JAMUN: Transferable Molecular Conformational Ensemble Generation with Walk-Jump Sampling

Ameya Daigavane[†*] **Bodhi P. Vani**[†], **Saeed Saremi, Joseph Kleinhenz**[‡], **Joshua Rackers**[‡]
MIT
ameyad@mit.edu

Prescient Design
Genentech Computational Sciences
South San Francisco, CA, USA
{vanib,saremis,kleinhej,rackersj}@gene.edu

Abstract

Conformational ensembles of protein structures are immensely important both to understanding protein function, and for drug discovery in novel modalities such as cryptic pockets. Current techniques for sampling ensembles are computationally inefficient, or do not transfer to systems outside their training data. We present Walk-**J**ump **A**ccelerated **M**olecular ensembles with **U**niversal **N**oise (JAMUN), a step towards the goal of efficiently sampling the Boltzmann distribution of arbitrary proteins. By extending Walk-Jump Sampling to point clouds, JAMUN enables ensemble generation at orders of magnitude faster rates than traditional molecular dynamics or state-of-the-art ML methods. Further, JAMUN is able to predict the stable basins of small peptides that were not seen during training.

1 Introduction

Proteins are inherently dynamic entities constantly in motion, and these movements can be vitally important. They are not well characterized as single structures as has traditionally been the case, but rather as ensembles of structures with an ergodic probability distribution (Henzler-Wildman & Kern, 2007). Protein motion is required for myglobin to bind oxygen and move it around the body (Miller & Phillips, 2021). Drug discovery on protein kinases depends on characterizing kinase conformational ensembles (Gough & Kalodimos, 2024). The search for druggable 'cryptic pockets' requires understanding protein dynamics, and antibody design is deeply affected by conformational ensembles (Colombo, 2023). However, while machine learning (ML) methods for molecular structure prediction have experienced enormous success recently, ML methods for dynamics have yet to have similar impact. ML models for generating molecular ensembles are widely considered the 'next frontier' (Bowman, 2024; Miller & Phillips, 2021; Zheng et al., 2023). In this work, we present JAMUN (Walk-**J**ump **A**ccelerated **M**olecular ensembles with **U**niversal **N**oise), a generative ML model which advances this frontier by demonstrating improvements in both speed and transferability over previous approaches.

While the importance of protein dynamics is well-established, it can be exceedingly difficult to sufficiently sample large biomolecular systems. The most common sampling method is molecular dynamics (MD), but it is limited by the need for very short time-steps of 1-2 femtoseconds. Many important protein dynamic phenomena occur on the timescale of milliseconds. Simulating with this resolution is (Borhani & Shaw, 2012) '...equivalent to tracking the advance and retreat of the glaciers of the last Ice Age – tens of thousands of years – by noting their locations each and every second.' Importantly, there is nothing fundamental about this small time-step limitation; it is

[*]Work performed when Ameya was an intern at Prescient Design.
[†]These authors contributed equally to this work.
[‡]These authors contributed equally to this work.

Preprint. Under review.

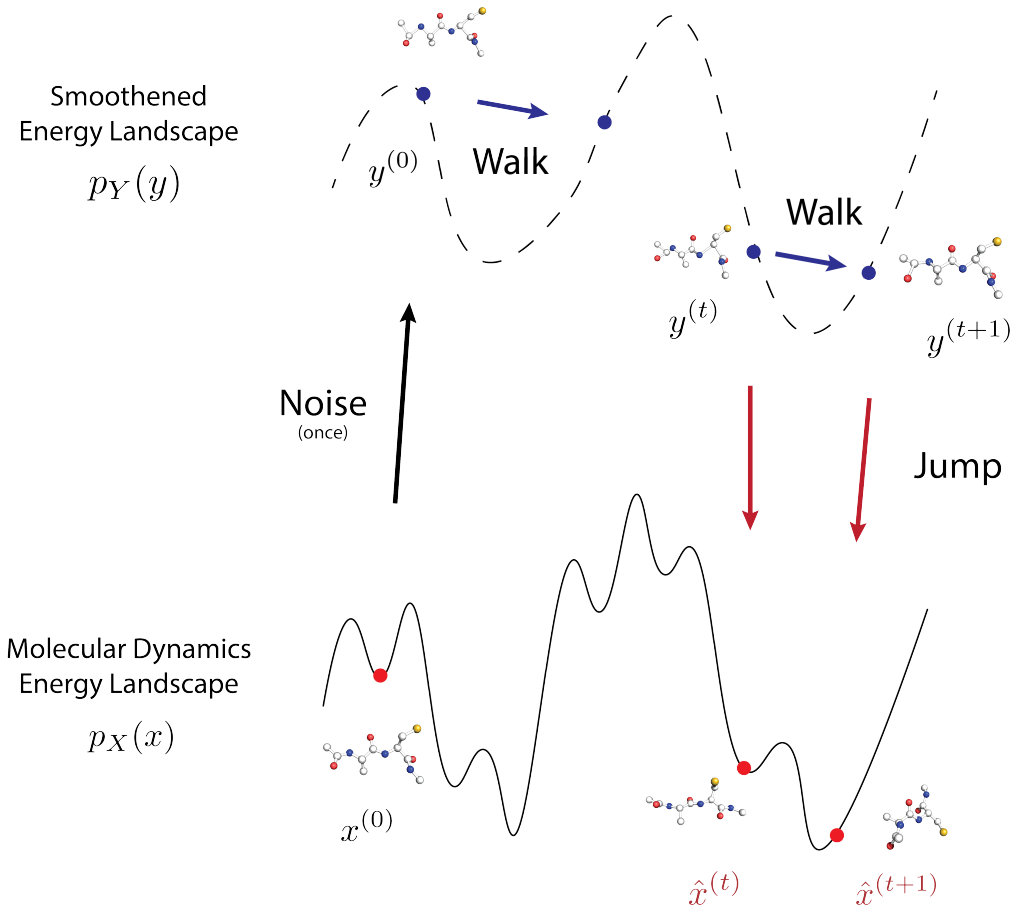


Figure 1: Overview of Walk-Jump Sampling.

an artifact of high-frequency motions, such as bond vibrations, that have little to no effect on protein ensembles (Leimkuhler & Matthews, 2015). Enhanced sampling methods have been developed in an attempt to accelerate sampling, but they often require domain knowledge about relevant collective variables, and, more importantly, do not address the underlying time-step problem. Other sampling methods, such as Monte Carlo-based methods, exist, but have seen limited success for large biomolecular systems (Vitalis & Pappu, 2009).

A large number of generative models have been developed to address the sampling inefficiency problems of MD using machine learning, including continuous normalizing flows, diffusion, and flow-matching (Noé et al., 2019; Arts et al., 2023; Klein et al., 2024b,a; Kim et al., 2024; Zheng et al., 2024; Jing et al., 2024a,b). These models have been applied to a variety of MD datasets from small molecules to peptides to full proteins. However, many of these models do not work well beyond their training data. (One exception is the recent Transferable Boltzmann Generators (Klein & Noé, 2024) model, with which we provide extensive comparison. AlphaFlow (Jing et al., 2024a) and Distributional Graphormer (Zheng et al., 2024) are technically transferable, but are much larger models trained on diverse datasets, making it harder to benchmark their transferability in the setting we discuss here. In the future, we will also compare against MDGen (Jing et al., 2024b), a recently released model for generative modelling of entire MD trajectories.) While the developments in this field have been immense, transferability remains a grand challenge. Without transferability, the usefulness of any ML model is extremely limited. For a true breakthrough in this area, an ML model must be able to generate conformational ensembles for molecules that are not in its training set.

We set out to solve this problem of transferability by developing an ML model informed by the physical priors of molecular dynamics data. JAMUN is a Walk-Jump Sampler (WJS) (Saremi & Hyvärinen, 2019) for point clouds parameterized with an $SE(3)$-equivariant denoiser. In WJS, noise is added to clean data and a denoising neural network is trained to recover the clean samples.

This denoiser defines the score function of the noisy manifold which we sample using Langevin dynamics (walk step) and allows us to periodically project back to the original data distribution (jump step). Crucially, the walk and jump steps are *decoupled* from each other. In MD, unlike for natural images or other settings where generative modeling is commonly applied, we typically are interested in sampling an ensemble of representative states rather than drawing single samples. In this setting it is advantageous to generate samples from trajectories that efficiently traverse the smoothed space, rather than starting over from an uninformative Gaussian prior for each sample as is commonly done in diffusion and flow matching. By adding noise at a carefully chosen scale, WJS simply smooths out the distribution enough to resolve sampling difficulties without fully destroying the information present in the data distribution. Moreover, Langevin dynamics, which is the same algorithm commonly used for MD simulations, but applied on the smoothed noisy manifold lends itself to simple, physical interpretations of model behavior.

We train JAMUN on a large dataset of MD simulations of two amino acid peptides. We demonstrate that this model can generalize to a holdout set of unseen peptides. In all of these cases, generation with JAMUN yields converged sampling of the conformational ensemble faster than MD with a standard force field. These results suggest that this transferability is a consequence of retaining the physical priors inherent in MD data. By smoothing out the underlying data distribution, JAMUN is able to produce the first transferable generative model for molecular conformational ensembles that is dramatically faster than MD simulation.

2 Related Work

The goal of building machine learning models that can generate conformational ensembles of molecular systems is not new. While a full overview of this field is beyond the scope of this work (see Aranganathan et al. (2024) for a recent review), we note a few relevant previous efforts. Boltzmann Generators (Noé et al., 2019) introduced the idea that a neural network could be used to transform the underlying data distribution into an easier-to-sample Gaussian distribution. There have been follow-on efforts which used diffusion models (Arts et al., 2023), flow-matching (Klein et al., 2024b), and continuous normalizing flows (Klein & Noé, 2024). The commonality in these models is the choice of target distribution; they all attempt to transform the MD data distribution into a simple Gaussian. This is a key difference between prior work and our model. Notably, no previous model in this area except Klein & Noé (2024) has been transferable.

There have also been efforts to build ML models for taking longer MD time-steps (Klein et al., 2024a; Schreiner et al., 2023; Hsu et al., 2024) and for approximating conformations of large proteins (Zheng et al., 2024; Jing et al., 2024a). These methods rely on hand-crafted featurizations (eg. backbone torsion angles). In practice, this has made generalization to unseen molecules challenging for these models as well.

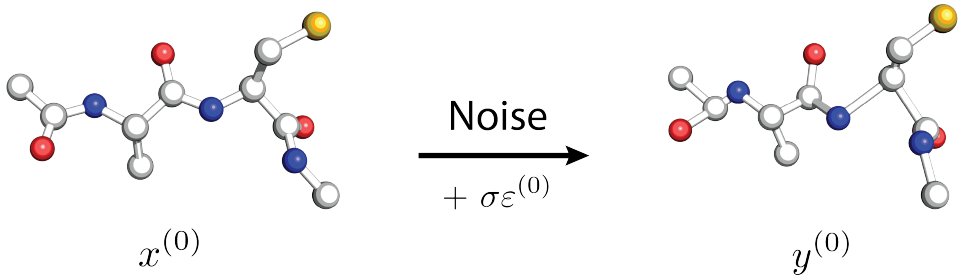
The above models are often classified as Boltzmann Generators or Boltzmann Emulators. Models in the former class are guaranteed to draw unbiased samples from the Boltzmann distribution, while models in the latter class do not have this guarantee. Strictly speaking, JAMUN is a Boltzmann Emulator, although in practical terms, as our results show, the difference is minimal.

JAMUN is a Walk-Jump Sampling method which uses an $SE(3)$-equivariant neural network for denoising. WJS is built on the seminal work of Neural Empirical Bayes (Saremi & Hyvärinen, 2019), and has been used in voxelized molecule generation (Pinheiro et al., 2024a,b) and protein sequence generation (Frey et al., 2023). Our work is the first to our knowledge to apply WJS to point clouds.

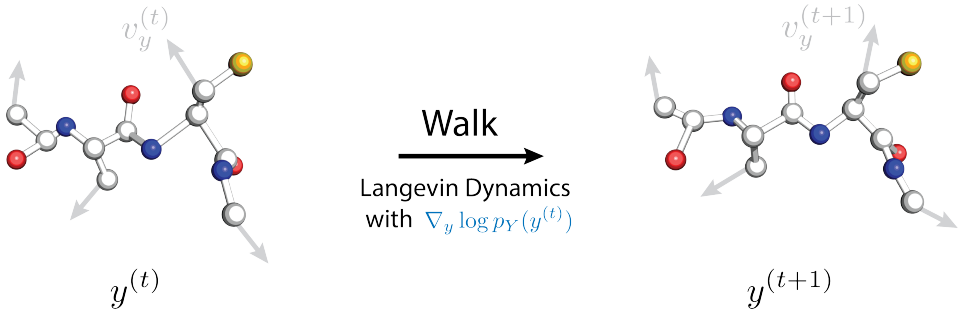
3 Methods

3.1 Representing Peptides as Point Clouds

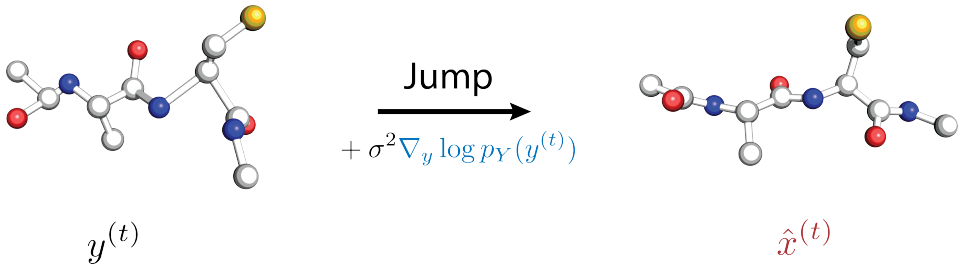
Each point cloud of N atoms can be represented by the 2-tuple (x, h) where $x \in \mathbb{R}^{N\times 3}$ represents the 3D coordinates of each of the N atoms and $h \in \mathbb{R}^D$ represents atom and bonding information. h is assumed to be known for each peptide, and not sampled over. For clarity of presentation, we omit the conditioning on h in the distributions and models below. We discuss how our model uses h in Section 3.4.



(a) Adding noise to an initial conformation $x^{(0)}$ to obtain $y^{(0)} \sim p_Y$.



(b) One iteration of BAOAB-discretized Langevin dynamics (Equation 2 and Equation 67) starting from $y^{(t)} \sim p_Y$ leads to a new sample $y^{(t+1)} \sim p_Y$.



(c) Denoising of $y^{(t)}$ according to Equation 6 gives us new samples $\hat{x}^{(t)}$.

Figure 2: Illustrating each of the noise (Figure 2a), walk (Figure 2b) and jump steps (Figure 2c) in Walk-Jump Sampling.

For each diamine at sampling time, we assume access to an initial sample $x^{(0)}$ where $x^{(0)} \in \mathbb{R}^{N \times 3}$ is sampled from the clean data distribution p_X.

Knowledge of $x^{(0)}$ and h is not an overly restrictive assumption, as they can be easily obtained (e.g. from crystallized structures from the PDB (Berman et al., 2000)), or even procedurally generated (e.g. the `fab` command in PyMOL, or the `sequence` command in the LEaP program packaged with the Amber force fields).

3.2 Walk-Jump Sampling

JAMUN operates by performing Walk-Jump sampling on molecular systems represented as 3D point clouds. A conceptual overview of the process is illustrated in Figure 1, with specific steps depicted in Figure 2.

Given the initial sample $x^{(0)} \sim p_X$, Walk-Jump performs the following steps:

1. **Noise** the initial structure $x^{(0)}$ to create the initial sample $y^{(0)}$ from the noisy data distribution p_Y (Figure 2a):

$$y^{(0)} = x^{(0)} + \sigma\varepsilon^{(0)} \text{ where } \varepsilon^{(0)} \sim \mathcal{N}(0, \mathbb{I}_{N \times 3}) \tag{1}$$

2. **Walk** to obtain samples $y^{(1)}, \ldots, y^{(N)}$ from p_Y using Langevin dynamics which conists of numerically solving the following Stochastic Differential Equation (SDE) (Figure 2b):

$$dy = v_y dt \tag{2}$$

$$dv_y = \nabla_y \log p_Y(y) dt - \gamma v_y dt + M^{-\frac{1}{2}} \sqrt{2} dB_t \tag{3}$$

where v_y represents the particle velocity, $\nabla_y \log p_Y(y)$ is the gradient of the log of the probability density function (called the score function) of p_Y, γ is friction, M is the mass, and B_t is the standard Wiener process in $N \times 3$-dimensions: $B_t \sim \mathcal{N}(0, t\mathbb{I}_{N\times 3})$. In practice, we employ the BAOAB solver (Appendix E) to integrate Equation 2 numerically.

