved CG dynamics.

Increasing the diversity of training data by incorporating WE simulations initiated from multiple initial starting points could reduce the required time to explore vastly more complex systems than the ones tested here. WE methods are particularly effective at sampling rare events, and using multi-starting point WE ensembles could help the model learn globally accurate CG potentials, as opposed to localized regions, that remain valid across folded, unfolded, and intermediate conformations. This would introduce challenges in maintaining a valid weighting scheme, likely requiring a more complex solution than provided here. Ablation studies including varying weight cutoffs, examining the contribution of weighted priors, and testing alternative model architectures will clarify which components of the pipeline most strongly benefit from importance-weighted training and which may introduce unnecessary complexity or bias.

Finally, this work only trained and evaluate models trained on a single protein. Evaluating out-of-domain generalization and multi-protein training is key for further development and usefulness of these models. Constructing composite datasets spanning many protein topologies will allow testing whether weighted-ensemble data improves transferability and reduces overfitting to specific systems. Developing a unified CG potential trained on WE data from diverse proteins, and embedding it into adaptive WE–CG simulation loops, would represent a significant step toward automated, physics-informed coarse-graining that can generalize across biomolecular systems.

Chapter 5

Optimizations

There were several opportunities for optimization throughout the process. The improvements that were implemented and potential future enhancements are outlined below.

5.1 Distributed Training

One easy win was transferring the existing training script from only working on a single node to working multi-node with PyTorch Distributed Data Parallel (DDP) [28][29].

The transition required restructuring the training program so that multiple independent Python processes could cooperate to train a single shared model. Originally, all computation occurred inside one process that optionally used `nn.DataParallel` to replicate work across GPUs within a single node. This approach had two limitations: it could not scale across nodes, and it created unnecessary synchronization overhead because the main process became a bottleneck for gradient communication. To overcome this, I implemented a proper distributed initialization step, which determines each process's global rank, world size, and local GPU index, and then calls `torch.distributed.init_process_group` so that all processes can communicate directly.

Each process then receives a unique shard of the dataset. This was accomplished by adding `DistributedSampler` objects for both training and validation datasets and by calling

`sampler.set_epoch(epoch)` each epoch to ensure deterministic shuffling across processes. Each process now constructs its own `DataLoader`, but together these loaders cover the entire dataset without overlapping samples.

The model is wrapped with `DistributedDataParallel` (DDP), with each process holding a full model replica located on a single GPU. DDP automatically synchronizes gradients via `all_reduce` operations during backpropagation, ensuring that all replicas remain in sync with minimal overhead. Because each process computes metrics on only its shard of the data, `all_reduce` operations are required to sum losses and normalization counts. Early stopping is evaluated only on rank 0 and broadcast to all other processes so that the entire job exits consistently.

To validate the performance, the time to complete the same model training was tested across 1 to 6 nodes. Indeed, the scaling behavior is linear with the number of nodes, except moving from one node to many which can be explained by reduced load on a single nodes GPUs. There is an expected point at which GPUs are no longer saturated and the overhead of communication will prevent further efficient horizontal scaling.

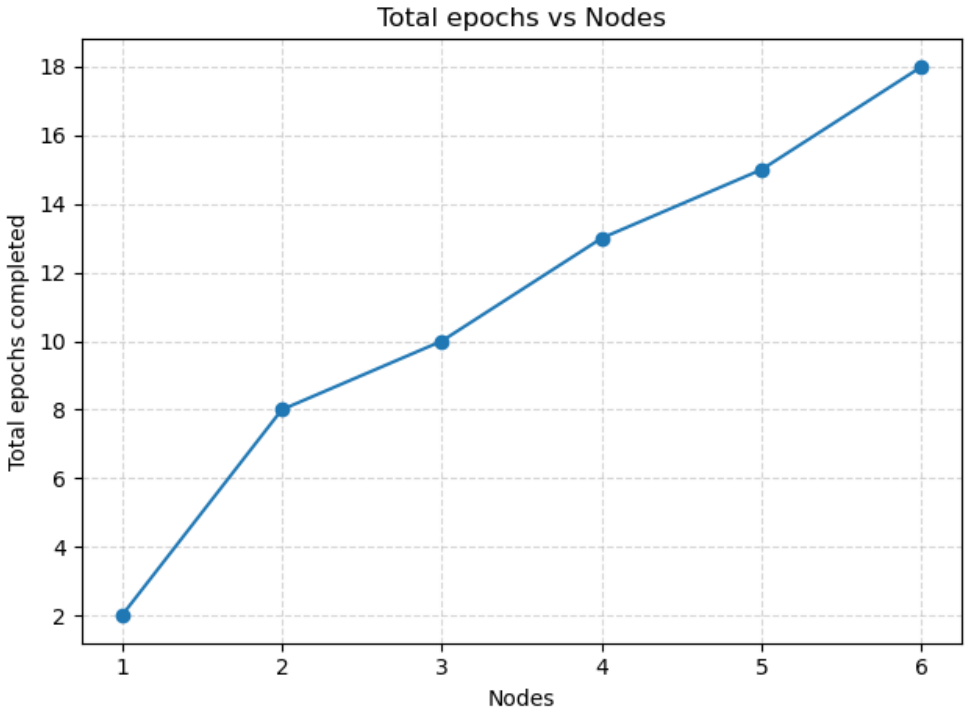

Figure 5.1: Number of total epochs evaluated in 2 hours with different amounts of nodes using PyTorch DDP.

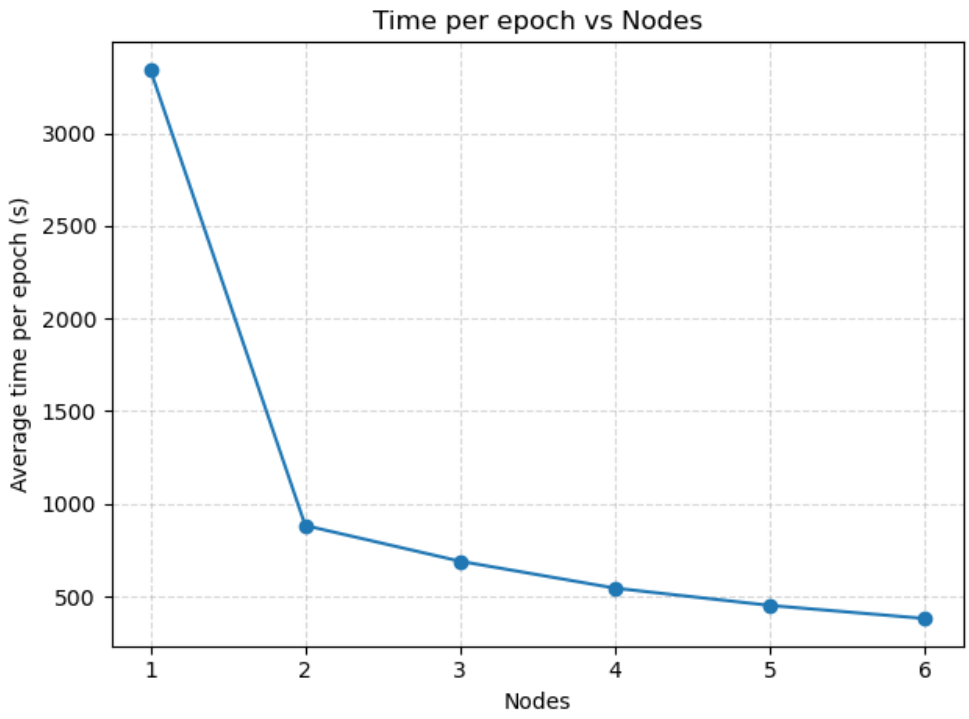

Figure 5.2: The average time per epoch over 2 hours evaluated with different amounts of nodes using PyTorch DDP.