3. **Jump** back to p_X to obtain samples $\hat{x}_1, \ldots, \hat{x}_N$ (Figure 2c) :

$$\hat{x}_i = \hat{x}(y_i) = \mathbb{E}[X \mid Y = y_i] \tag{4}$$

$\hat{x}(\cdot) \equiv \mathbb{E}[X \mid Y = \cdot]$ is called the denoiser. It corresponds to the minimizer (Appendix C) of the ℓ_2-loss between clean samples X and samples denoised back from $Y = X + \sigma\varepsilon$.

$$\hat{x}(\cdot) = \underset{f:\mathbb{R}^{N\times 3}\to\mathbb{R}^{N\times 3}}{\arg\min} \mathbb{E}_{\substack{X\sim p_X, \varepsilon\sim\mathcal{N}(0,\mathbb{I}_{N\times 3}) \\ Y = X+\sigma\varepsilon}} [\|f(Y) - X\|^2] \tag{5}$$

As shown by Robbins (1956); Miyasawa (1960) (and Appendix D), the denoiser $\hat{x}$ is closely linked to the score $\nabla_y \log p_Y$:

$$\hat{x}(y) = y + \sigma^2 \nabla_y \log p_Y(y) \tag{6}$$

Importantly, the score function $\nabla_y \log p_Y$ shows up in both the **walk** and **jump** steps, and we need to approximate this quantity.

3.3 Learning to Denoise

In order to run Walk-Jump Sampling as outlined above, we have the choice of modelling either the score $\nabla_y \log p_Y$ or the denoiser $\hat{x}$ as they are equivalent by Equation 6. Here, we follow the recommendations of Karras et al. (2022, 2024) which were originally developed for diffusion models in images. In particular, we model the denoiser as a neural network $\hat{x}_\theta(y, \sigma) \approx \hat{x}(y)$ parameterized by model parameters θ. We appropriately modify their construction for the point cloud context as detailed in Appendix B.

Note that while the normalization is applicable for any noise level, we only need to learn a model at a **single, fixed noise level** σ. This is unlike training diffusion and flow-matching models where a large range of noise levels are required for sampling. In particular, the choice of noise level σ for WJS is important because mode-mixing becomes faster as σ is increased, but the task asked of the denoiser becomes harder. We discuss the noise levels employed in this work in Section 5.

The denoiser $\hat{x}_\theta$ thus takes in noisy point clouds y and outputs clean point clouds $\hat{x}_\theta(y)$. To be precise, training the denoiser $\hat{x}_\theta$ consists of solving the following optimization problem:

$$\theta^* = \arg\min_\theta \mathbb{E}_{\substack{X\sim p_X, \varepsilon\sim\mathcal{N}(0,\mathbb{I}_{N\times 3}) \\ Y = X+\sigma\varepsilon}} \|\hat{x}_\theta(Y, \sigma) - X\|^2 \tag{7}$$

to obtain θ^*, the optimal model parameters. We approximate the expectation in Equation 7 by sampling $X \sim p_X$ and $\varepsilon \sim \mathcal{N}(0, \mathbb{I}_{N\times 3})$, as is standard in the empirical risk minimzation (ERM) (Vapnik, 1991) setting. We minimize the loss as a function of model parameters θ using the first-order optimizer Adam (Kingma & Ba, 2017) in PyTorch 2.0 (Ansel et al., 2024; Falcon & The PyTorch Lightning team, 2019).

3.4 Parametrization of the Denoiser Network

We describe the parametrization of the denoiser network $\hat{x}_\theta(y, \sigma)$ which will approximate $\hat{x}(y)$. While we fix a noise level σ here, we describe the general construction for an arbitrary noise level σ, which could be useful for training $E(3)$-equivariant and $SE(3)$-equivariant diffusion and flow-matching models as well.

We adapt the analysis and choices of Karras et al. (2022, 2024) for the point-cloud setting, leading to the following parametrization (Figure 3):

$$\hat{x}_\theta(y, \sigma) = c_{\text{skip}}(\sigma) y + c_{\text{out}}(\sigma) F_\theta(c_{\text{in}}(\sigma) y, c_{\text{noise}}(\sigma)) \tag{8}$$

where F_θ represents a learned network parameterized by θ. $c_{\text{skip}}(\sigma), c_{\text{out}}(\sigma), c_{\text{in}}(\sigma), c_{\text{noise}}(\sigma)$ are fixed functions from $\mathbb{R}^+$ to $\mathbb{R}$, chosen to normalize the effective inputs and outputs to F_θ. These coefficients are chosen to encourage re-use of the input y at low noise levels, but the opposite at high noise levels. We discuss their derivation in Appendix B.

Similar to Klein & Noé (2024), the initial embedding (Figure 4) computes:

- Edge-level Features: Edges are computed using a radial distance cutoff. The edge features are a concatenation of a one-hot bondedness feature and the radial distance embedded using Bessel functions.
- Atom-level features: Embedding of the atomic number (eg. C and N), and the atom name following PDB notation (eg. CA, CB for alpha and beta carbons).
- Residue-level Features: Embedding of the residue code (eg. ALA, CYS), and the sequence index of the associated residue (eg. $0, 1, \ldots$).

The atom-level and residue-level features leverage the known h feature of the peptide.

F_θ (Figure 5) is a geometric graph neural network (GNN) model similar to NequIP (Batzner et al., 2022; Thomas et al., 2018). Importantly, F_θ is chosen to be $SE(3)$-equivariant, in contrast to existing methods (Hoogeboom et al., 2022; Klein & Noé, 2024; Klein et al., 2024a) that utilize the $E(3)$-equivariant EGNN model (Satorras et al., 2022). As rightly pointed out by Dumitrescu et al. (2024), $E(3)$-equivariant models are equivariant under parity, which means that are forced to transform mirrored structures identically. When we experimented with $E(3)$-equivariant architectures, we found symmetric Ramachandran plots which arise from the unnecessary parity constraint of the denoising network. For this reason, Klein & Noé (2024); Klein et al. (2024a) use a 'chirality checker' to post-hoc fix the generated structures from their model; for JAMUN, such post-processing is unnecessary because our model can distinguish between chiral structures.

Our model for $\hat{x}_\theta$ is built with the `e3nn` library (Geiger & Smidt, 2022), and only contains $8.2\mathrm{M}$ parameters. We discuss the specific hyperparameters in Appendix H.

4 Datasets

First, we demonstrate our method on the uncapped amino acids used in Timewarp (Klein et al., 2024a), specifically the 2AA-large dataset consisting of 380 diamines split into 200 train, 80 validation and 100 test diamines. The molecules in this dataset have a single peptide bond between amino acids of varying identities. This results in a distribution well represented by two dihedral angles, the ϕ angle of the second residue and the ψ angle of the first residue. The termini are zwitterionic amino and carboxyl groups. These are not ideal analogues of amino acids in proteins due to local charge interactions as well as lack of steric effects. However, we run experiments on this dataset to directly compare JAMUN to the Transferable Boltzman Generator model (Klein & Noé, 2024). For the remainder of this paper, we will refer to this dataset as the uncapped-2AA dataset, where 2AA refers to two amino acid long polypeptides.

As is common in molecular dynamics simulations of very small peptides, we also generate a similar dataset with ACE (acetyl) and NME (N-methyl amide) caps. As illustrated in Figure 7, these caps introduce additional peptide bonds with the first and last residues. These peptide bonds remove the need for the zwitterion, while the methyl group provides some steric interactions. This results in a distribution which requires at least 4 dihedrals to be well-represented: the ϕ and ψ angles of both residues. These capping groups increase the complexity of the modelling task, but ensure more realistic distributions of conformations. For the remainder of this paper, we will refer to this dataset as the capped-2AA dataset, where 2AA refers to two amino acid long polypeptides.

We ensure that our unbiased molecular dynamics runs are converged or representative by comparing against biased molecular dynamics runs using Non-Equilibrium Umbrella Sampling (NEUS) (Dinner et al., 2018; Vani et al., 2022), a trajectory stratification based enhanced sampling algorithm. The protein is represented by the AMBER03 force field**?**. The simulations are performed at 300 K with the BAOAB integratorLeimkuhler & Matthews (2013) in OpenMMEastman et al. (2017); LINCS is used to constrain the lengths of bonds to hydrogen atomsHess et al. (1997); Particle Mesh Ewald is used to calculate electrostaticsDarden et al. (1993); the step size was 2 fs. The systems are solvated with TIP3P water models and equilibrated under NVT and NPT for 100ps each.

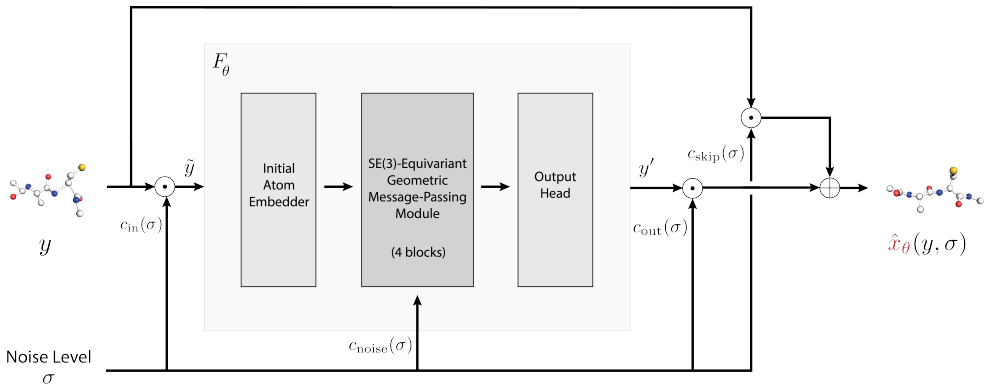


Figure 3: Overview of the denoiser network $\hat{x}_\theta$. The submodule F_θ sees input atom coordinates $\tilde{y} = c_{\text{in}}(\sigma)y$ and outputs predicted atom coordinates y', which gets scaled and added to a noise-conditional skip connection to finally obtain $\hat{x}_\theta(y)$.

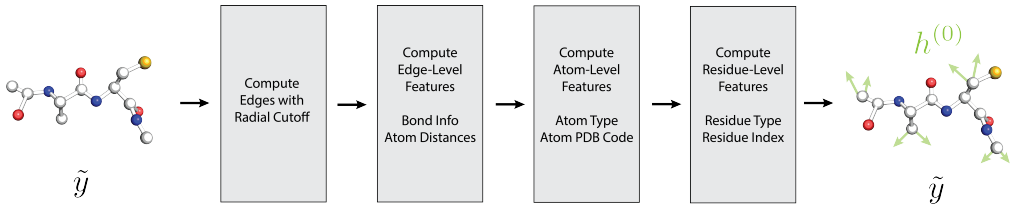


Figure 4: Overview of the initial embedder in the denoiser network, creating initial features $h^{(0)}$ at each atom and edge.

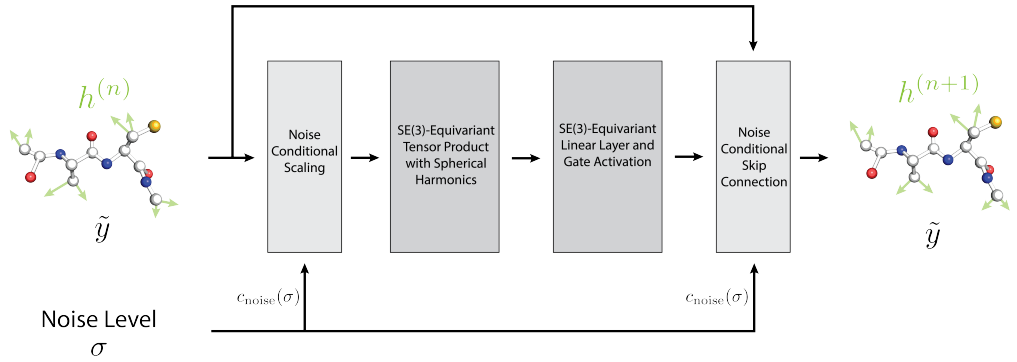


Figure 5: Overview of a single $SE(3)$-equivariant message-passing block (indexed by n) in the denoiser network. There are four such blocks iteratively updating the atom features from $h^{(0)}$ to $h^{(4)}$. The atom coordinates denoted by $\tilde{y} = c_{\text{in}}(\sigma)y$ (and hence, the edge features) are unchanged throughout these blocks.

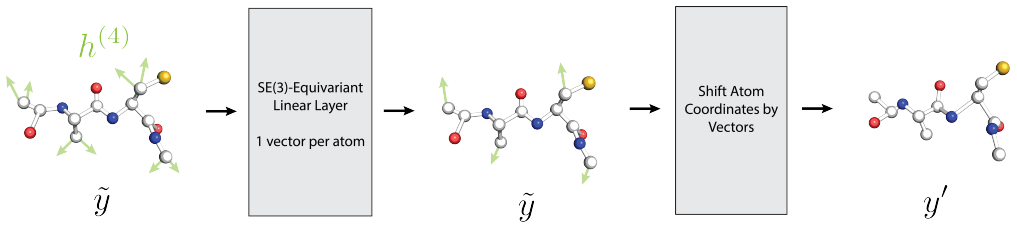


Figure 6: Overview of the output head, which predicts the coordinates $y' = F_\theta(c_{\text{in}}(\sigma)y, c_{\text{noise}}(\sigma))$.

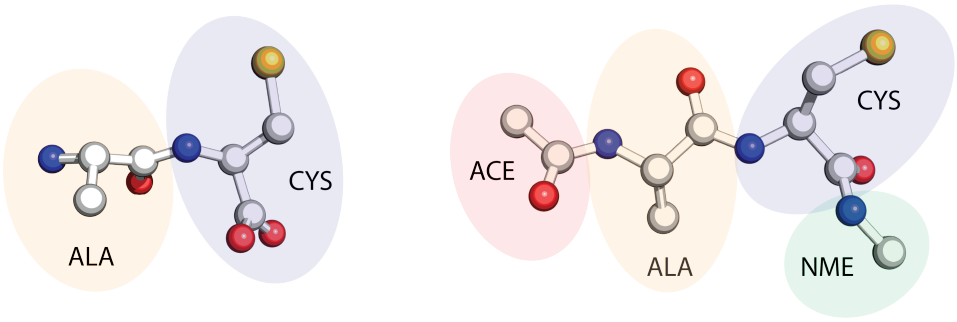


Figure 7: A side-by-side comparison of uncapped (left) compared to capped (right) ALA-CYS. The acetyl (ACE) and N-methyl (NME) capping groups provide steric hindrance and prevent local charge interactions on the N-terminal and C-terminal ends respectively.

We also explore the 4AA-large dataset as introduced by Timewarp (Klein et al., 2024a), which consists of 1459 train, 379 validation and 182 test uncapped-4AA compounds. The generalization task here is much harder, because the number of 4AA compounds in the training set is less than 1% of the total number of possible 4AA compounds (20^4).

5 Results

5.1 Capped and Uncapped 2AA Datasets

We train one model for each dataset, with $\sigma = 0.4$Å for the capped-2AA dataset and with $\sigma = 0.6$Å for the uncapped-2AA dataset. The same noise level is applied when training and sampling for all molecules of the dataset, although the optimal noise level can vary from dataset to dataset. This is what we mean by *universal*. This range of noise is in a sense characteristic to the molecular ensemble paradigm. This scale of noise is large enough to result in significant disruption of structure, leading to the smoothed Gaussian convolved 'walk' manifold. However, the scale is also small enough to avoid atoms 'swapping' position, for instance, or pairs of bonded atoms ending up very far from each other with reasonable probability.

Across both datasets, we find that JAMUN samples conformational ensembles of short peptides faster than conventional MD while remaining reasonably faithful to thermodynamics. Even though JAMUN is trained on all-atom coordinates, it generates remarkable diversity in ϕ and ψ dihedrals without explicit supervision. We also show a remarkable speedup relative to TBG (Klein & Noé, 2024). However, unlike the TBG models, we only have access to the score of the distribution, and cannot precisely reweight using the true probability, making JAMUN a Boltzmann emulator.

In Figure 8 and Figure 9 we show distributions of backbone $\phi-\psi$ torsion angles in samples generated using both molecular dynamics and JAMUN for eight test peptides in the challenging capped dataset. These peptides are chosen to represent the diversity of states as well as subtle perturbations from single-point amino acid changes in the dataset. As the molecular dynamics serves as our ground truth, we show the full dataset. However, for demonstrative purposes, we also show the distribution when it has been run for the same wall-clock GPU time as JAMUN requires to sample 640000 points. This number was chosen to reflect reasonable sampling for validation capped-2AA molecules. This can be variable, for the specific 8 capped-2AA molecules we choose here, the sampling time with JAMUN is shown in Table 1. This corresponds to a range of 100ps to 300ps of MD simulation time (depending on the capped-2AA molecule) performed within an hour of sampling with JAMUN.

These results clearly show that JAMUN is sampling the conformational ensembles of unseen peptides with high accuracy. The vast majority of low-energy basins in the Ramachandran plot are captured by the model. Moreover, JAMUN is sampling these states much faster than conventional MD. Comparing the 'JAMUN" and 'Benchmark Molecular Dynamics' columns of Figure 8 and Figure 9 shows that MD lags dramatically behind. At the point where JAMUN has sampled the entire distribution, MD is often still stuck sampling a single basin.

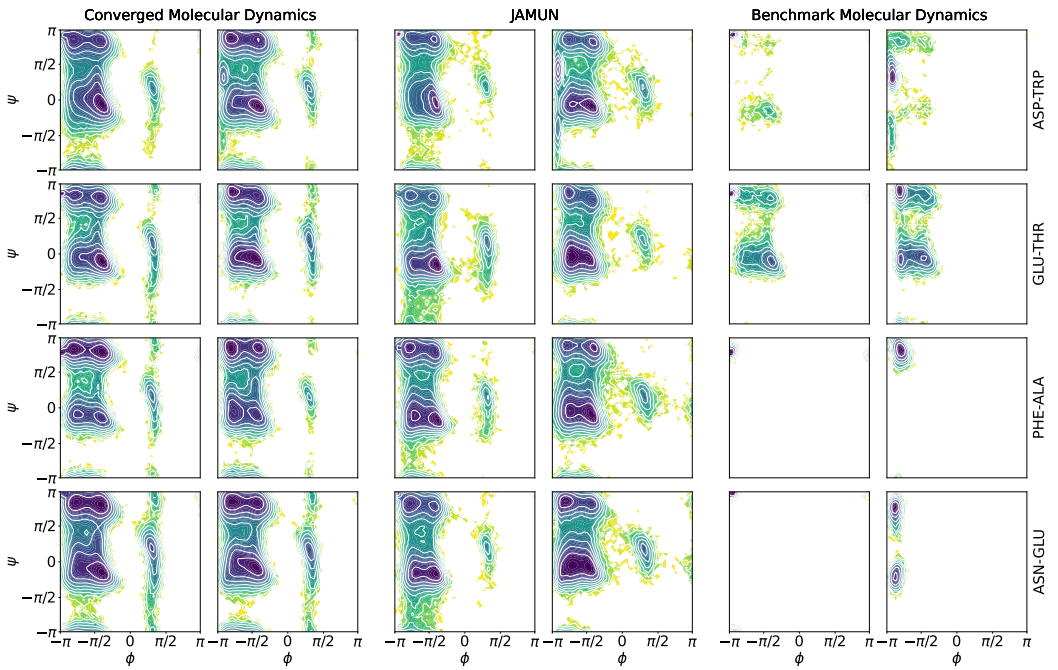


Figure 8: Histograms (in negative log scale, yellow indicating low probability to blue indicating high probability) of backbone $\phi - \psi$ torsion angles for four example capped-2AA from the test set. Each pair of Ramachandran plots show the distribution for a single capped-2AA, with each row corresponding to capped-2AA identity. The first column shows samples from the fully converged MD distribution. The second shows 640000 samples from JAMUN. The third column shows a distribution obtained from an MD trajectory run for the same wall-clock time as JAMUN.

Capped-2AA	**Total Sampling Time** 640000 samples	**Time per Sample**
ASP-TRP	38 min	3.56 ms
GLU-THR	27 min	2.53 ms
PHE-ALA	28 min	2.62 ms
ASN-GLU	29 min	2.71 ms
CYS-TRP	34 min	3.18 ms
GLY-ASN	22 min	2.06 ms
HIS-PRO	30 min	2.81 ms
ILE-GLY	22 min	2.06 ms

Table 1: Sampling times for 640000 samples from JAMUN of each capped-2AA molecule, as measured on a NVIDIA RTX A100 GPU.

While the capped dataset represents higher diversity and fidelity to true peptide thermodynamics, for a fair comparison, we use the uncapped-2AA dataset to compare against the TBG model (Klein & Noé, 2024). Note that our hyperparameters are optimized for the capped-2AA dataset. Nevertheless, we show a significant improvement in sampling, as visualized in Figure 10. We simulate both models for an equal amount of compute-time, including post-processing steps such as topology and chirality checks for TBG. By visual inspection, we see that JAMUN samples all states in the unseen test peptides, whereas the TBG models misses a basin in the given time.

As a convergence metric, we look at Jenson-Shannon divergence between binned torsion angles in Figure 11. This allows us to compare the rate at which JAMUN and MD converge to the 'correct' distribution. We calculate the Jenson-Shannon divergence for molecular dynamics and JAMUN runs of equal GPU time (using the data generated for Figure 8 and Figure 9). We also analyze fully converged molecular dynamics runs for reference. It appears that while JAMUN converges quicker

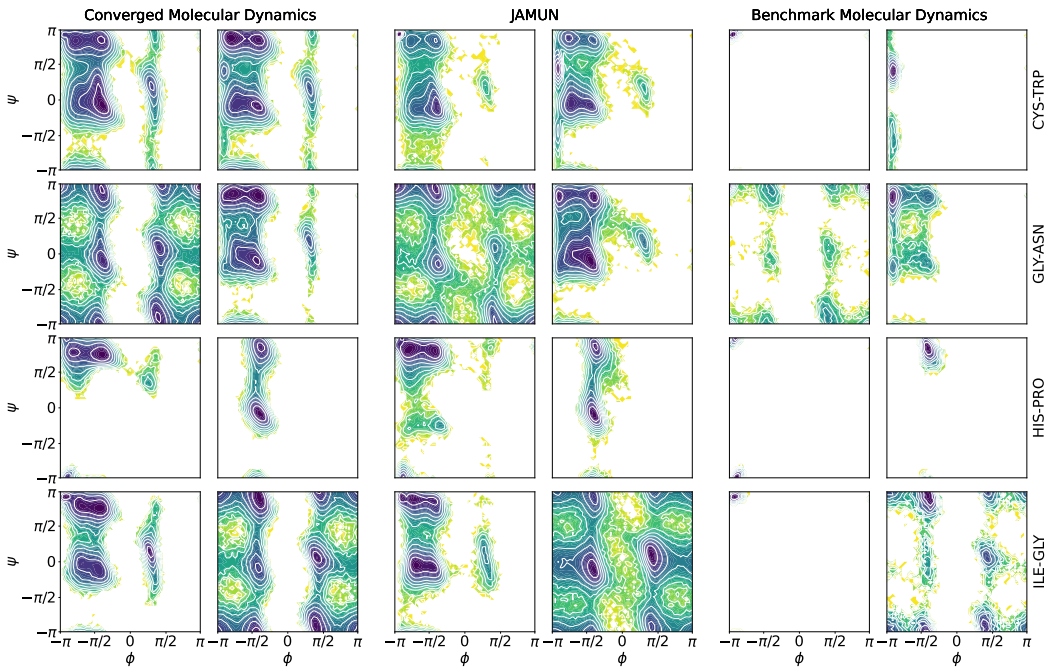


Figure 9: Histograms (in negative log scale, yellow indicating low probability to blue indicating high probability) of backbone $\phi - \psi$ torsion angles for four other example capped-2AA from the test set. Each pair of Ramachandran plots show the distribution for a single capped-2AA, with each row corresponding to capped-2AA identity. The first column shows the fully converged molecular dynamics distribution. The second shows 640000 samples from JAMUN. The third column shows a distribution obtained from an MD trajectory run for the exact wall-clock time as JAMUN.

than MD, and samples relevant states faster, it does not ever converge to the same accuracy. It is likely that part of the discrepency is from JAMUN's oversampling of rare and transition regions. In particular, we see from the Ramachandran plots that occasionally the standard transition paths are not the most frequent and instead spurious transition paths are sampled along. JAMUN is not trained on kinetics and cannot faithfully reproduce them. This is a well-understood phenomena in stochastic processes in a rough to smooth surface (Zwanzig, 1988). It is also evident from the smooth nature of JAMUN's Jenson-Shannon time series, as compared to the step-wise behavior of MD that JAMUN has a more natural mixing of states without kinetic traps. However, much of the spurious sampling can be avoided with smaller noise values at the cost of sampling efficiency. As such, this Langevin dynamics on a noised surface provides an interesting avenue of research from a stochastic processes point of view.

From a biological and drug-design standpoint, we are relatively uninterested in kinetics or even the exact distribution. However, we are extremely interested in the sampling of diverse metastable states with some sense of their relative stabilities. To this end, JAMUN does an impressive job of sampling metastable states particular to specific 2AA species with thermodynamic fidelity. We demonstrate this with the use of Markov State Models trained on converged MD data in Appendix A. The figures demonstrate that across the test set of capped peptides, JAMUN (i) captures the precise shape of states extremely well, and (ii) samples nuanced states that are not obvious by eye in a Ramachandran plot with high accuracy.

5.2 Uncapped 4AA Dataset

We also trained another JAMUN model on the uncapped-4AA dataset, as discussed in Section 4. We sample 50000 structures from JAMUN on the unseen 4AA AMIG, which is *one tenth* of the length of the reference MD trajectory for AMIG in Timewarp.

We see that JAMUN is able to precisely match the distribution-level statistics of the true MD trajectory, in terms of backbone torsion angles (Figure 13), side-chain torsion angles (Figure 14) and

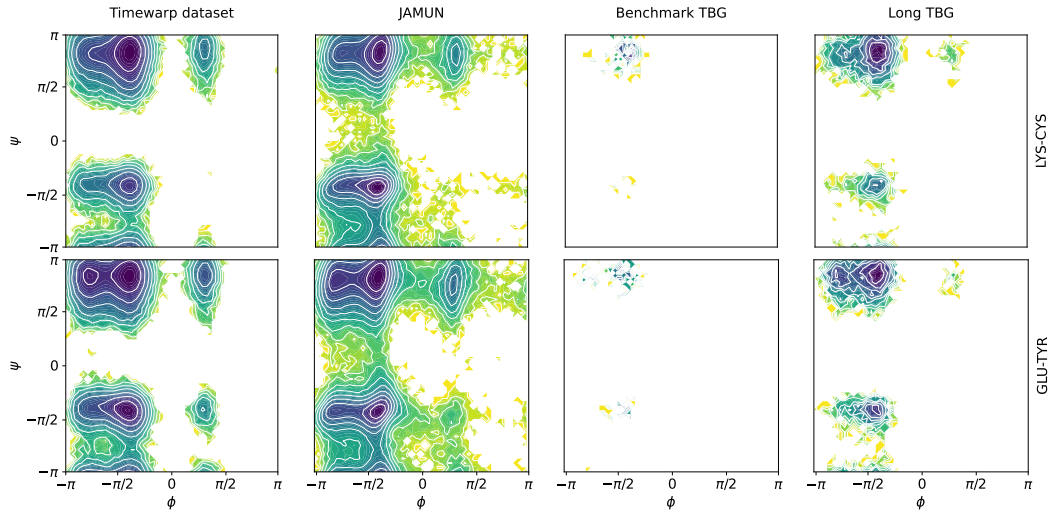


Figure 10: Histograms (in negative log scale, yellow indicating low probability to blue indicating high probability) of backbone $\phi - \psi$ torsion angles from two example uncapped-2AA from the uncapped test set. Each Ramachandran plot shows the distribution for a single uncapped-2AA, with each row corresponding to uncapped-2AA identity. The first column shows the full distribution from the Timewarp dataset. The second shows 640000 samples from JAMUN. The third column shows a distribution obtained from the TBG model run for the exact same amount of time as the JAMUN sampler. The fourth shows results from the TBG model run for 5000 samples (run for roughly 10 times as long as JAMUN).

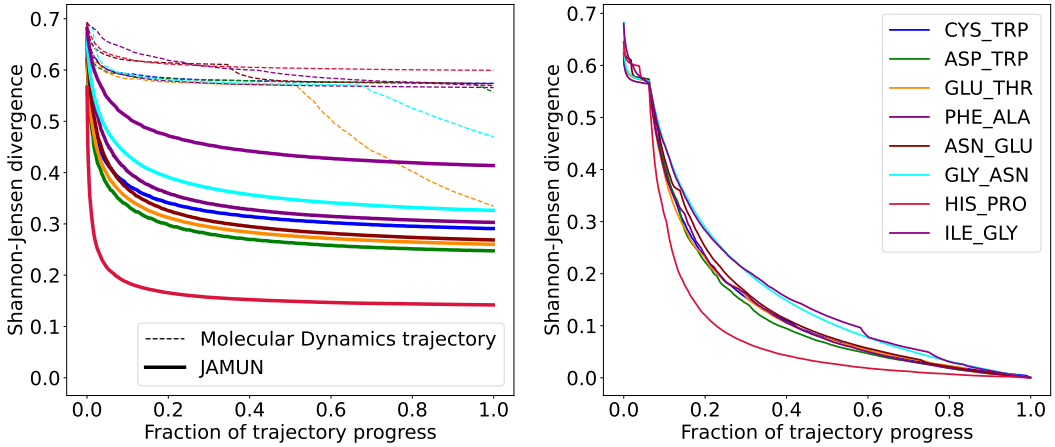


Figure 11: Jensen-Shannon divergences plotted along time. (Left) A comparison between JAMUN and MD where the x-axis is a trajectory progress coordinate such that all JAMUN runs are for 640000 samples while MD runs are truncated to be the same GPU time. (Right) Jenson-Shannon divergence of a long fully converged MD trajectory.

time-lagged independent component analysis (TICA) projections (Figure 12). Results were obtained using PyEMMA (Scherer et al., 2015), with help from the analysis scripts of MDGen (Jing et al., 2024b).

We tried running the TBG (Klein & Noé, 2024) model on 4AA compounds, but could not do so due to a shape mismatch error, indicating that the model does not support sampling 4AA compounds at this time.

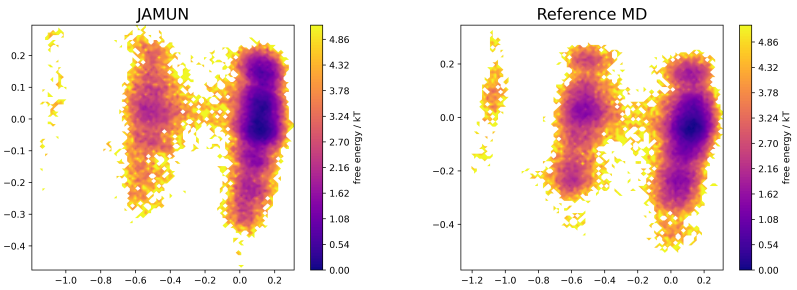


Figure 12: TICA projections computed for JAMUN samples (left) and the reference MD trajectory (right) for the AMIG compound. TICA eigenvectors for both projections were computed once, from the reference MD trajectory.