5.2 Trajectory Packing

Trajectory packing is a method to run parallel trajectories with TorchMD-Net simultaneously in a single GPU call, thereby reducing the overhead and increasing the GPU utilization when performing forward propagation of a TorchMD-Net system.

In the `CGMLPropagator` implementation, this is achieved by constructing a TorchMD system with an explicit replica dimension and assigning each WESTPA segment in the current work batch to one of these replicas. The number of available replicas is configured via the WEST configuration (the `block_size` parameter), and the propagator asserts that the number of active segments does not exceed this capacity. Instead of creating a separate TorchMD system for every segment, all segment configurations (coordinates and velocities) are written into a single batched array, where the leading dimension indexes the replica / segment.

For each segment, the initial state is prepared depending on whether the segment continues from a previous iteration or starts a new trajectory. For continuing segments, the propagator loads the last saved frame from the parent segment's trajectory file and copies those coordinates and velocities into the appropriate replica slot. For new trajectories, it resets the coordinates to the reference coarse-grained structure and samples fresh velocities from a Maxwell–Boltzmann distribution at the target temperature. A simple mapping from segment index to replica index ensures that each segment occupies a fixed slice along the replica dimension throughout the propagation.

Once all replicas are initialized, integration proceeds in a fully batched fashion. The propagator calls the TorchMD integrator, which advances all replicas simultaneously for a block of MD steps on the GPU. Batched tensors are appended to in-memory lists, thereby storing the packed trajectories as a stack of frames along the time dimension and replicas along the batch dimension.

The `block_size` parameter was varied in increments of 2^n with $n = \{0, 1, \cdots, 7\} \cup \{10\}$ to test limited trajectory packing, sufficient trajectory packing, and over-utilization states. Each ran Chignolin for 30 iterations with a 9×9 MAB Binning, targeting 6 walkers per bin.

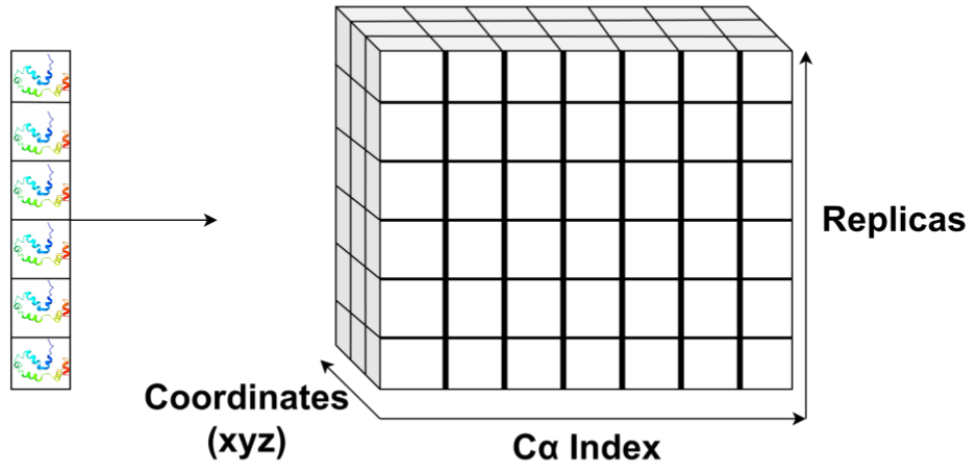

Figure 5.3: (Left) An array of the same protein (λ-repressor) which is equivalent to (Right) a tensor with the replicas as the vertical component, $C\alpha$ on the horizontal component, and the coordinates of the $C\alpha$ in the depth component. This tensor is sent to the GPU and evaluated in parallel, allowing multiple forward passes through the network to be evaluated simultaneously.

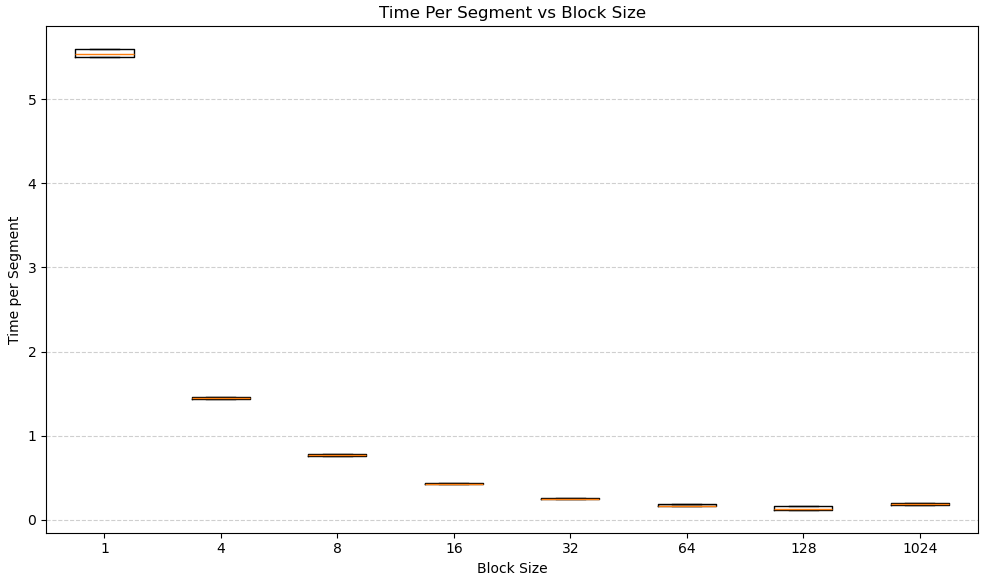

Figure 5.4: Time to compute a segment over block_size(for multi-node, this is the time to complete n simultaneous segments divided by n, where n is the block_size). The 25th, 50th, and 75th percentile are shown. A clear trend of greater block_size increases speed.

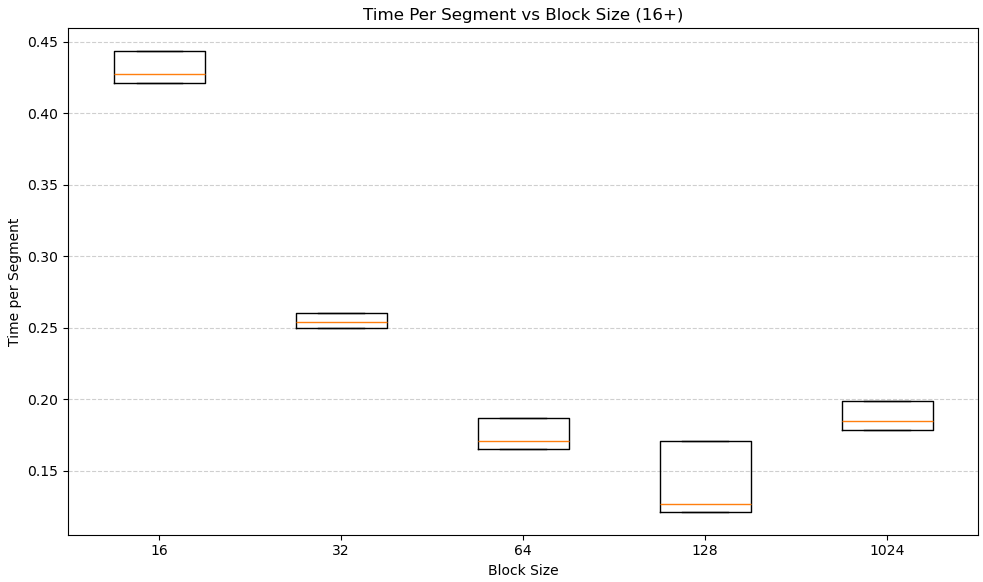

Figure 5.5: Time to compute a segment over block_size (Restricted to a block_size of 16 and greater). The 25th, 50th, and 75th percentile are shown. While greater block_size decreases compute, this ends between 128 and 1024.

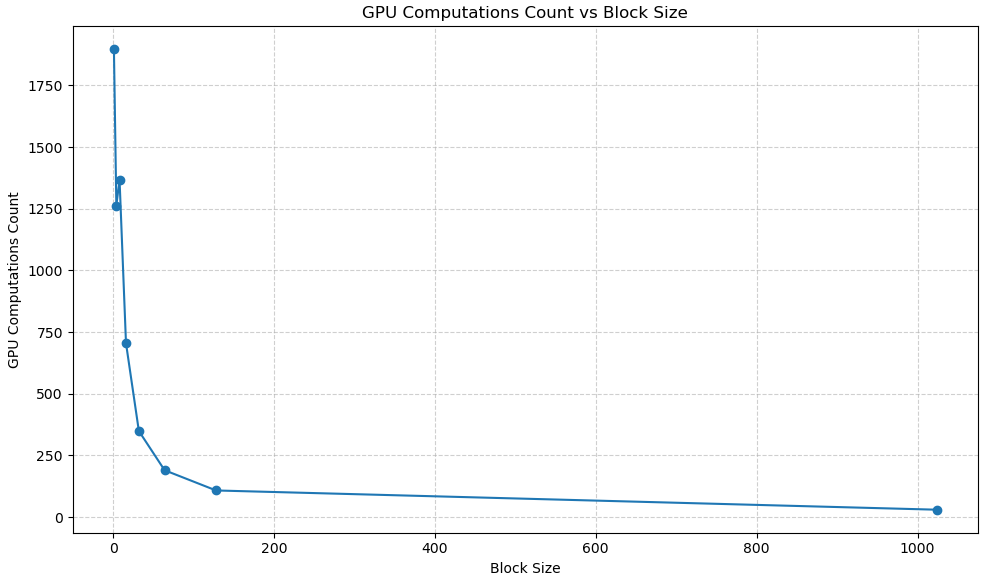

Figure 5.6: The amount of independent GPU computations required to complete 30 iterations. At a block_size of 1, the GPU computation count is equivalent to the number of segments, since each segment is processed independently. At the other extreme, a block_size of 1024 means that every iteration fits on a single GPU.

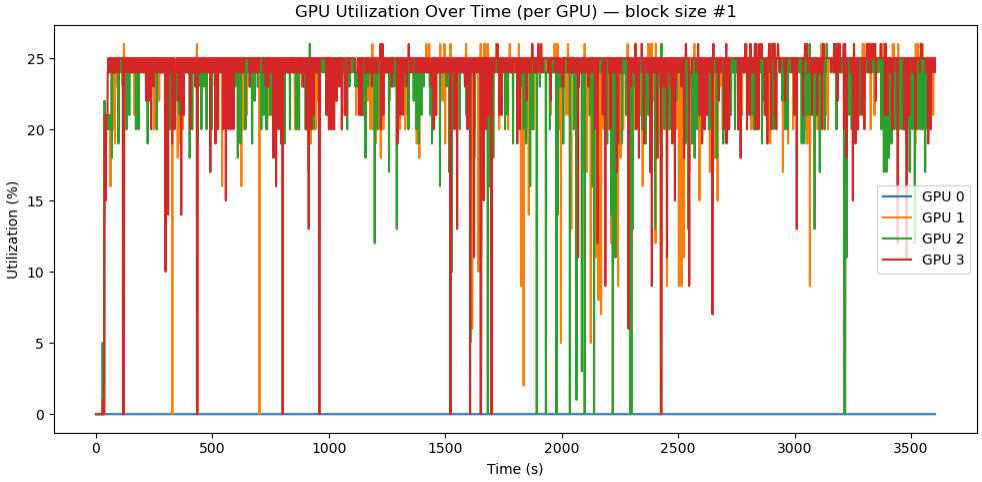

Figure 5.7: GPU utilization for GPUs 1-3 over time for a block_size of 1 (Note: GPU 0 exists but is associated with the head node and thus not utilized). Each GPU is constantly used but only reaching a maximum of around 25% utilization. This means the GPU is under-utilized but the limited downtime suggests segments are bottlenecked waiting for a free GPU.

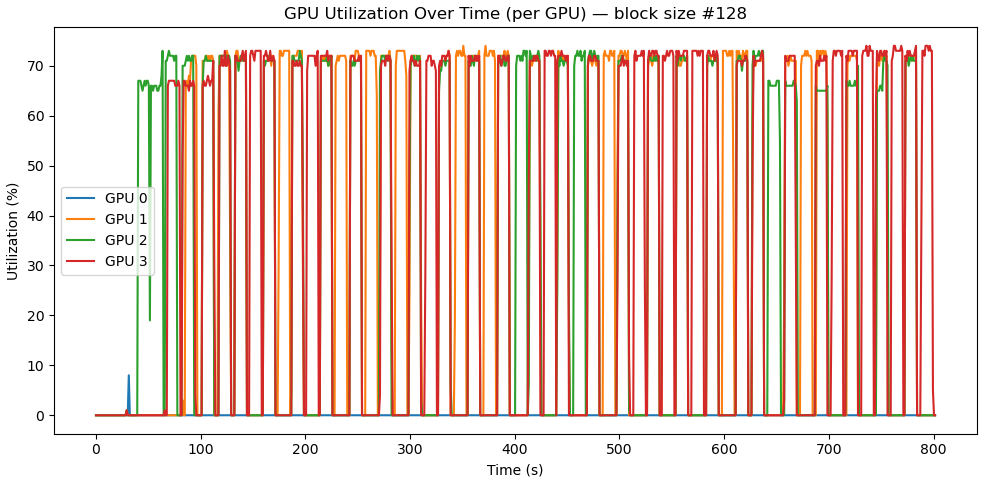

Figure 5.8: GPU utilization for a block_size of 128. Utilization stands around 75% per GPU with longer blocks without usage due to segments being processed faster than they are being requested, meaning there is very low latency between a requested processing of a segment and the beginning of the computation.