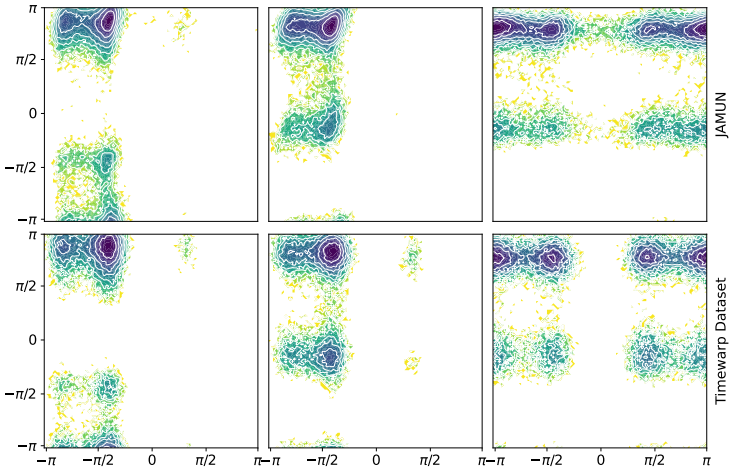


Figure 13: Ramachandran plots for each pair of dihedral angles for JAMUN samples (top row) and the reference MD trajectory (bottom row) for the AMIG compound.

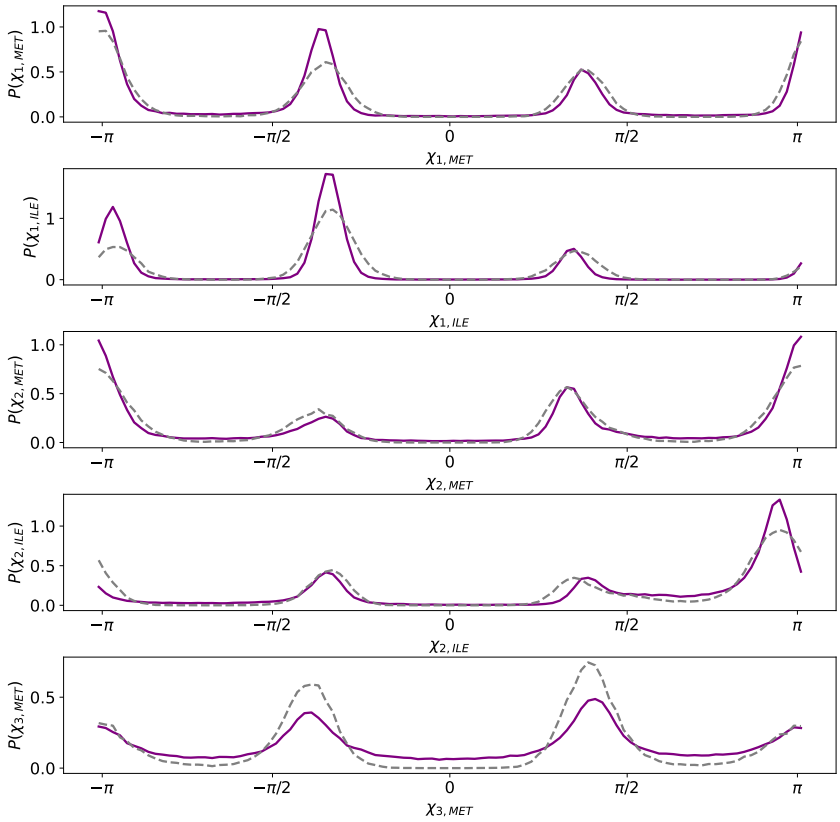


Figure 14: One-dimensional histogram of descriptors for JAMUN samples (purple) and the reference timewarp data (great) for the AMIG peptide.

6 Conclusion

We present JAMUN, a Walk-Jump Sampling model for generating ensembles of molecular conformations. The model is trained on molecular dynamics data from two amino acid peptides and is transferable to peptides outside of its training set. It produces accurate conformational ensembles faster than MD or previous ML approaches. This represents an important first step toward the ultimate goal of a transferable generative model for protein conformational ensembles.

The model has some limitations that motivate future work. While it is highly transferable in the space of two amino acid peptides, we have not tested the model's ability to transfer to larger proteins. We leave exploring this important direction to future work. Additionally, while the current $SE(3)$-equivariant denoiser architecture works well, further development of the denoising network could speed up sampling. Alternative jump methods, such as multiple denoising steps (à la diffusion), could also serve to sharpen generation. Lastly, a promising direction that has not yet been explored is the application of classical enhanced sampling methods, such as metadynamics, for traversing the noisy space.

JAMUN is the first model to use Walk-Jump Sampling on point clouds, and the first to apply Walk-Jump Sampling to molecular dynamics data. This paradigm gives the model a clear physics interpretation. By adding noise, JAMUN is able to simulate on a smoother manifold than ordinary MD. Sampling on this smooth surface enables the model to generate accurate molecular conformational ensembles faster than both MD and previous ML models.

References

Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, Geeta Chauhan, Anjali Chourdia, Will Constable, Alban Desmaison, Zachary DeVito, Elias Ellison, Will Feng, Jiong Gong, Michael Gschwind, Brian Hirsh, Sherlock Huang, Kshiteej Kalambarkar, Laurent Kirsch, Michael Lazos, Mario Lezcano, Yanbo Liang, Jason Liang, Yinghai Lu, CK Luk, Bert Maher, Yunjie Pan, Christian Puhrsch, Matthias Reso, Mark Saroufim, Marcos Yukio Siraichi, Helen Suk, Michael Suo, Phil Tillet, Eikan Wang, Xiaodong Wang, William Wen, Shunting Zhang, Xu Zhao, Keren Zhou, Richard Zou, Ajit Mathews, Gregory Chanan, Peng Wu, and Soumith Chintala. PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation. In *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. ACM, April 2024. doi: 10.1145/3620665.3640366. URL `https://pytorch.org/assets/pytorch2-2.pdf`.

Akashnathan Aranganathan, Xinyu Gu, Dedi Wang, Bodhi Vani, and Pratyush Tiwary. Modeling Boltzmann weighted structural ensembles of proteins using AI based methods. 2024.

Marloes Arts, Victor Garcia Satorras, Chin-Wei Huang, Daniel Zugner, Marco Federici, Cecilia Clementi, Frank Noé, Robert Pinsler, and Rianne van den Berg. Two for one: Diffusion models and force fields for coarse-grained molecular dynamics. *Journal of Chemical Theory and Computation*, 19(18):6151–6159, 2023.

Simon Batzner, Albert Musaelian, Lixin Sun, Mario Geiger, Jonathan P. Mailoa, Mordechai Kornbluth, Nicola Molinari, Tess E. Smidt, and Boris Kozinsky. E(3)-equivariant graph neural networks for data-efficient and accurate interatomic potentials. *Nature Communications*, 13 (1):2453, May 2022. ISSN 2041-1723. doi: 10.1038/s41467-022-29939-5. URL `https://doi.org/10.1038/s41467-022-29939-5`.

Helen M. Berman, John Westbrook, Zukang Feng, Gary Gilliland, T. N. Bhat, Helge Weissig, Ilya N. Shindyalov, and Philip E. Bourne. The Protein Data Bank. *Nucleic Acids Research*, 28(1):235–242, 01 2000. ISSN 0305-1048. doi: 10.1093/nar/28.1.235. URL `https://doi.org/10.1093/nar/28.1.235`.

David W Borhani and David E Shaw. The future of molecular dynamics simulations in drug discovery. *Journal of computer-aided molecular design*, 26:15–26, 2012.

Gregory R Bowman. AlphaFold and Protein Folding: Not Dead Yet! The Frontier Is Conformational Ensembles. *Annual Review of Biomedical Data Science*, 7, 2024.

Giorgio Colombo. Computing allostery: from the understanding of biomolecular regulation and the discovery of cryptic sites to molecular design. *Current Opinion in Structural Biology*, 83:102702, 2023.

Tom Darden, Darrin York, and Lee Pedersen. Particle mesh ewald: An n log(n) method for ewald sums in large systems. *The Journal of Chemical Physics*, 98(12):10089–10092, June 1993. doi: 10.1063/1.464397. URL https://doi.org/10.1063/1.464397.

Aaron R Dinner, Jonathan C Mattingly, Jeremy O B Tempkin, Brian Van Koten, and Jonathan Weare. Trajectory stratification of stochastic dynamics. *SIAM Rev Soc Ind Appl Math*, 60(4):909–938, November 2018.

Alexandru Dumitrescu, Dani Korpela, Markus Heinonen, Yogesh Verma, Valerii Iakovlev, Vikas Garg, and Harri Lähdesmäki. Field-based Molecule Generation, 2024. URL https://arxiv.org/abs/2402.15864.

Peter Eastman, Jason Swails, John D. Chodera, Robert T. McGibbon, Yutong Zhao, Kyle A. Beauchamp, Lee-Ping Wang, Andrew C. Simmonett, Matthew P. Harrigan, Chaya D. Stern, Rafal P. Wiewiora, Bernard R. Brooks, and Vijay S. Pande. OpenMM 7: Rapid development of high performance algorithms for molecular dynamics. *PLOS Computational Biology*, 13(7): e1005659, July 2017. doi: 10.1371/journal.pcbi.1005659. URL https://doi.org/10.1371/journal.pcbi.1005659.

William Falcon and The PyTorch Lightning team. PyTorch Lightning, March 2019. URL https://github.com/Lightning-AI/lightning.

Nathan C Frey, Daniel Berenberg, Karina Zadorozhny, Joseph Kleinhenz, Julien Lafrance-Vanasse, Isidro Hotzel, Yan Wu, Stephen Ra, Richard Bonneau, Kyunghyun Cho, et al. Protein discovery with discrete walk-jump sampling. *arXiv preprint arXiv:2306.12360*, 2023.

Mario Geiger and Tess Smidt. e3nn: Euclidean neural networks. *arXiv preprint arXiv:2207.09453*, 2022.

Nancy R Gough and Charalampos G Kalodimos. Exploring the conformational landscape of protein kinases. *Current Opinion in Structural Biology*, 88:102890, 2024.

Katherine Henzler-Wildman and Dorothee Kern. Dynamic personalities of proteins. *Nature*, 450 (7172):964–972, December 2007.

Berk Hess, Henk Bekker, Herman JC Berendsen, and Johannes GEM Fraaije. Lincs: a linear constraint solver for molecular simulations. *Journal of computational chemistry*, 18(12):1463–1472, 1997.

Emiel Hoogeboom, Victor Garcia Satorras, Clément Vignac, and Max Welling. Equivariant Diffusion for Molecule Generation in 3D, 2022. URL https://arxiv.org/abs/2203.17003.

Tim Hsu, Babak Sadigh, Vasily Bulatov, and Fei Zhou. Score dynamics: Scaling molecular dynamics with picoseconds time steps via conditional diffusion model. *Journal of Chemical Theory and Computation*, 20(6):2335–2348, 2024.

Bowen Jing, Bonnie Berger, and Tommi Jaakkola. AlphaFold meets flow matching for generating protein ensembles. *arXiv preprint arXiv:2402.04845*, 2024a.

Bowen Jing, Hannes Stärk, Tommi Jaakkola, and Bonnie Berger. Generative modeling of molecular dynamics trajectories, 2024b. URL https://arxiv.org/abs/2409.17808.

W. Kabsch. A solution for the best rotation to relate two sets of vectors. *Acta Crystallographica Section A*, 32(5):922–923, Sep 1976. doi: 10.1107/S0567739476001873. URL https://doi.org/10.1107/S0567739476001873.

Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the Design Space of Diffusion-Based Generative Models. In *Proc. NeurIPS*, 2022.

Tero Karras, Miika Aittala, Jaakko Lehtinen, Janne Hellsten, Timo Aila, and Samuli Laine. Analyzing and Improving the Training Dynamics of Diffusion Models. In *Proc. CVPR*, 2024.

Stefanie Kieninger and Bettina G. Keller. GROMACS Stochastic Dynamics and BAOAB Are Equivalent Configurational Sampling Algorithms. *Journal of Chemical Theory and Computation*, 18 (10):5792–5798, 2022.

Joseph C Kim, David Bloore, Karan Kapoor, Jun Feng, Ming-Hong Hao, and Mengdi Wang. Scalable normalizing flows enable boltzmann generators for macromolecules. *arXiv preprint arXiv:2401.04246*, 2024.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. URL `https://arxiv.org/abs/1412.6980`.

Leon Klein and Frank Noé. Transferable Boltzmann Generators. *arXiv preprint arXiv:2406.14426*, 2024.

Leon Klein, Andrew Foong, Tor Fjelde, Bruno Mlodozeniec, Marc Brockschmidt, Sebastian Nowozin, Frank Noé, and Ryota Tomioka. Timewarp: Transferable acceleration of molecular dynamics by learning time-coarsened dynamics. *Advances in Neural Information Processing Systems*, 36, 2024a.

Leon Klein, Andreas Krämer, and Frank Noé. Equivariant flow matching. *Advances in Neural Information Processing Systems*, 36, 2024b.

Ben Leimkuhler and Charles Matthews. *Molecular Dynamics: With Deterministic and Stochastic Numerical Methods*. Number 39 in Interdisciplinary Applied Mathematics. Springer International Publishing : Imprint: Springer, Cham, 1st ed. 2015 edition, 2015. ISBN 978-3-319-16375-8.

Benedict Leimkuhler and Charles Matthews. Rational Construction of Stochastic Numerical Methods for Molecular Sampling. *Applied Mathematics Research eXpress*, 2013(1):34–56, June 2012. ISSN 1687-1200. doi: 10.1093/amrx/abs010. URL `https://doi.org/10.1093/amrx/abs010`. _eprint: https://academic.oup.com/amrx/article-pdf/2013/1/34/397230/abs010.pdf.

Benedict Leimkuhler and Charles Matthews. Robust and efficient configurational molecular sampling via langevin dynamics. *The Journal of Chemical Physics*, 138(17):174102, May 2013. doi: 10.1063/1.4802990. URL `https://doi.org/10.1063/1.4802990`.

Mitchell D Miller and George N Phillips. Moving beyond static snapshots: Protein dynamics and the Protein Data Bank. *Journal of Biological Chemistry*, 296, 2021.

Koichi Miyasawa. An Empirial Bayes Estimator of the Mean of a Normal Population. *Bulletin de l'Institut international de statistique.*, 38(4):181–188, 1960.

Frank Noé, Simon Olsson, Jonas Köhler, and Hao Wu. Boltzmann generators: Sampling equilibrium states of many-body systems with deep learning. *Science*, 365(6457):eaaw1147, 2019.

Vijay S Pande, Kyle Beauchamp, and Gregory R Bowman. Everything you wanted to know about markov state models but were afraid to ask. *Methods*, 52(1):99–105, June 2010.

Pedro O Pinheiro, Arian Jamasb, Omar Mahmood, Vishnu Sresht, and Saeed Saremi. Structure-based Drug Design by Denoising Voxel Grids. *arXiv preprint arXiv:2405.03961*, 2024a.

Pedro O Pinheiro, Joshua Rackers, Joseph Kleinhenz, Michael Maser, Omar Mahmood, Andrew Watkins, Stephen Ra, Vishnu Sresht, and Saeed Saremi. 3D molecule generation by denoising voxel grids. *Advances in Neural Information Processing Systems*, 36, 2024b.

Herbert Robbins. An Empirical Bayes Approach to Statistics. In *Proceedings of the Third Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics*, volume 3.1, 1956.

Matthias Sachs, Benedict Leimkuhler, and Vincent Danos. Langevin dynamics with variable coefficients and nonconservative forces: from stationary states to numerical methods. *Entropy*, 19(12): 647, 2017.

Saeed Saremi and Aapo Hyvärinen. Neural Empirical Bayes. *Journal of Machine Learning Research*, 20(181):1–23, 2019.

Victor Garcia Satorras, Emiel Hoogeboom, and Max Welling. E(n) Equivariant Graph Neural Networks, 2022. URL https://arxiv.org/abs/2102.09844.

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, 2015. doi: 10.1021/acs.jctc.5b00743. URL https://doi.org/10.1021/acs.jctc.5b00743. PMID: 26574340.

Mathias Schreiner, Ole Winther, and Simon Olsson. Implicit transfer operator learning: multiple time-resolution surrogates for molecular dynamics. *arXiv preprint arXiv:2305.18046*, 2023.

Nathaniel Thomas, Tess Smidt, Steven Kearnes, Lusann Yang, Li Li, Kai Kohlhoff, and Patrick Riley. Tensor field networks: Rotation-and translation-equivariant neural networks for 3d point clouds. *arXiv preprint arXiv:1802.08219*, 2018.

S. Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(4):376–380, 1991. doi: 10.1109/34.88573.

Bodhi P Vani, Jonathan Weare, and Aaron R Dinner. Computing transition path theory quantities with trajectory stratification. *J Chem Phys*, 157(3):034106, July 2022.