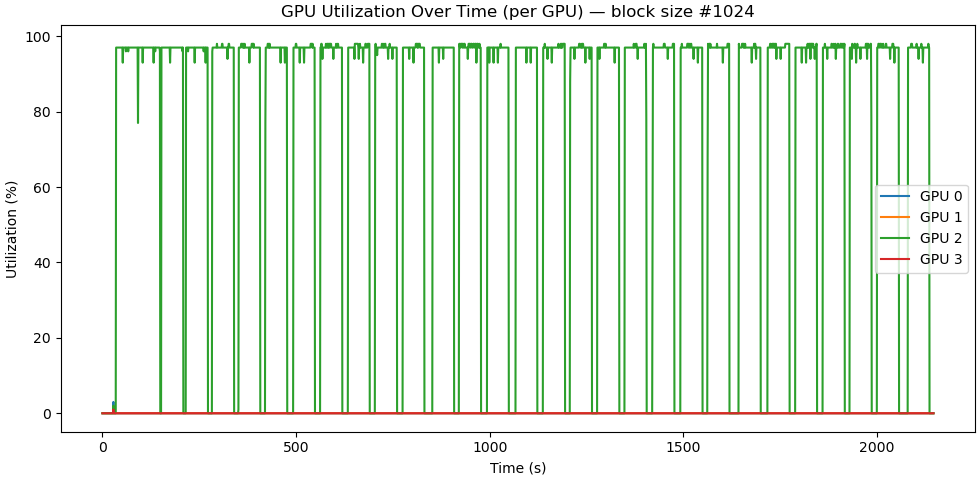

Figure 5.9: GPU utilization for a block_size of 1024. Since the amount of segments per iteration is well below that block size, this means that the entire iteration is loaded into one GPU and run all simultaneously. This leads to a nearly 100% utilization of the GPU. It is worth nothing that despite increased utilization, the time per segment increased and the total time required to complete 30 iterations is longer than with a block_size of 128. This suggests that there is a balance between too much and too little utilization, as over-saturating a GPU appears to decrease its performance.

5.3 Future Optimizations

While some of the optimizations outlined here are useful, there are additional optimizations that can be made in the future to further accelerate molecular dynamics. Certain optimizations are clear, such as finding the optimal number of nodes for training (the maximum before performance decreases) or the optimal block size for a given configuration in trajectory packing.

Further, the trajectory packing, while useful, is restricted to TorchMD-Net force fields, meaning that all-atom dynamics remain singularly progressed in time with one GPU, leaving utilization low. Adding trajectory packing into OpenMM could have significant acceleration for parallel dynamic propagation like WESTPA. While the implementation of this optimization would likely be much more complex and require differing strategies, the core idea remains the same, providing potential orders of magnitude speed as outlined in Figure 5.4.

Bibliography

[1] B. J. Alder and T. E. Wainwright. Phase transition for a hard sphere system. *The Journal of Chemical Physics*, 27(5):1208–1209, 11 1957.

[2] A. Rahman. Correlations in the motion of atoms in liquid argon. *Phys. Rev.*, 136:A405–A411, Oct 1964.

[3] Loup Verlet. Computer "experiments" on classical fluids. i. thermodynamical properties of lennard-jones molecules. *Phys. Rev.*, 159:98–103, Jul 1967.

[4] Jean-Paul Ryckaert, Giovanni Ciccotti, and Herman J.C Berendsen. Numerical integration of the cartesian equations of motion of a system with constraints: molecular dynamics of n-alkanes. *Journal of Computational Physics*, 23(3):327–341, 1977.

[5] Hans C Andersen. Rattle: A “velocity” version of the shake algorithm for molecular dynamics calculations. *Journal of Computational Physics*, 52(1):24–34, 1983.

[6] Tom Darden, Darrin York, and Lee Pedersen. Particle mesh ewald: An nlog(n) method for ewald sums in large systems. *The Journal of Chemical Physics*, 98(12):10089–10092, 06 1993.

[7] J. A. McCammon, B. R. Gelin, and M. Karplus. Dynamics of folded proteins. *Nature*, 267(5612):585–590, June 1977.

[8] Mark James Abraham, Teemu Murtola, Roland Schulz, Szilárd Páll, Jeremy C. Smith, Berk Hess, and Erik Lindahl. Gromacs: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. *SoftwareX*, 1-2:19–25, 2015.

[9] 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. *Journal of Computational Chemistry*, 4:187–217, 1983.

[10] Peter Eastman, Raimondas Galvelis, Rodrigo P. Peláez, Carlos R. A. Abreu, Scott E. Farr, Emilio Gallicchio, Artem Gorenko, Michael M. Henry, Fan Hu, Jing Huang, Andreas Krämer, Julien Michel, John A. Mitchell, Vijay S. Pande, João P. G. L. M. Rodrigues, Juan Rodriguez-Guerra, Andrew C. Simmonett, Sagar Singh, Jason Swails, Peter Turner, Yutong Wang, Ivan Zhang, John D. Chodera, Gianni De Fabritiis, and Thomas E. Markland. Openmm 8: Molecular dynamics simulation with machine learning potentials. *Journal of Physical Chemistry B*, 128(1):109–116, 2023.

[11] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A. A. Kohl, Andrew J. Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W. Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. Highly accurate protein structure prediction with AlphaFold. *Nature*, 596(7873):583–589, 2021.

[12] David E. Shaw, Peter J. Adams, Asaph Azaria, Joseph A. Bank, Brannon Batson, Alistair Bell, Michael Bergdorf, Jhanvi Bhatt, J. Adam Butts, Timothy Correia, Robert M. Dirks, Ron O. Dror, Michael P. Eastwood, Bruce Edwards, Amos Even, Peter Feldmann, Michael Fenn, Christopher H. Fenton, Anthony Forte, Joseph Gagliardo, Gennette Gill, Maria Gorlatova, Brian Greskamp, J.P. Grossman, Justin Gullingsrud, Anissa Harper, William Hasen-

plaugh, Mark Heily, Benjamin Colin Heshmat, Jeremy Hunt, Douglas J. Ierardi, Lev Iserovich, Bryan L. Jackson, Nick P. Johnson, Mollie M. Kirk, John L. Klepeis, Jeffrey S. Kuskin, Kenneth M. Mackenzie, Roy J. Mader, Richard McGowen, Adam McLaughlin, Mark A. Moraes, Mohamed H. Nasr, Lawrence J. Nociolo, Lief O'Donnell, Andrew Parker, Jon L. Peticolas, Goran Pocina, Cristian Predescu, Terry Quan, John K. Salmon, Carl Schwink, Keun Sup Shim, Naseer Siddique, Jochen Spengler, Tamas Szalay, Raymond Tabladillo, Reinhard Tartler, Andrew G. Taube, Michael Theobald, Brian Towles, William Vick, Stanley C. Wang, Michael Wazlowski, Madeleine J. Weingarten, John M. Williams, and Kevin A. Yuh. Anton 3: twenty microseconds of molecular dynamics simulation before lunch. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21, New York, NY, USA, 2021. Association for Computing Machinery.

[13] George A. Huber and Samuel Kim. Weighted-ensemble brownian dynamics simulations for protein association reactions. *Biophysical Journal*, 70(1):97–110, Jan 1996.

[14] D.M. Zuckerman and L.T. Chong. Weighted ensemble simulation: review of methodology, applications, and software. *Annu. Rev. Biophys.*, 46:43–57, 2017.

[15] David Aristoff and Daniel M. Zuckerman. Optimizing weighted ensemble sampling of steady states. *Multiscale Modeling & Simulation*, 18(2):646–673, 2020.