V. Vapnik. Principles of risk minimization for learning theory. In J. Moody, S. Hanson, and R.P. Lippmann (eds.), *Advances in Neural Information Processing Systems*, volume 4. Morgan-Kaufmann, 1991. URL https://proceedings.neurips.cc/paper_files/paper/1991/file/ff4d5fbbafdf976cfdc032e3bde78de5-Paper.pdf.

Andreas Vitalis and Rohit V Pappu. Methods for monte carlo simulations of biomacromolecules. *Annual reports in computational chemistry*, 5:49–76, 2009.

Li-E Zheng, Shrishti Barethiya, Erik Nordquist, and Jianhan Chen. Machine learning generation of dynamic protein conformational ensembles. *Molecules*, 28(10):4047, 2023.

Shuxin Zheng, Jiyan He, Chang Liu, Yu Shi, Ziheng Lu, Weitao Feng, Fusong Ju, Jiaxi Wang, Jianwei Zhu, Yaosen Min, et al. Predicting equilibrium distributions for molecular systems with deep learning. *Nature Machine Intelligence*, pp. 1–10, 2024.

Robert Zwanzig. Diffusion in a rough potential. *Proceedings of the National Academy of Sciences*, 85(7):2029–2030, 1988.

A Distribution analyses

We demonstrate the quantitative gain in sampling through metrics like the Jenson-Shannon divergence and the clear qualitative gain in sampling diversity. Here we show this visually by employing Markov State Models (MSMs) (Pande et al., 2010) to characterize metastability. MSMs are the most commonly used clustering algorithm for times series in molecular dynamics. They function by first projecting on a low-dimensional manifold, most commonly the Time-lagged Independent Component Analysis (TICA), finely clustering the samples and then solving for the eigenvalues of a transition probability matrix. Here, we use the PYEMMA2 (Scherer et al., 2015) implementation. We use a lag time of ten steps, five tics, 200 k-means clusters to obtain 10 macrostates for the unbiased well-sampled molecular dynamics trajectories. We perform an implied timescales analysis to ensure that these states are a reasonable representation. We can then use this representation to better understand the features of our space and qualitatively evaluate JAMUN. In Figure 15 to Figure 22, we plot PMFs (potential of mean force) in grayscale for JAMUN's samples with the Markov State Model histogram superimposed in color. Each pair of Ramachandran plots correspond to a single metastable state.

In particular, note that oblong metastable basins corresponding to the $-\pi$ region in ϕ, small metastability in the $(-\pi, \pi)$ area and even the specific shapes of commonly occuring basins can be signatures particular to specific 2AA molecules and are well-emulated by JAMUN.

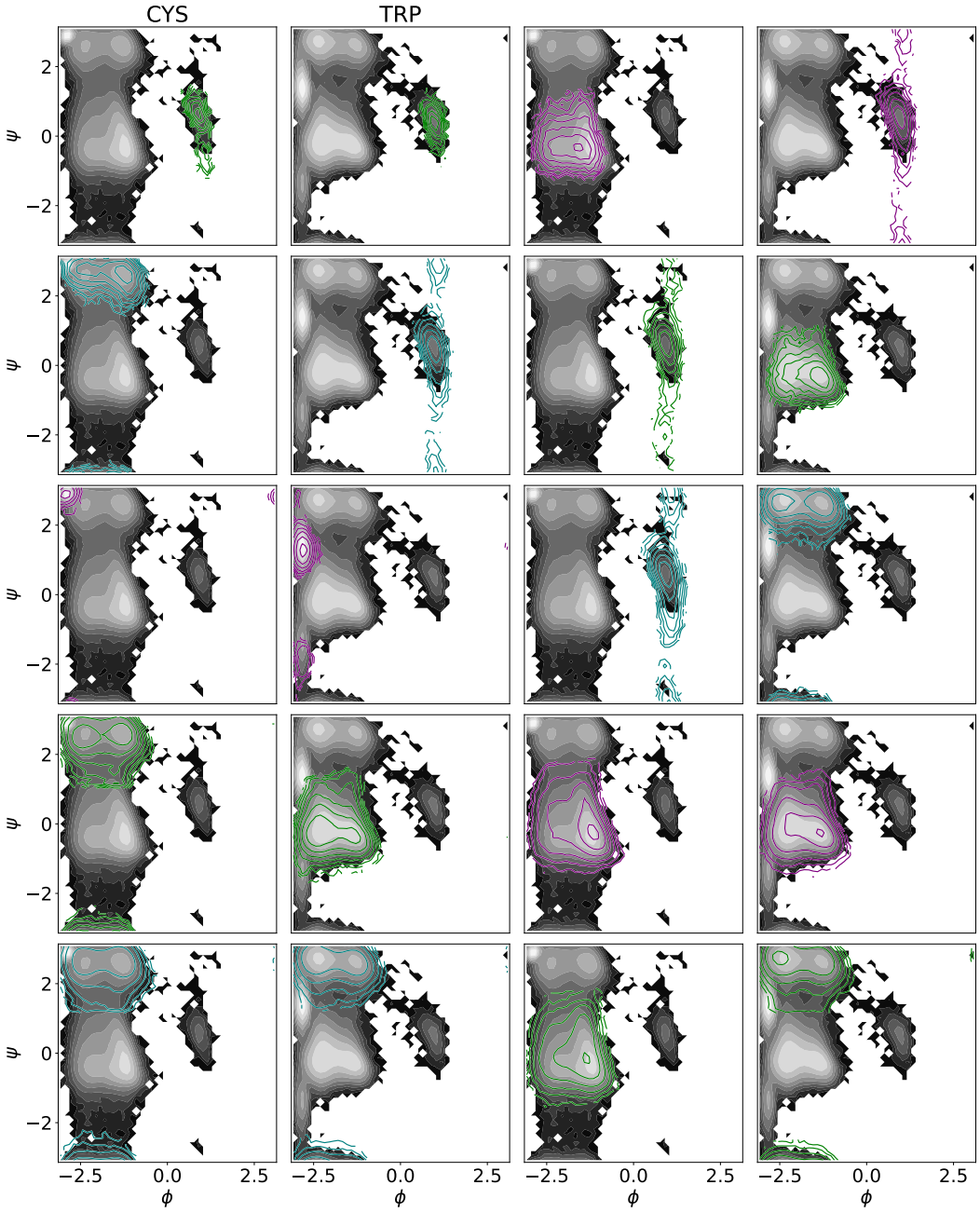


Figure 15: Markov states superimposed on sampled PMFs for CYS-TRP. This figure shows the results of running JAMUN with Markov state model negative log histograms superimposed on them. The MSMs are trained on fully converged MD data and represent metastable states. Each pair of Ramachandran plots correspond to a single metastable basin. Note that the gray-scale represents JAMUN samples while the colored is from molecular dynamics.

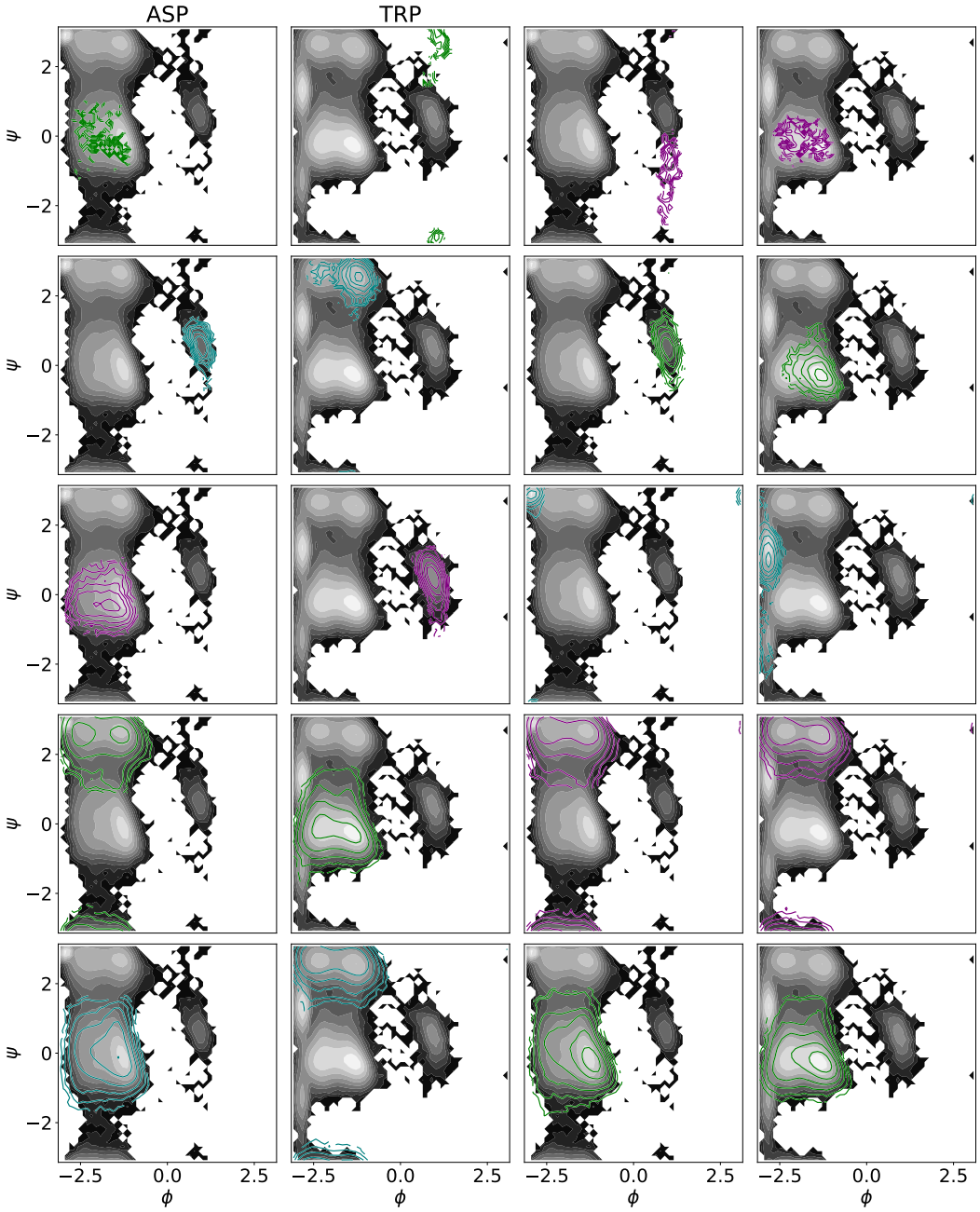


Figure 16: Markov states superimposed on sampled PMFs for ASP-THR. This figure shows the results of running JAMUN with Markov state model negative log histograms superimposed on them. The MSMs are trained on fully converged MD data and represent metastable states. Each pair of Ramachandran plots correspond to a single metastable basin. Note that the gray-scale represents JAMUN samples while the colored is from molecular dynamics.

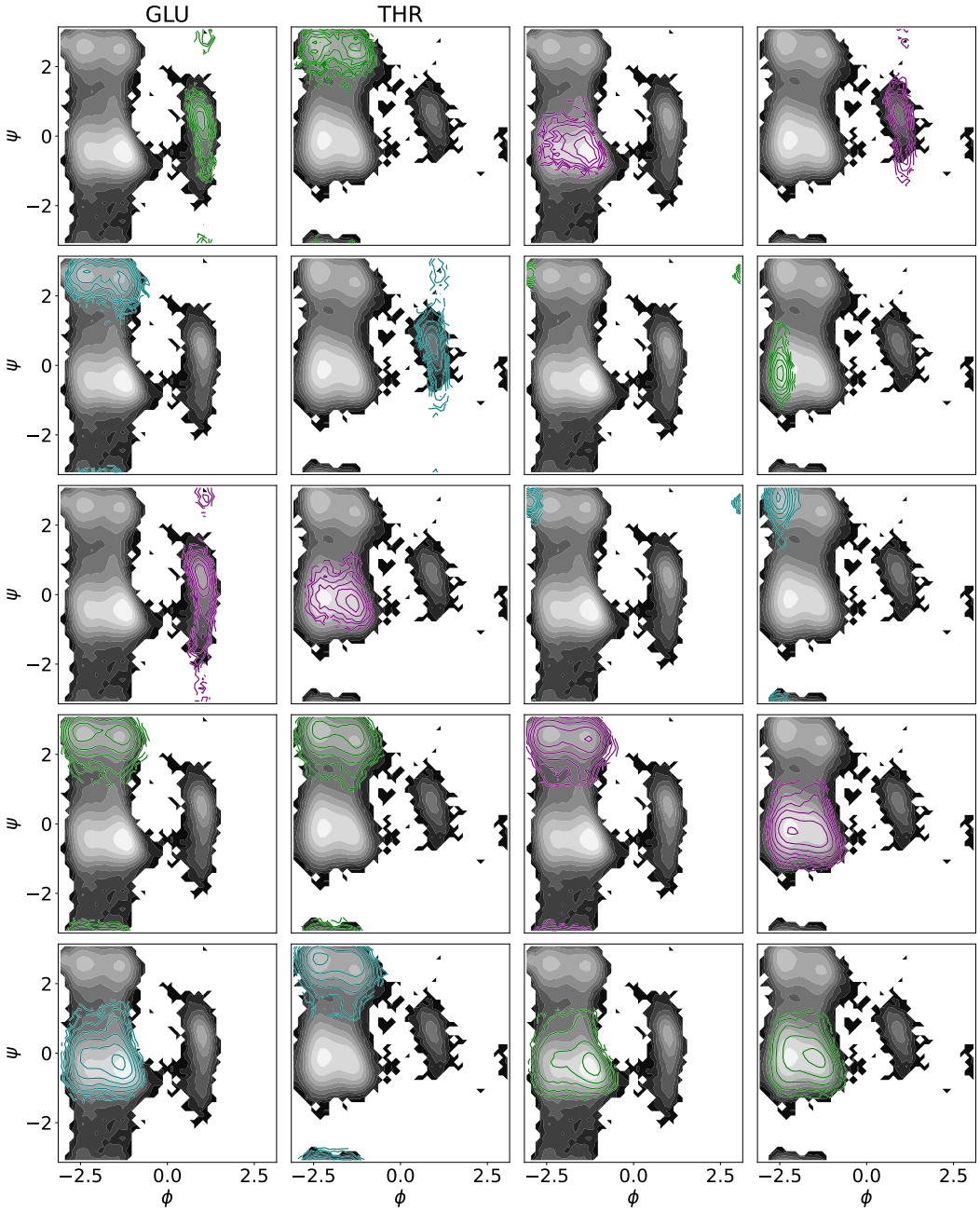


Figure 17: Markov states superimposed on sampled PMFs for GLU-THR. This figure shows the results of running JAMUN with Markov state model negative log histograms superimposed on them. The MSMs are trained on fully converged MD data and represent metastable states. Each pair of Ramachandran plots correspond to a single metastable basin. Note that the gray-scale represents JAMUN samples while the colored is from molecular dynamics.