[16] Bin W. Zhang, David Jasnow, and Daniel M. Zuckerman. The "weighted ensemble" path sampling method is statistically exact for a broad class of stochastic processes and binning procedures. *The Journal of Chemical Physics*, 132(5):054107, 2010.

[17] Paul A. Torrillo, Anthony T. Bogetti, and Lillian T. Chong. A minimal, adaptive binning scheme for weighted ensemble simulations. *J. Phys. Chem. A*, 125(7):1642–1649, 2021.

[18] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 1263–1272. JMLR.org, 2017.

[19] K. T. Schütt, P.-J. Kindermans, H. E. Sauceda, S. Chmiela, A. Tkatchenko, and K.-R. Müller. Schnet: a continuous-filter convolutional neural network for modeling quantum interactions. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 992–1002, Red Hook, NY, USA, 2017. Curran Associates Inc.

[20] Raul P. Pelaez, Guillem Simeon, Raimondas Galvelis, Antonio Mirarchi, Peter Eastman, Stefan Doerr, Philipp Thölke, Thomas E. Markland, and Gianni De Fabritiis. Torchmd-net 2.0: Fast neural network potentials for molecular simulations. *Journal of Chemical Theory and Computation*, 20(10):4076–4087, 2024.

[21] Philipp Thölke and Gianni Fabritiis. Torchmd-net: Equivariant transformers for neural network based molecular potentials, 02 2022.

[22] Brooke E. Husic, Nicholas E. Charron, Daniel Lemm, Jinzhe Wang, Adrià Pérez, Mariusz Majewski, Andreas Krämer, Yao-Tsung Chen, Simon Olsson, Gianni de Fabritiis, Frank Noé, and Cecilia Clementi. Coarse graining molecular dynamics with graph neural networks. *The Journal of Chemical Physics*, 153(19):194101, 2020.

[23] Kresten Lindorff-Larsen, Stefano Piana, Ron O. Dror, and David E. Shaw. How fast-folding proteins fold. *Science*, 334(6055):517–520, October 2011.

[24] Maciej Majewski, Adrià Pérez, Philipp Thölke, Stefan Doerr, Nicholas E. Charron, Toni Giorgino, Brooke E. Husic, Cecilia Clementi, Frank Noé, and Gianni De Fabritiis. Machine learning coarse-grained potentials of protein thermodynamics. *Nature Communications*, 14(1):5739, September 2023.

[25] John D. Russo, She Zhang, Jeremy M. G. Leung, Anthony T. Bogetti, Jeff P. Thompson, Alex J. DeGrave, Paul A. Torrillo, A. J. Pratt, Kim F. Wong, Junchao Xia, Jeremy Copper-

man, Joshua L. Adelman, Matthew C. Zwier, David N. LeBard, Daniel M. Zuckerman, and Lillian T. Chong. Westpa 2.0: High-performance upgrades for weighted ensemble simulations and analysis of longer-timescale applications. *Journal of Chemical Theory and Computation*, 18(2):638–649, January 2022.

[26] Martin K. Scherer, Benjamin Trendelkamp-Schroer, Fabian Paul, Guillermo Pérez-Hernández, Moritz Hoffmann, Nuria Plattner, Christoph Wehmeyer, Jan-Hendrik Prinz, and Frank Noé. Pyemma 2: A software package for estimation, validation, and analysis of markov models. *Journal of Chemical Theory and Computation*, 11(11):5525–5542, October 2015.

[27] David Aristoff. An ergodic theorem for the weighted ensemble method. *Journal of Applied Probability*, 59:1–15, 01 2022.

[28] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. *PyTorch: an imperative style, high-performance deep learning library*. Curran Associates Inc., Red Hook, NY, USA, 2019.

[29] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. Pytorch distributed: experiences on accelerating data parallel training. *Proc. VLDB Endow.*, 13(12):3005–3018, August 2020.

Chapter 6

Appendix

6.1 Proof of Theorem 2

Let $(X_t)_{t=0,\tau,2\tau,\dots}$ be a Markov process on a measurable space $\mathscr{X}$ with initial distribution μ_0 and transition kernel K. Let μ_k denote the law of $X_{k\tau}$. At each time $t_k = k\tau$, let

$$\widehat{\mu}_k^N = \sum_{i=1}^{N_k} W_i^k \delta_{X_i^k}, \qquad \sum_{i=1}^{N_k} W_i^k = 1,$$

be the empirical measure produced by the weighted ensemble procedure. Then for every bounded measurable $g : \mathscr{X} \to \mathbb{R}$ and every $k \geq 0$,

$$\mathbb{E}\left[\sum_{i=1}^{N_k} W_i^k g(X_i^k)\right] = \int_{\mathscr{X}} g(x)\, \mu_k(dx).$$

Proof. Initialization: $X_i^0 \sim \mu_0$ i.i.d. and $W_i^0 = 1/N_0$. Thus

$$\mathbb{E}\left[\sum_i W_i^0 g(X_i^0)\right] = \mathbb{E}[g(X_0)] = \langle g, \mu_0 \rangle.$$

Inductive hypothesis: for some $k \geq 0$,

$$\mathbb{E}\left[\sum_{i=1}^{N_k} W_i^k g(X_i^k)\right] = \int g(x)\,\mu_k(dx).$$

Propagation step: define

$$X_i^{k+1,-} \sim K(X_i^k, \cdot), \qquad W_i^{k+1,-} = W_i^k.$$

Then

$$\mathbb{E}[g(X_i^{k+1,-}) \mid X_i^k] = \int g(y)\,K(X_i^k, dy).$$

Let $h(x) = \int g(y)K(x,dy)$. Using conditional expectation,

$$\mathbb{E}\left[\sum_i W_i^{k+1,-} g(X_i^{k+1,-})\right] = \mathbb{E}\left[\sum_i W_i^k h(X_i^k)\right].$$

By the inductive hypothesis,

$$\mathbb{E}\left[\sum_i W_i^k h(X_i^k)\right] = \int h(x)\,\mu_k(dx) = \int g(y)\,\mu_{k+1}(dy).$$

Hence

$$\mathbb{E}\left[\sum_i W_i^{k+1,-} g(X_i^{k+1,-})\right] = \int g\,d\mu_{k+1}. \tag{1}$$

Resampling step: partition $\mathscr{X}$ into bins $B_1, \ldots, B_B$. For each bin b, define

$$S_b = \{i : X_i^{k+1,-} \in B_b\}, \qquad W_b = \sum_{i \in S_b} W_i^{k+1,-},$$

and select an integer $n_b \geq 0$. If $n_b > 0$, set $\omega_b = W_b/n_b$ and draw indices $I_{b,1}, \ldots, I_{b,n_b}$ independently from S_b with

$$\mathbb{P}(I_{b,j} = i) = \frac{W_i^{k+1,-}}{W_b}, \qquad i \in S_b.$$

Define

$$X_{b,j}^{k+1} = X_{I_{b,j}}^{k+1,-}, \qquad W_{b,j}^{k+1} = \omega_b.$$

Let the post-resampling set be $\{(X_i^{k+1}, W_i^{k+1})\}$.