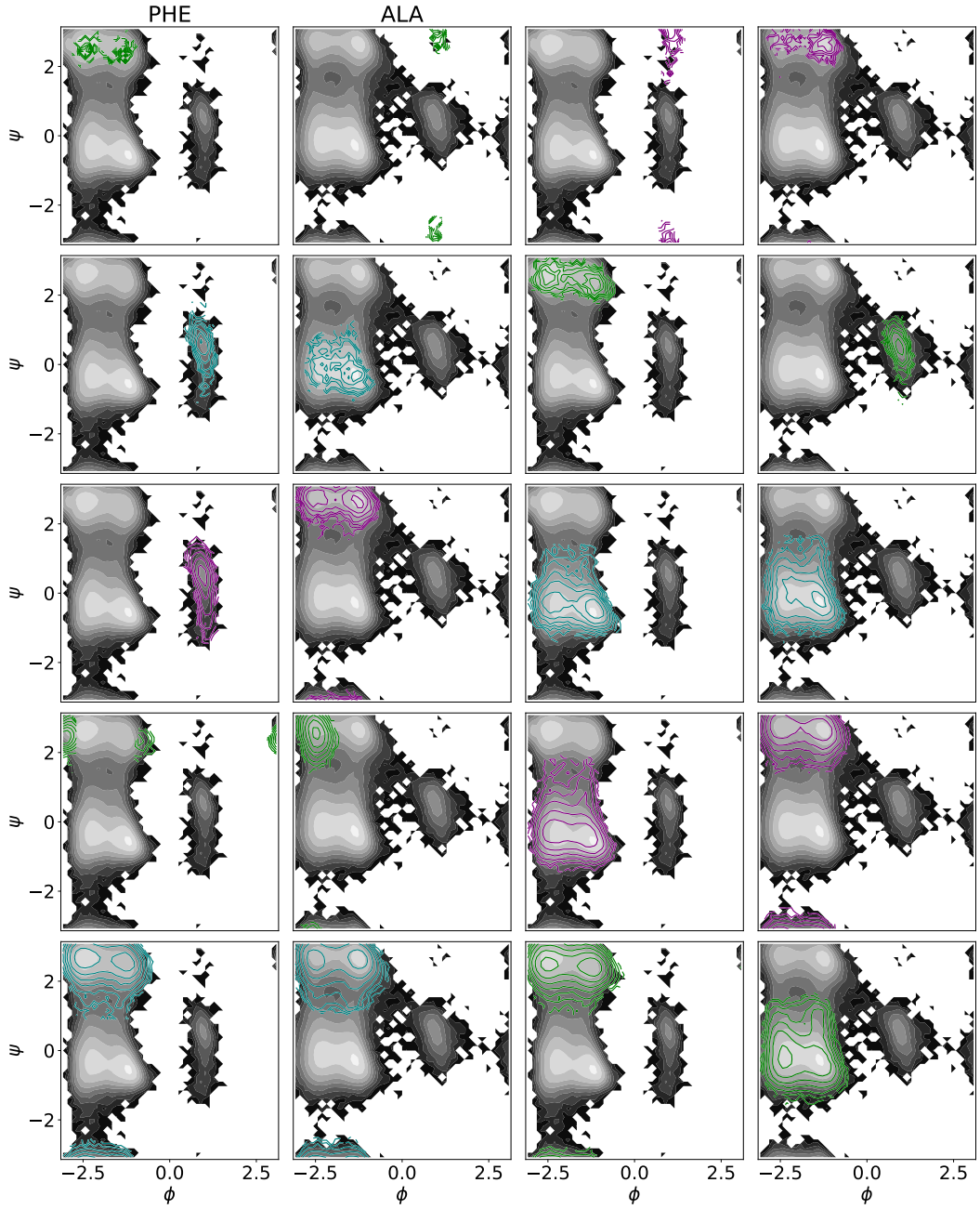


Figure 18: Markov states superimposed on sampled PMFs for PHE-ALA. This figure shows the results of running JAMUN with Markov state model negative log histograms superimposed on them. The MSMs are trained on fully converged MD data and represent metastable states. Each pair of Ramachandran plots correspond to a single metastable basin. Note that the gray-scale represents JAMUN samples while the colored is from molecular dynamics.

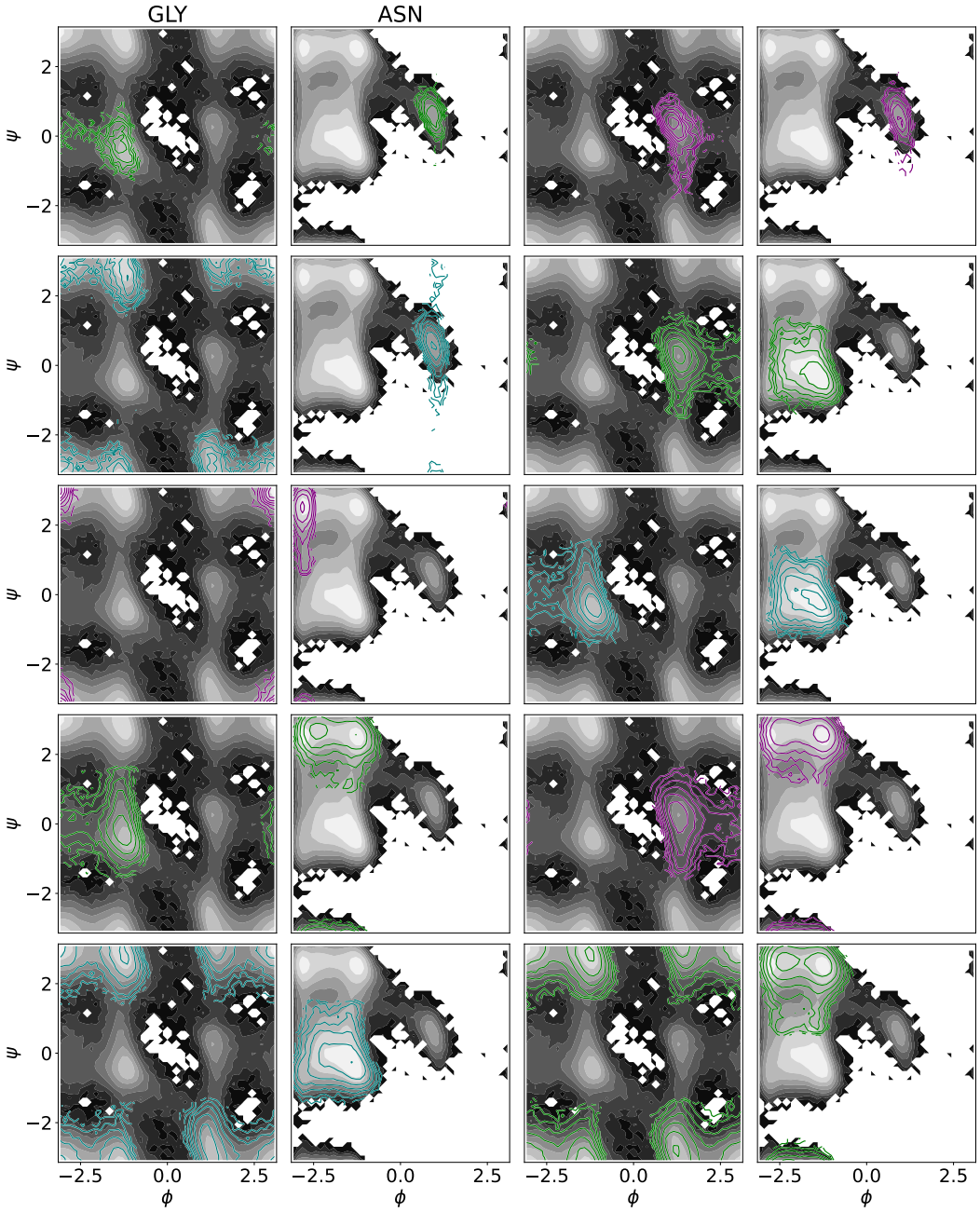


Figure 19: Markov states superimposed on sampled PMFs for GLY-ASN. This figure shows the results of running JAMUN with Markov state model negative log histograms superimposed on them. The MSMs are trained on fully converged MD data and represent metastable states. Each pair of Ramachandran plots correspond to a single metastable basin. Note that the gray-scale represents JAMUN samples while the colored is from molecular dynamics.

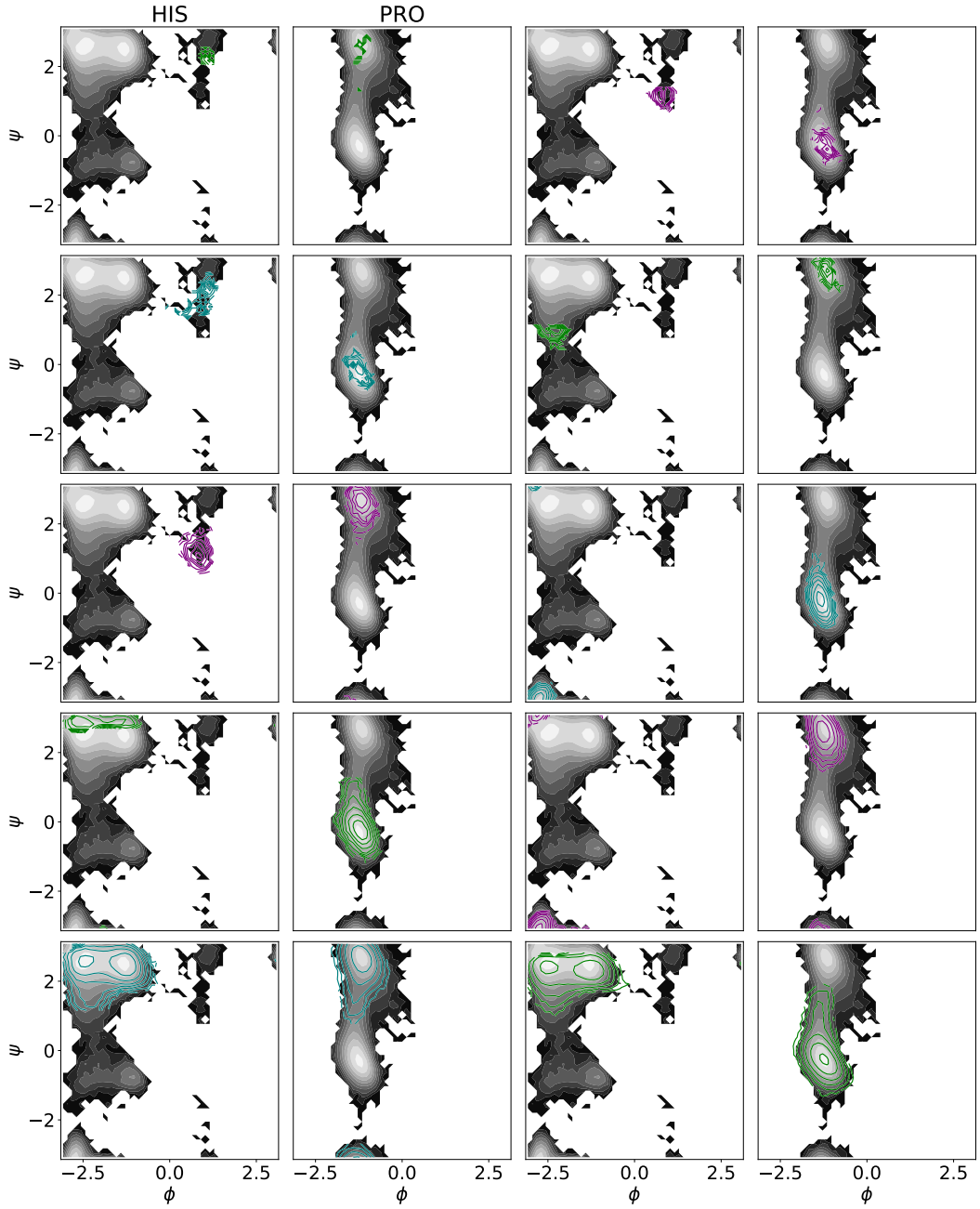


Figure 20: Markov states superimposed on sampled PMFs for HIS-PRO. This figure shows the results of running JAMUN with Markov state model negative log histograms superimposed on them. The MSMs are trained on fully converged MD data and represent metastable states. Each pair of Ramachandran plots correspond to a single metastable basin. Note that the gray-scale represents JAMUN samples while the colored is from molecular dynamics.

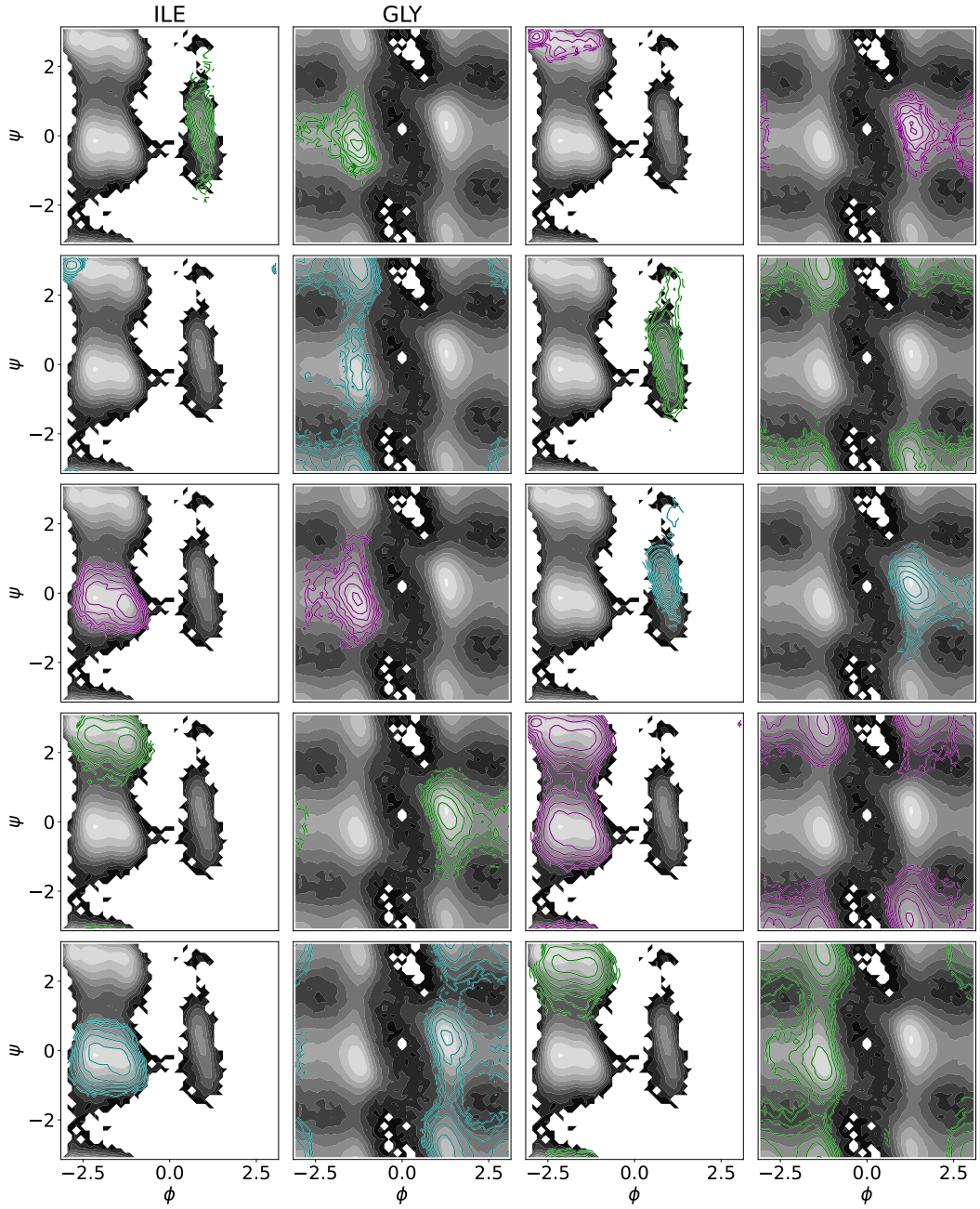


Figure 21: Markov states superimposed on sampled PMFs for ILE-GLY. This figure shows the results of running JAMUN with Markov state model negative log histograms superimposed on them. The MSMs are trained on fully converged MD data and represent metastable states. Each pair of Ramachandran plots correspond to a single metastable basin. Note that the gray-scale represents JAMUN samples while the colored is from molecular dynamics.