For each bin,

$$\mathbb{E}[g(X_{b,j}^{k+1}) \mid \{X_i^{k+1,-}, W_i^{k+1,-}\}] = \sum_{i \in S_b} \frac{W_i^{k+1,-}}{W_b} g(X_i^{k+1,-}),$$

hence

$$\mathbb{E}\left[\sum_{j=1}^{n_b} W_{b,j}^{k+1} g(X_{b,j}^{k+1}) \mid \text{pre}\right] = W_b \sum_{i \in S_b} \frac{W_i^{k+1,-}}{W_b} g(X_i^{k+1,-}) = \sum_{i \in S_b} W_i^{k+1,-} g(X_i^{k+1,-}).$$

Summing over bins,

$$\mathbb{E}\left[\sum_{i=1}^{N_{k+1}} W_i^{k+1} g(X_i^{k+1}) \mid \text{pre}\right] = \sum_{i=1}^{N_k} W_i^{k+1,-} g(X_i^{k+1,-}).$$

Taking expectation,

$$\mathbb{E}\left[\sum_{i=1}^{N_{k+1}} W_i^{k+1} g(X_i^{k+1})\right] = \mathbb{E}\left[\sum_i W_i^{k+1,-} g(X_i^{k+1,-})\right]. \tag{2}$$

Combining (1) and (2) yields

$$\mathbb{E}\left[\sum_i W_i^{k+1} g(X_i^{k+1})\right] = \int g(x)\, \mu_{k+1}(dx),$$

which completes the induction. □

Corollary 2.1 (Importance Sampling Interpretation)**.** *For each fixed k, the weighted ensemble measure* $\widehat{\mu}_k^N = \sum_i W_i^k \delta_{X_i^k}$ *satisfies*

$$\mathbb{E}\left[\sum_i W_i^k g(X_i^k)\right] = \int g(x)\, \mu_k(dx)$$

for every bounded measurable g. Hence the weights W_i^k act as Radon–Nikodym derivatives $d\mu_k / d\nu_k$ with respect to the sampling distribution ν_k induced by the stochastic propagation and resampling steps.

6.2 GPU Utilization Graphs

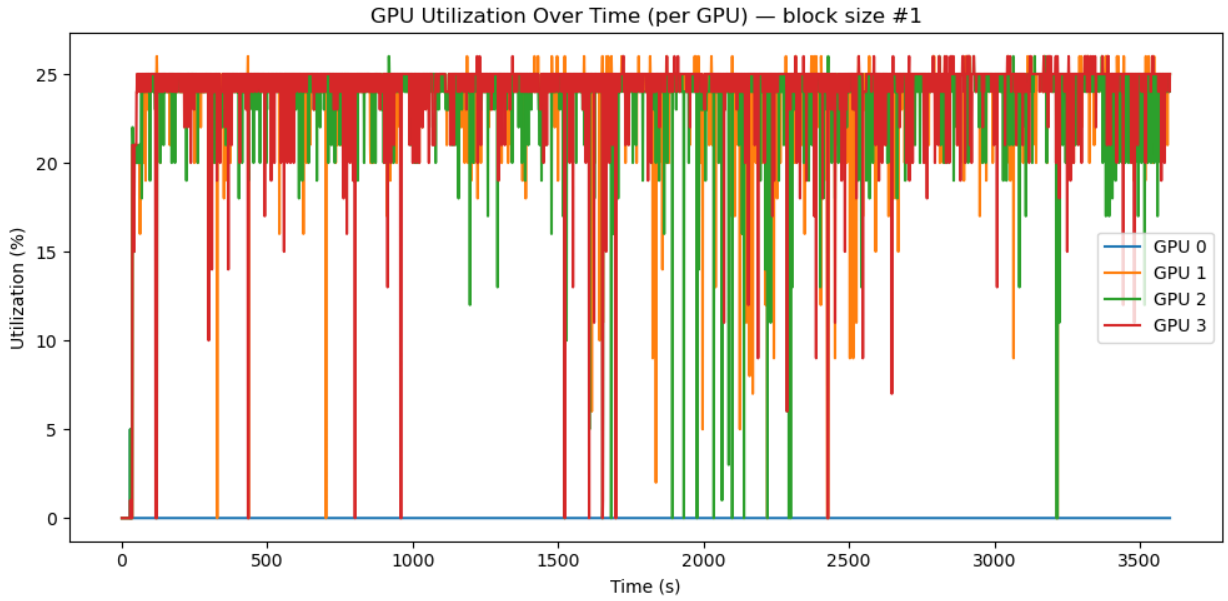

Figure 6.1: GPU Utilization Graph: Block Size = 1

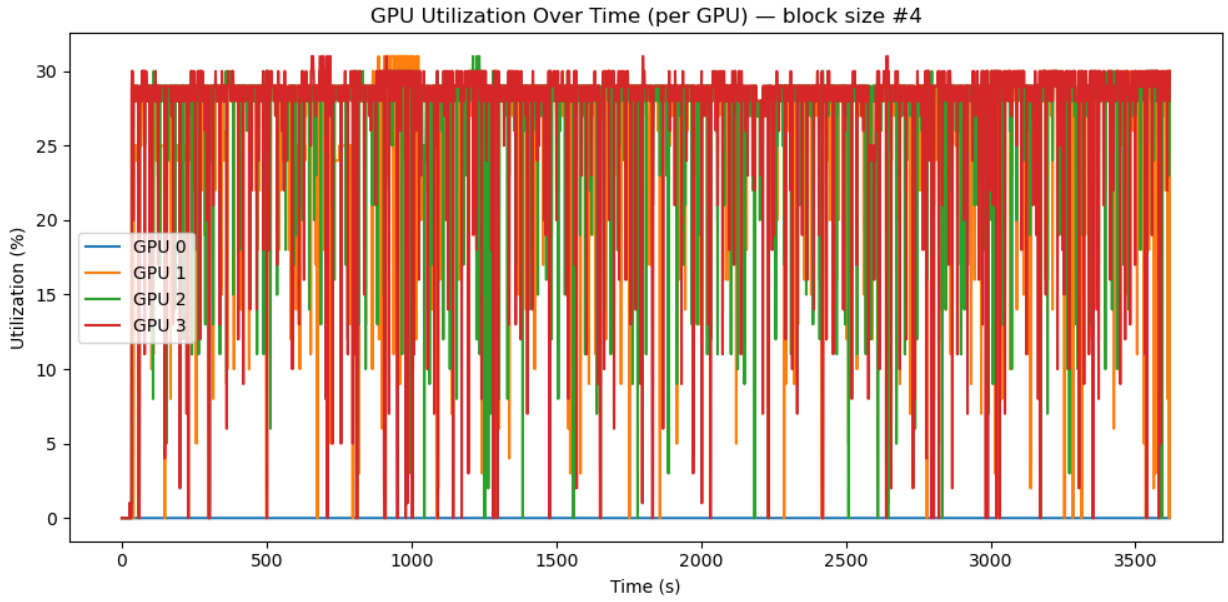

Figure 6.2: GPU Utilization Graph: Block Size = 4

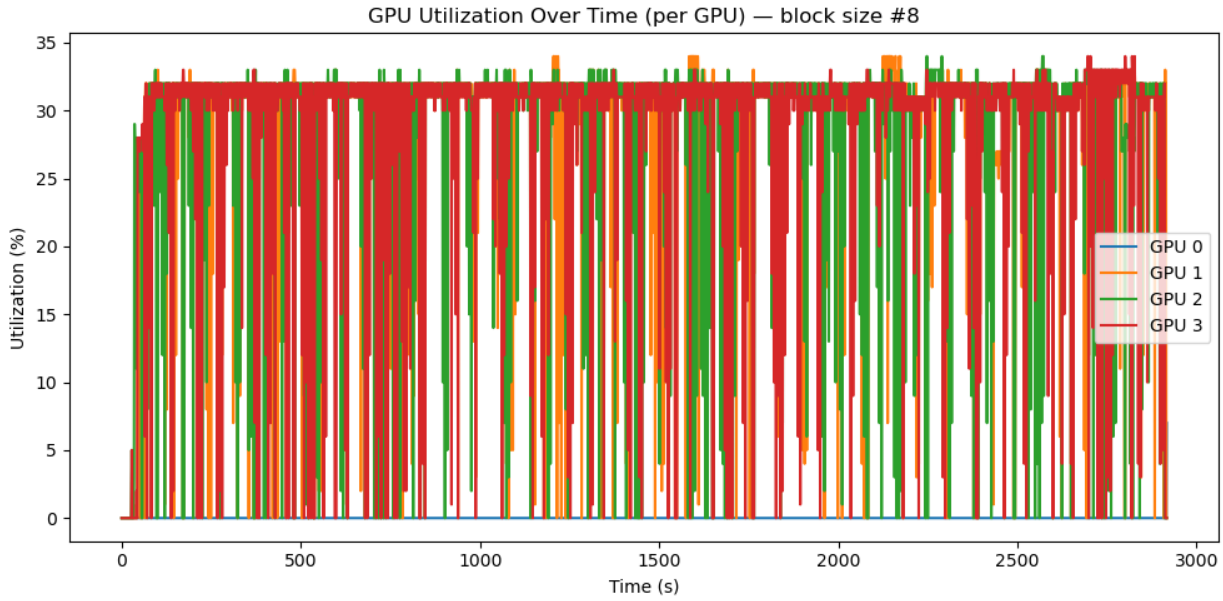

Figure 6.3: GPU Utilization Graph: Block Size = 8

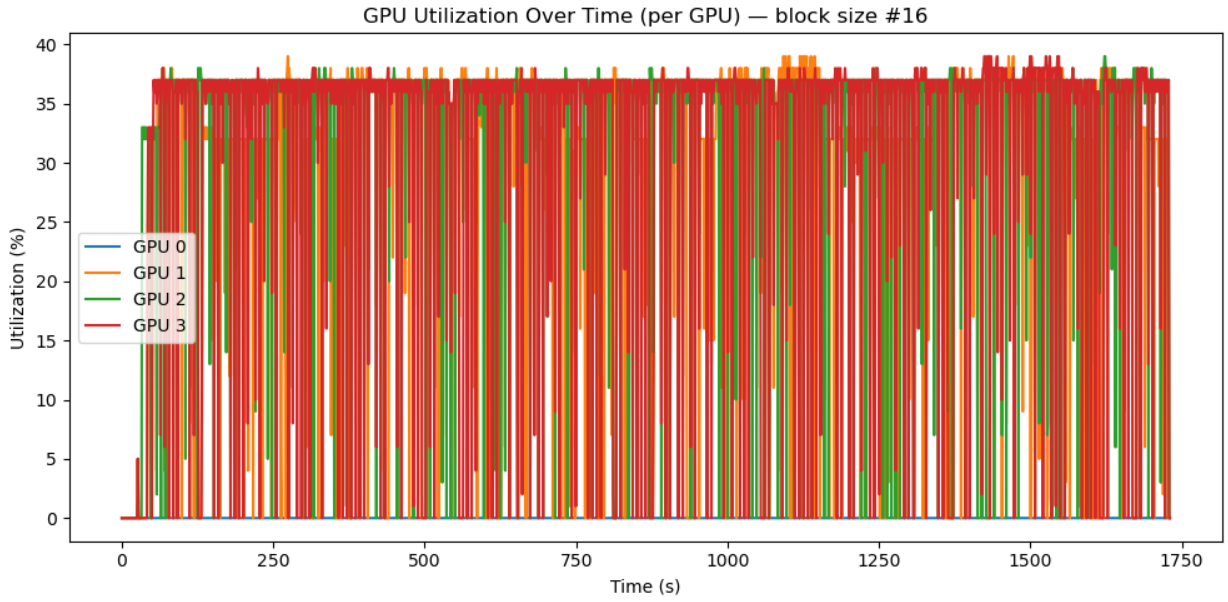

Figure 6.4: GPU Utilization Graph: Block Size = 16