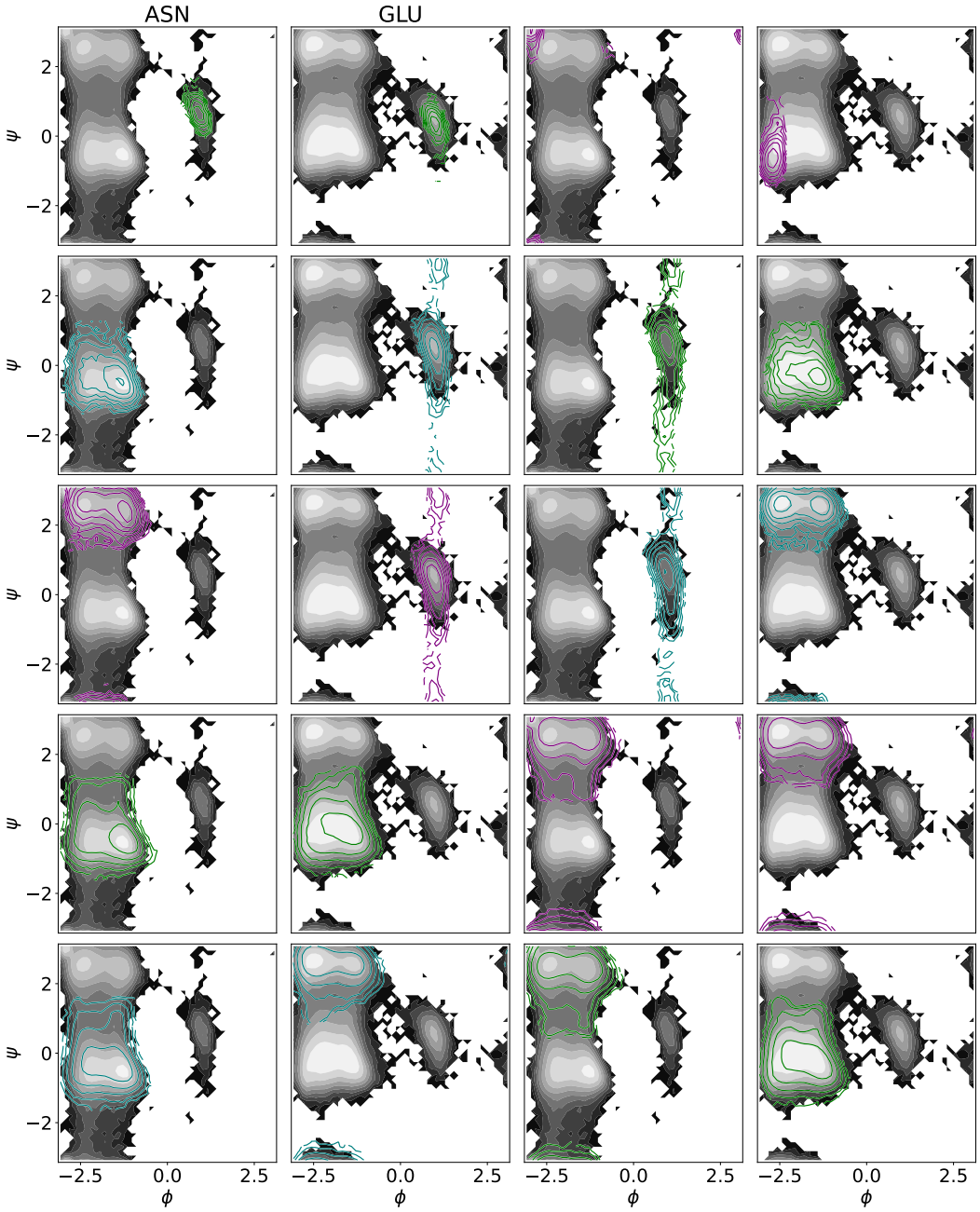


Figure 22: Markov states superimposed on sampled PMFs for ASN-GLU. This figure shows the results of running JAMUN with Markov state model negative log histograms superimposed on them. The MSMs are trained on fully converged MD data and represent metastable states. Each pair of Ramachandran plots correspond to a single metastable basin. Note that the gray-scale represents JAMUN samples while the colored is from molecular dynamics.

B Normalization

As the noise level σ is increased, $y = x + \sigma\varepsilon$ where $\varepsilon \sim \mathcal{N}(0, \mathbb{I}_{N\times 3})$ expands in space. Let $\tilde{y}$ represent the 'normalized' input y, as seen by the network F_θ:

$$\tilde{y} = c_{\text{in}}(\sigma)y \tag{9}$$

To control the expansion of y, $c_{\text{in}}(\sigma)$ is chosen such that the following property holds:

$$\mathbb{E}_{\substack{(i,j)\sim\text{Uniform}(E)\\ \varepsilon\sim\mathcal{N}(0,\mathbb{I}_{N\times 3})}}[\|\tilde{y}_i - \tilde{y}_j\|^2] = 1 \text{ at all noise levels } \sigma. \tag{10}$$

Note that this is distinct from the normalization chosen by (Karras et al., 2022, 2024), which normalizes $\|y\|$ directly. The intuition behind this normalization is that the GNN model F_θ does not operate on atom positions y directly, but instead uses the relative vectors $y_i - y_j$ to account for translation invariance, and controlling this object directly ensures that the topology of the graph does not change with varying noise level σ.

To achieve this, we compute:

$$c_{\text{in}}(\sigma) = \frac{1}{\sqrt{C + 6\sigma^2}} \tag{11}$$

where $C = \mathbb{E}_{(i,j)\sim\text{Uniform}(E)} \|x_i - x_j\|^2$ can be easily estimated from the true data distribution. The full derivation can be found in Section B.1.

As the input is now appropriately normalized, the target output of the network F_θ should also be appropriately normalized. A full derivation, found in Section B.2, leads to:

$$c_{\text{skip}}(\sigma) = \frac{C}{C + 6\sigma^2} \tag{12}$$

$$c_{\text{out}}(\sigma) = \sqrt{\frac{C \cdot 6\sigma^2}{C + 6\sigma^2}} \tag{13}$$

$$c_{\text{noise}}(\sigma) = \log_{10}\sigma \tag{14}$$

The noise normalization is a scaled version of the recommendation of $\frac{1}{4}\ln\sigma$ for images in Karras et al. (2022, 2024).

B.1 Input Normalization

Fix an $(i, j) \in E$ from Equation 10. As $\varepsilon_i, \varepsilon_j \overset{\text{iid}}{\sim} \mathcal{N}(0, \mathbb{I}_3)$, we have $\varepsilon_i - \varepsilon_j \sim \mathcal{N}(0, 2\mathbb{I}_3)$ from the closure of the multivariate Gaussian under linear combinations. Thus, for each component $d = 1, 2\text{and}3$, we have: $(\varepsilon_i - \varepsilon_j)_{(d)} \sim \mathcal{N}(0, 2)$ and hence:

$$\mathbb{E}_{\varepsilon\sim\mathcal{N}(0,\mathbb{I}_{N\times 3})}[(x_i - x_j)^T(\varepsilon_i - \varepsilon_j)] = \sum_{d=1}^{3}(x_i - x_j)_{(d)}\mathbb{E}[(\varepsilon_i - \varepsilon_j)_{(d)}] = 0 \tag{15}$$

$$\mathbb{E}_{\varepsilon\sim\mathcal{N}(0,\mathbb{I}_{N\times 3})}[\|\varepsilon_i - \varepsilon_j\|^2] = \sum_{d=1}^{3}\mathbb{E}[(\varepsilon_i - \varepsilon_j)^2_{(d)}] = 6 \tag{16}$$

We can now compute:

$$\begin{aligned}
&\mathbb{E}_z[\|\tilde{y}_i - \tilde{y}_j\|^2] \\
&= c_{\text{in}}(\sigma)^2\mathbb{E}_\varepsilon[\|y_i - y_j\|^2] \\
&= c_{\text{in}}(\sigma)^2\mathbb{E}_\varepsilon[\|x_i - x_j + \sigma(\varepsilon_i - \varepsilon_j)\|^2] \\
&= c_{\text{in}}(\sigma)^2\left(\|x_i - x_j\|^2 + 2\sigma\mathbb{E}_\varepsilon[(x_i - x_j)^T(\varepsilon_i - \varepsilon_j)] + \sigma^2\mathbb{E}_\varepsilon[\|\varepsilon_i - \varepsilon_j\|^2]\right) \\
&= c_{\text{in}}(\sigma)^2\left(\|x_i - x_j\|^2 + \sigma^2\mathbb{E}_\varepsilon[\|\varepsilon_i - \varepsilon_j\|^2]\right) \\
&= c_{\text{in}}(\sigma)^2\left(\|x_i - x_j\|^2 + 6\sigma^2\right).
\end{aligned} \tag{17}$$

Now, taking the expectation over all $(i,j) \in E$ uniformly:

$$\begin{aligned}\mathbb{E}_{\substack{(i,j)\sim\text{Uniform}(E)\\ \varepsilon\sim\mathcal{N}(0,\mathbb{I}_{N\times 3})}}[\|\tilde{y}_i - \tilde{y}_j\|^2] &= \mathbb{E}_{(i,j)\sim\text{Uniform}(E)}[\mathbb{E}_\varepsilon[\|\tilde{y}_i - \tilde{y}_j\|^2]] \\ &= c_{\text{in}}(\sigma)^2 \left(\mathbb{E}_{(i,j)\sim\text{Uniform}(E)} \|x_i - x_j\|^2 + 6\sigma^2\right) \end{aligned} \tag{18}$$

Let $C = \mathbb{E}_{(i,j)\sim\text{Uniform}(E)} \|x_i - x_j\|^2$, which we estimate from the true data distribution. Then, from Equation 18 and our intended normalization given by Equation 10:

$$c_{\text{in}}(\sigma) = \frac{1}{\sqrt{C + 6\sigma^2}} \tag{19}$$

B.2 Output Normalization

The derivation here is identical that of (Karras et al., 2022, 2024), but with our normalization. The denoising loss at a single noise level is:

$$\mathcal{L}(\hat{x}_\theta, \sigma) = \mathbb{E}_{X\sim p_X}\mathbb{E}_{\varepsilon\sim\mathcal{N}(0,\mathbb{I}_{N\times 3})}[\|\hat{x}_\theta(X + \sigma\varepsilon, \sigma) - X\|^2] \tag{20}$$

which gets weighted across a distribution p_σ of noise levels by (unnormalized) weights $\lambda(\sigma)$:

$$\begin{aligned}\mathcal{L}(\hat{x}_\theta) &= \mathbb{E}_{\sigma\sim p_\sigma}[\lambda(\sigma)\mathcal{L}(\hat{x}_\theta, \sigma)] \\ &= \mathbb{E}_{\sigma\sim p_\sigma}\mathbb{E}_{X\sim p_X}\mathbb{E}_{\varepsilon\sim\mathcal{N}(0,\mathbb{I}_{N\times 3})}[\lambda(\sigma)\|\hat{x}_\theta(X + \sigma\varepsilon, \sigma) - X\|^2] \\ &= \mathbb{E}_{\sigma\sim p_\sigma}\mathbb{E}_{X\sim p_X}\mathbb{E}_{Y\sim\mathcal{N}(X,\sigma^2\mathbb{I}_{N\times 3})}[\lambda(\sigma)\|\hat{x}_\theta(Y, \sigma) - X\|^2] \\ &= \mathbb{E}_{\sigma\sim p_\sigma}\mathbb{E}_{X\sim p_X}\mathbb{E}_{Y\sim\mathcal{N}(X,\sigma^2\mathbb{I}_{N\times 3})}[\lambda(\sigma)\|c_{\text{skip}}(\sigma)Y + c_{\text{out}}(\sigma)F_\theta(c_{\text{in}}(\sigma)Y, c_{\text{noise}}(\sigma)) - x\|^2] \\ &= \mathbb{E}_{\sigma\sim p_\sigma}\mathbb{E}_{X\sim p_X}\mathbb{E}_{Y\sim\mathcal{N}(X,\sigma^2\mathbb{I}_{N\times 3})}\left[\lambda(\sigma)c_{\text{out}}(\sigma)^2\left\|F_\theta(c_{\text{in}}(\sigma)Y, c_{\text{noise}}(\sigma)) - \frac{x - c_{\text{skip}}(\sigma)Y}{c_{\text{out}}(\sigma)}\right\|^2\right] \\ &= \mathbb{E}_{\sigma\sim p_\sigma}\mathbb{E}_{X\sim p_X}\mathbb{E}_{Y\sim\mathcal{N}(X,\sigma^2\mathbb{I}_{N\times 3})}\left[\lambda(\sigma)c_{\text{out}}(\sigma)^2\|F_\theta(c_{\text{in}}(\sigma)Y, c_{\text{noise}}(\sigma)) - F\|^2\right] \end{aligned} \tag{21}$$

where:

$$F(y, \sigma) = \frac{x - c_{\text{skip}}(\sigma)y}{c_{\text{out}}(\sigma)} \tag{22}$$

is the effective training target for the network F_θ. We want to normalize F similarly as the network input:

$$\mathbb{E}_{\substack{(i,j)\sim\text{Uniform}(E)\\ \varepsilon\sim\mathcal{N}(0,\mathbb{I}_{N\times 3})}}[\|F_i - F_j\|^2] = 1 \text{ at all noise levels } \sigma. \tag{23}$$

Again, for a fixed $(i,j) \in E$, we have:

$$\begin{aligned}\mathbb{E}_\varepsilon\|F_i - F_j\|^2 &= \frac{\mathbb{E}_\varepsilon\|(x_i - x_j) - c_{\text{skip}}(\sigma)(y_i - y_j)\|^2}{c_{\text{out}}(\sigma)^2} \\ &= \frac{\mathbb{E}_\varepsilon\|(1 - c_{\text{skip}}(\sigma))(x_i - x_j) - c_{\text{skip}}(\sigma)\sigma\cdot(\varepsilon_i - \varepsilon_j)\|^2}{c_{\text{out}}(\sigma)^2} \\ &= \frac{(1 - c_{\text{skip}}(\sigma))^2\|x_i - x_j\|^2 + c_{\text{skip}}(\sigma)^2\cdot 6\sigma^2}{c_{\text{out}}(\sigma)^2} \end{aligned} \tag{24}$$

and hence:

$$\begin{aligned}&\mathbb{E}_{\substack{(i,j)\sim\text{Uniform}(E)\\ \varepsilon\sim\mathcal{N}(0,\mathbb{I}_{N\times 3})}}[\|F_i - F_j\|^2] = 1 \\ \implies &\frac{(1 - c_{\text{skip}}(\sigma))^2\cdot C + c_{\text{skip}}(\sigma)^2\cdot 6\sigma^2}{c_{\text{out}}(\sigma)^2} = 1 \\ \implies &c_{\text{out}}(\sigma)^2 = (1 - c_{\text{skip}}(\sigma))^2\cdot C + c_{\text{skip}}(\sigma)^2\cdot 6\sigma^2 \end{aligned} \tag{25}$$

where C was defined above. Now, to minimize $c_{\text{out}}(\sigma)$ to maximize reuse and avoid amplifying network errors, as recommended by Karras et al. (2022, 2024):

$$\begin{aligned} &\frac{d}{dc_{\text{skip}}(\sigma)} c_{\text{out}}(\sigma)^2 = 0 \\ \implies &-2(1 - c_{\text{skip}}(\sigma)) \cdot C + 2c_{\text{skip}}(\sigma) \cdot 6\sigma^2 = 0 \\ \implies &c_{\text{skip}}(\sigma) = \frac{C}{C + 6\sigma^2} \end{aligned} \tag{26}$$

Substituting into Equation 25, we get after some routine simplification:

$$c_{\text{out}}(\sigma) = \sqrt{\frac{C \cdot 6\sigma^2}{C + 6\sigma^2}} \tag{27}$$

The noise normalization is chosen as $c_{\text{noise}}(\sigma) = \log_{10} \sigma$, a scaled version of the recommendation of $\frac{1}{4} \ln \sigma$ for images in Karras et al. (2022, 2024).

From Equation 21, we set $\lambda(\sigma) = \frac{1}{c_{\text{out}}(\sigma)^2}$ to normalize the loss at at all noise levels, as in Karras et al. (2022, 2024).

B.3 Rotational Alignment

As described in Algorithm 1, we use the Kabsch-Umeyama algorithm (Kabsch, 1976; Umeyama, 1991) to rotationally align y to x before calling the denoiser.

Algorithm 1 Rotational Alignment with the Kabsch-Umeyama Algorithm

Require: Noisy Sample $y \in \mathbb{R}^{N \times 3}$, True Sample $x \in \mathbb{R}^{N \times 3}$.
$H \leftarrow x^T y$ ▷ $H \in \mathbb{R}^{3 \times 3}$
$U, S, V^T \leftarrow \text{SVD}(H)$ ▷ $U, V \in \mathbb{R}^{3 \times 3}$
$\mathbf{R}^* \leftarrow U\text{diag}[1, 1, \det(U)\det(V)]V^T$
return $y(\mathbf{R}^*)^T$

Note that both y and x are mean-centered to respect translational equivariance:

$$\sum_{i=1}^{N} y_i = \vec{0} \in \mathbb{R}^3 \tag{28}$$

$$\sum_{i=1}^{N} x_i = \vec{0} \in \mathbb{R}^3 \tag{29}$$

so there is no net translation.

C The Denoiser Minimizes the Expected Loss

Here, we prove Equation 5, rewritten here for clarity:

$$\hat{x}(\cdot) \equiv \mathbb{E}[X \mid Y = \cdot] = \underset{f:\mathbb{R}^{N\times 3} \to \mathbb{R}^{N \times 3}}{\arg\min} \mathbb{E}_{\substack{X \sim p_X, \varepsilon \sim \mathcal{N}(0, \mathbb{I}_{N\times 3}) \\ Y = X + \sigma\varepsilon}} [\|f(Y) - X\|^2] \tag{30}$$

First, we can decompose the loss over the domain $\mathbb{R}^{N \times 3}$ of Y:

$$\mathbb{E}_{\substack{X \sim p_X, \varepsilon \sim \mathcal{N}(0, \mathbb{I}_{N\times 3}) \\ Y = X + \sigma\varepsilon}} [\|f(Y) - X\|^2] = \mathbb{E}_{X \sim p_X, Y \sim p_Y} [\|f(Y) - X\|^2] \tag{31}$$

$$= \int_{\mathbb{R}^{N\times 3}} \int_{\mathbb{R}^{N\times 3}} \|f(y) - x\|^2 \, p_{X,Y}(x, y) dx dy \tag{32}$$

$$= \int_{\mathbb{R}^{N\times 3}} \underbrace{\int_{\mathbb{R}^{N\times 3}} \|f(y) - x\|^2 \, p_{Y|X}(y \mid x) p_X(x) dx dy}_{l(f,y)} \tag{33}$$

$$= \int_{\mathbb{R}^{N\times 3}} l(f, y) dy \tag{34}$$

where $l(f, y) \geq 0$ for all functions f and inputs y. Hence, any minimizer f^* must minimize the local denoising loss $l(f^*, y)$ at each point $y \in \mathbb{R}^{N\times 3}$. For a fixed $y \in \mathbb{R}^{N\times 3}$, the loss $l(f, y)$ is convex as a function of $f(y)$. Hence, the global minimizer can be found by finding the critical points of $l(f, y)$ as a function of $f(y)$:

$$\nabla_{f(y)} l(f, y) = 0 \tag{35}$$

$$\implies \nabla_{f(y)} \int_{\mathbb{R}^{N\times 3}} \|f(y) - x\|^2 \, p_{Y|X}(y \mid x) p_X(x) dx = 0 \tag{36}$$

$$\implies \int_{\mathbb{R}^{N\times 3}} 2(f^*(y) - x) p_{Y|X}(y \mid x) p_X(x) dx = 0 \tag{37}$$

Rearranging:

$$f^*(y) = \frac{\int_{\mathbb{R}^{N\times 3}} x \, p_{Y|X}(y \mid x) p_X(x) dx}{\int_{\mathbb{R}^{N\times 3}} p_{Y|X}(y \mid x) p_X(x) dx} \tag{38}$$

$$= \frac{\int_{\mathbb{R}^{N\times 3}} x \, p_{Y|X}(y \mid x) p_X(x) dx}{p_Y(y)} \tag{39}$$

$$= \int_{\mathbb{R}^{N\times 3}} x \, \frac{p_{Y|X}(y \mid x) p_X(x)}{p_Y(y)} dx \tag{40}$$

$$= \int_{\mathbb{R}^{N\times 3}} x \, p_{X|Y}(x \mid y) dx \tag{41}$$

$$= \mathbb{E}[X \mid Y = y] \tag{42}$$

$$= \hat{x}(y) \tag{43}$$

by Bayes' rule. Hence, the denoiser as defined by Equation 4 is indeed the minimzer of the denoising loss:

$$\hat{x}(\cdot) \equiv \mathbb{E}[X \mid Y = \cdot] = \underset{f:\mathbb{R}^{N\times 3}\to\mathbb{R}^{N\times 3}}{\arg\min} \; \mathbb{E}_{\substack{X\sim p_X, \varepsilon\sim\mathcal{N}(0,\mathbb{I}_{N\times 3}) \\ Y = X+\sigma\varepsilon}} [\|f(Y) - X\|^2] \tag{44}$$

as claimed.

D Relating the Score and the Denoiser

Here, we rederive Equation 6, relating the score function $\nabla \log p_Y$ and the denoiser $\hat{x}$, as first shown by Robbins (1956); Miyasawa (1960).

Let $X \sim p_X$ defined over $\mathbb{R}^{N\times 3}$ and $\eta \sim \mathcal{N}(0, \mathbb{I}_{N\times 3})$. Let $Y = X + \sigma\eta$, which means:

$$p_{Y|X}(y \mid x) = \mathcal{N}(y; x, \mathbb{I}_{N\times 3}) = \frac{1}{(2\pi\sigma^2)^{\frac{3N}{2}}} \exp\left(-\frac{\|y - x\|^2}{2\sigma^2}\right) \tag{45}$$

Then:

$$\mathbb{E}[X \mid Y = y] = y + \sigma^2 \nabla_y \log p_Y(y) \tag{46}$$

To prove this:

$$\nabla_y p_{Y|X}(y \mid x) = -\frac{y - x}{\sigma^2} p_{Y|X}(y \mid x) \tag{47}$$

$$\implies (x - y) p_{Y|X}(y \mid x) = \sigma^2 \nabla_y p_{Y|X}(y \mid x) \tag{48}$$

$$\implies \int_{\mathbb{R}^{N\times 3}} (x - y) p_{Y|X}(y \mid x) \, p_X(x) dx = \int_{\mathbb{R}^{N\times 3}} \sigma^2 \nabla_y p_{Y|X}(y \mid x) \, p_X(x) dx \tag{49}$$

By Bayes' rule:

$$p_{Y|X}(y \mid x) p_X(x) = p_{X,Y}(x, y) = p_{X|Y}(x|y) p_Y(y) \tag{50}$$

and, by definition of the marginals:

$$\int_{\mathbb{R}^{N\times 3}} p_{X,Y}(x, y) dx = p_Y(y) \tag{51}$$

For the left-hand side, we have:

$$\int_{\mathbb{R}^{N\times 3}} (x-y)p_{Y|X}(y \mid x)p_X(x)dx = \int_{\mathbb{R}^{N\times 3}} (x-y)p_{X,Y}(x,y)dx \tag{52}$$

$$= \int_{\mathbb{R}^{N\times 3}} xp_{X,Y}(x,y)dx - \int_{\mathbb{R}^{N\times 3}} yp_{X,Y}(x,y)dx \tag{53}$$

$$= p_Y(y)\left(\int_{\mathbb{R}^{N\times 3}} xp_{X|Y}(x \mid y)dx - y\int_{\mathbb{R}^{N\times 3}} p_{X|Y}(x \mid y)dx\right) \tag{54}$$

$$= p_Y(y)\left(\mathbb{E}[X \mid Y=y] - y\right) \tag{55}$$

For the right-hand side, we have:

$$\sigma^2 \int_{\mathbb{R}^{N\times 3}} \nabla_y p_{Y|X}(y \mid x)p_X(x)dx = \sigma^2\nabla_y \int_{\mathbb{R}^{N\times 3}} p_{Y|X}(y \mid x)p_X(x)dx \tag{56}$$

$$= \sigma^2\nabla_y \int_{\mathbb{R}^{N\times 3}} p_{X,Y}(x,y)dx \tag{57}$$

$$= \sigma^2\nabla_y p_Y(y) \tag{58}$$

Thus,

$$p_Y(y)\left(\mathbb{E}[X \mid Y=y] - y\right) = \sigma^2\nabla_y p_Y(y) \tag{59}$$

$$\implies \mathbb{E}[X \mid Y=y] = y + \sigma^2\frac{\nabla_y p_Y(y)}{p_Y(y)} \tag{60}$$

$$= y + \sigma^2\nabla_y \log p_Y(y) \tag{61}$$

as claimed.

E Numerical Solvers for Langevin Dynamics

As mentioned in Section 3.2, solving the Stochastic Differential Equation corresponding to Langevin dynamics is often performed numerically. In particular, BAOAB (Leimkuhler & Matthews, 2012, 2015; Sachs et al., 2017) refers to a 'splitting method' that solves the Langevin dynamics SDE by splitting it into three different components labelled by $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{O}$ below:

$$dy = \underbrace{v_y dt}_{\mathcal{A}} \tag{62}$$

$$dv_y = \underbrace{M^{-1}\nabla_y \log p_Y(y)dt}_{\mathcal{B}} \underbrace{- \gamma v_y dt + \sqrt{2\gamma}M^{-\frac{1}{2}}dB_t}_{\mathcal{O}} \tag{63}$$

where both $y, v_y \in \mathbb{R}^d$. This leads to the following update operators:

$$\mathcal{A}_{\Delta t}\begin{bmatrix} y \\ v_y \end{bmatrix} = \begin{bmatrix} y + v_y\Delta t \\ v_y \end{bmatrix} \tag{64}$$

$$\mathcal{B}_{\Delta t}\begin{bmatrix} y \\ v_y \end{bmatrix} = \begin{bmatrix} y \\ v_y + M^{-1}\nabla_y \log p_Y(y)\Delta t \end{bmatrix} \tag{65}$$

$$\mathcal{O}_{\Delta t}\begin{bmatrix} y \\ v_y \end{bmatrix} = \begin{bmatrix} y \\ e^{-\gamma\Delta t}v_y + M^{-\frac{1}{2}}\sqrt{1-e^{-2\gamma\Delta t}}B \end{bmatrix} \tag{66}$$

where $B \sim \mathcal{N}(0, \mathbb{I}_d)$ is resampled every iteration. As highlighted by Kieninger & Keller (2022), the $\mathcal{A}$ and $\mathcal{B}$ updates are obtained by simply discretizing the updates highlighted in Equation 62 by the Euler method. The $\mathcal{O}$ update refers to a explicit solution of the Ornstein-Uhlenbeck process, which we rederive for completeness in Appendix F.

Finally, the iterates of the BAOAB algorithm are given by a composition of these update steps, matching the name of the method:

$$\begin{bmatrix} y^{(t+1)} \\ v_y^{(t+1)} \end{bmatrix} = \mathcal{B}_{\frac{\Delta t}{2}}\mathcal{A}_{\frac{\Delta t}{2}}\mathcal{O}_{\Delta t}\mathcal{A}_{\frac{\Delta t}{2}}\mathcal{B}_{\frac{\Delta t}{2}}\begin{bmatrix} y^{(t)} \\ v_y^{(t)} \end{bmatrix} \tag{67}$$

F The Ornstein-Uhlenbeck Process

For completeness, we discuss the distributional solution of the Ornstein-Uhlenbeck process, taken directly from the excellent Leimkuhler & Matthews (2015). In one dimension, the Ornstein-Uhlenbeck Process corresponds to the following Stochastic Differential Equation (SDE):

$$dv_y = -\gamma v_y dt + \sqrt{2\gamma} M^{-\frac{1}{2}} dB_t \tag{68}$$

Multiplying both sides by the integrating factor $e^{\gamma t}$:

$$e^{\gamma t} dv_y = -\gamma e^{\gamma t}(v_y dt + e^{\gamma t}\sqrt{2\gamma} M^{-\frac{1}{2}} dB_t \tag{69}$$

$$\implies e^{\gamma t}(dv_y + \gamma v_y dt) = e^{\gamma t}\sqrt{2\gamma} M^{-\frac{1}{2}} dB_t \tag{70}$$

and identifying:

$$e^{\gamma t}(dv_y + \gamma v_y dt) = d(e^{\gamma t} v_y) \tag{71}$$

We get after integrating from t_1 to t_2, two adjacent time steps of our integration grid:

$$d(e^{\gamma t} v_y) = e^{\gamma t}\sqrt{2\gamma} M^{-\frac{1}{2}} dB_t \tag{72}$$

$$\implies \int_{t_1}^{t_2} d(e^{\gamma t} v_y) = \int_{t_1}^{t_2} e^{\gamma t}\sqrt{2\gamma} M^{-\frac{1}{2}} dB_t \tag{73}$$

$$\implies e^{\gamma t_2} v_y(t_2) - e^{\gamma t_1} v_y(t_1) = \sqrt{2\gamma} M^{-\frac{1}{2}} \int_{t_1}^{t_2} e^{\gamma t} dB_t \tag{74}$$

Now, for a Wiener process B_t, if $g(t)$ is a deterministic function, $\int_{t_1}^{t_2} g(t) dB_t$ is distributed as $\mathcal{N}\left(0, \int_{t_1}^{t_2} g(t)^2 dt\right)$ by Itô's integral. Thus, applying this result to $g(t) = e^{\gamma t}$, we get:

$$e^{\gamma t_2} v_y(t_2) - e^{\gamma t_1} v_y(t_1) = \sqrt{2\gamma} M^{-\frac{1}{2}} \mathcal{N}\left(0, \frac{e^{2\gamma t_2} - e^{2\gamma t_1}}{2\gamma}\right) \tag{75}$$

$$\implies v_y(t_2) = e^{-\gamma(t_2 - t_1)} v_y(t_1) + \sqrt{2\gamma} M^{-\frac{1}{2}} e^{-\gamma t_2} \mathcal{N}\left(0, \frac{e^{2\gamma t_2} - e^{2\gamma t_1}}{2\gamma}\right) \tag{76}$$

$$= e^{-\gamma(t_2 - t_1)} v_y(t_1) + \sqrt{2\gamma} M^{-\frac{1}{2}} \sqrt{\frac{1 - e^{2\gamma(t_1 - t_2)}}{2\gamma}} \mathcal{N}(0, 1) \tag{77}$$

$$= e^{-\gamma(t_2 - t_1)} v_y(t_1) + M^{-\frac{1}{2}} \sqrt{1 - e^{2\gamma(t_1 - t_2)}} \mathcal{N}(0, 1) \tag{78}$$

In the $N \times 3$ dimensional case, as the Wiener processes are all independent of each other, we directly get:

$$v_y(t_2) = e^{-\gamma(t_2 - t_1)} v_y(t_1) + M^{-\frac{1}{2}} \sqrt{1 - e^{2\gamma(t_1 - t_2)}} \mathcal{N}(0, \mathbb{I}_{N \times 3}) \tag{79}$$

Setting $\Delta t = t_2 - t_1$, we get the form of the $\mathcal{O}$ operator (Equation 64) of the BAOAB integrator in Appendix E.

G Parallelizing Sampling with Multiple Independent Chains

To increase sampling throughput, we initialize multiple chains: $y_1^{(0)}, \ldots, y_{N_{\text{ch}}}^{(0)}$, where:

$$y_{\text{ch}}^{(0)} = x^{(0)} + \sigma \varepsilon_{\text{ch}}^{(0)} \tag{80}$$

where $\varepsilon_{\text{ch}}^{(0)} \overset{\text{iid}}{\sim} \mathcal{N}(0, \mathbb{I}_{N \times 3})$ for $\text{ch} = 1, \ldots, N_{\text{ch}}$ are all independent of each other. We evolve these chains with independent walk steps (Equation 2) and denoise each chain with independent jump steps (Equation 6). This independence allows batching over the $y_{\text{ch}}^{(t)}$ over all chains ch at each iteration t. This improves throughput significantly.

Note that at $t = 0$, the chains are correlated as they are all initialized from the same $x^{(0)}$. However, if the number of samples per chain is large enough, the chains are no longer correlated, as they have now mixed into the stationary distribution. Here, we use $N_{\text{ch}} = 32$ different chains.

H Hyperparameters

H.1 Datasets

For the capped diamines, we use 320,000 snapshots for 200 training diamines, solvated in explicit water and simulated using OpenMM. For the uncapped diamines, we use the 2AA-Large dataset from Timewarp.

H.2 Model

We use a fully-connected graph for message-passing. The hidden features $h^{(n)}$ for $n = 0, \ldots, 4$ contain 120 scalar and 32 vector features per atom. We use spherical harmonics up to $l = 1$ for the tensor product.

H.3 Training

The learning rate for Adam is $.002$. Models are trained with a batch size of 42 over each of 6 NVIDIA RTX A100 GPUs.

H.4 Sampling

For sampling capped diamines, we use $\sigma = 0.4$ Å. For sampling uncapped diamines, we use $\sigma = 0.6$ Å. We set $M = 1$. We use friction of $\gamma = 0.1$ and a Langevin dynamics step size of $\Delta t = \sigma$.