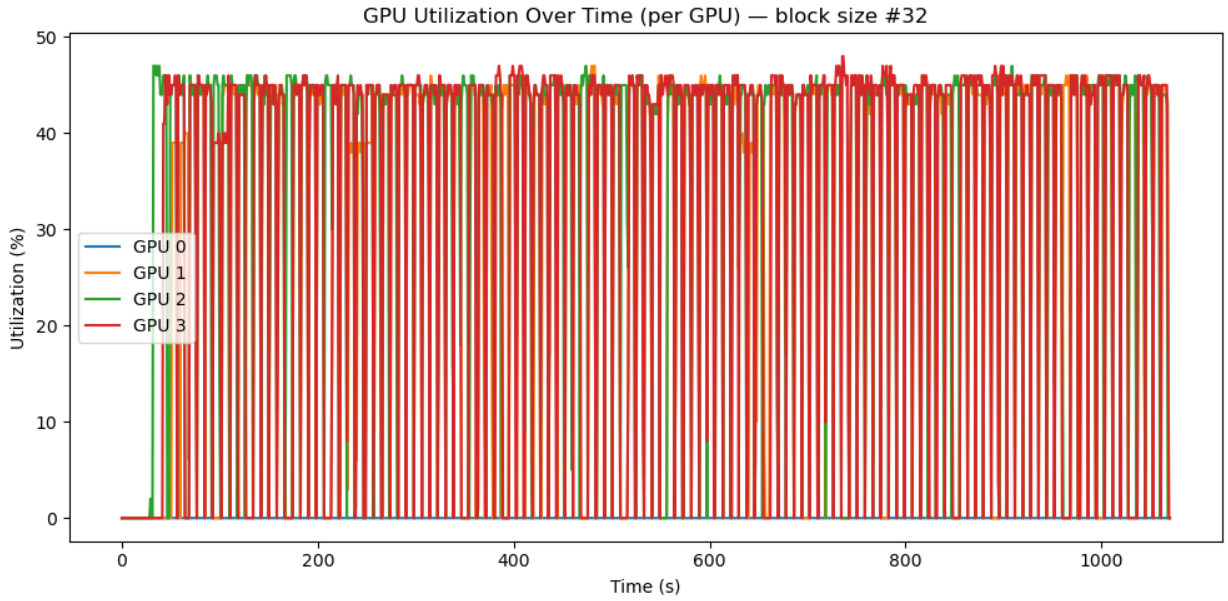

Figure 6.5: GPU Utilization Graph: Block Size = 32

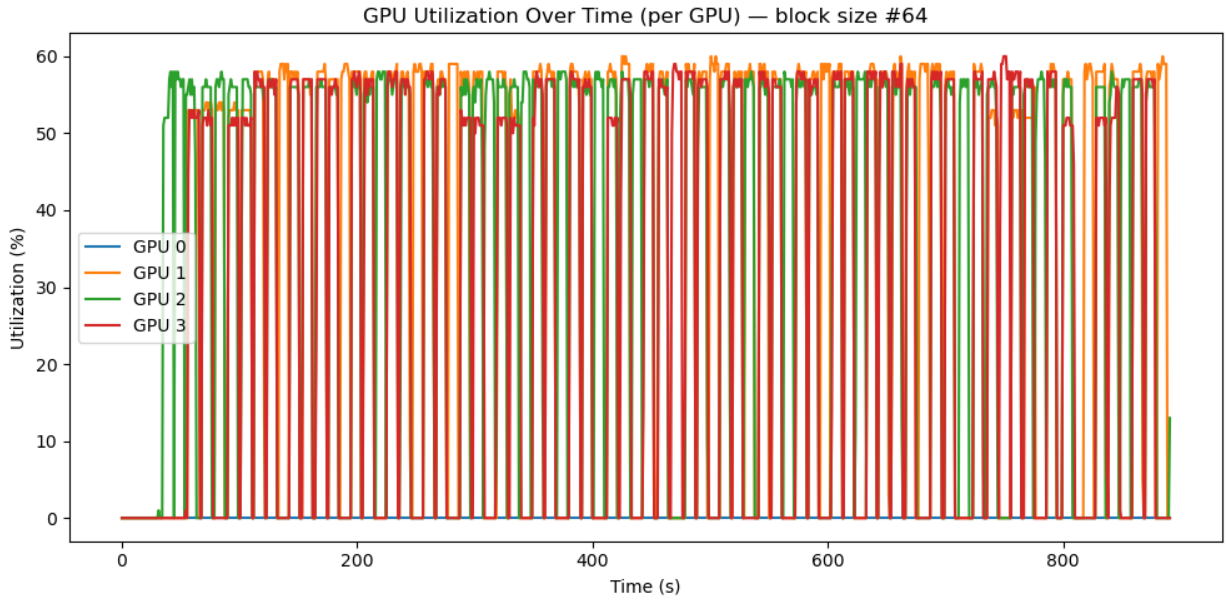

Figure 6.6: GPU Utilization Graph: Block Size = 64

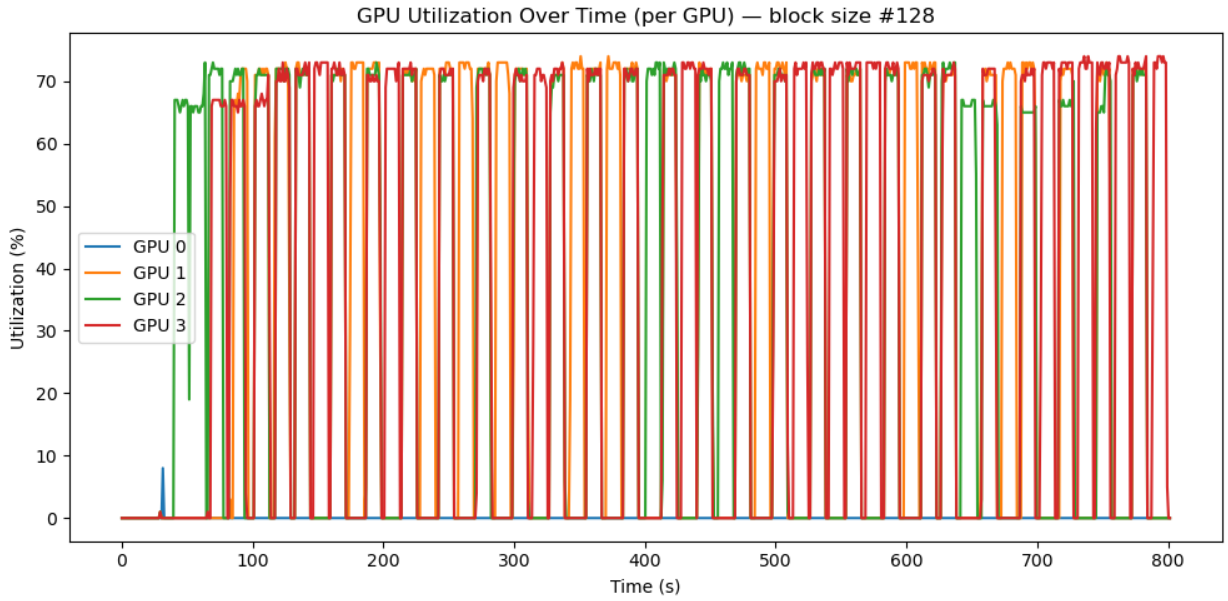

Figure 6.7: GPU Utilization Graph: Block Size = 128

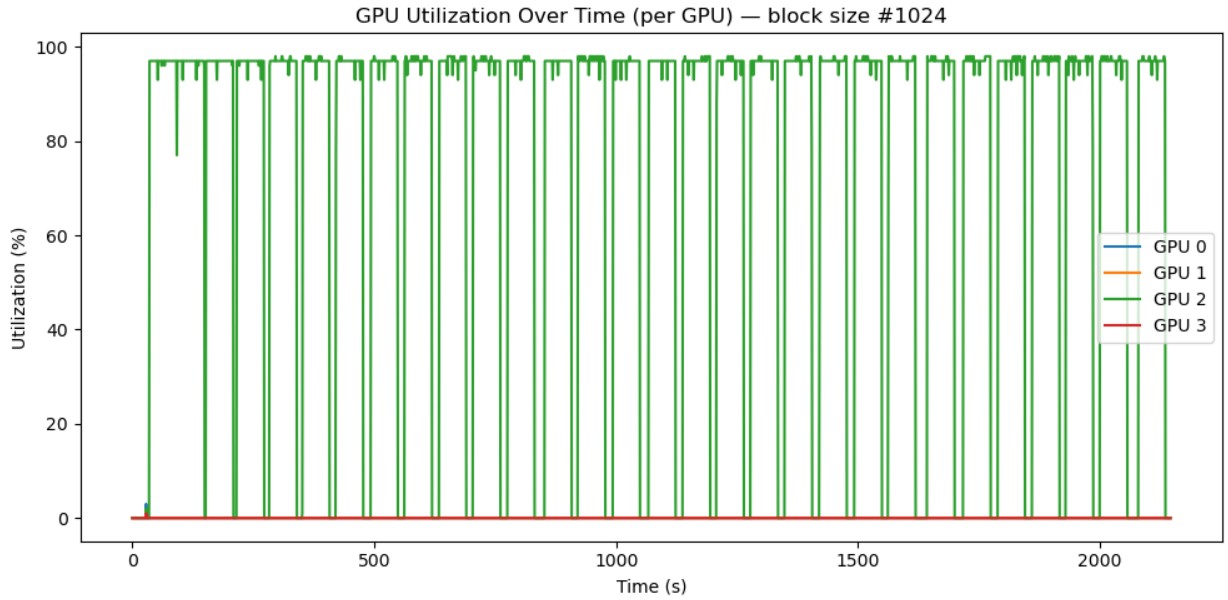

Figure 6.8: GPU Utilization Graph: Block Size = 1024