# JAMUN: TRANSFERABLE MOLECULAR CONFORMATIONAL ENSEMBLE GENERATION WITH WALK-JUMP SAMPLING

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Conformational ensembles of protein structures are immensely important to understanding protein function. Current techniques for sampling ensembles are computationally inefficient, or do not transfer to systems outside their training data. We present walk-Jump Accelerated Molecular ensembles with Universal Noise (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 generators. Further, JAMUN is able to predict the stable basins of small peptides that were not seen during training.

## 1 INTRODUCTION

Molecules are not static. They move, and these movements can be vitally important. 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.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-Jump Accelerated Molecular ensembles with Universal Noise), 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 "...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."Borhani & Shaw (2012) Importantly, there is nothing fundamental about this small time-step limitation; it is 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 expert user input, 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); Zheng et al. (2024); Jing et al. (2024); Kim et al. (2024) These models have been applied to a variety of MD datasets from small molecules to peptides to full proteins. However, none of these models have proven

[*]These authors contributed equally to this work.

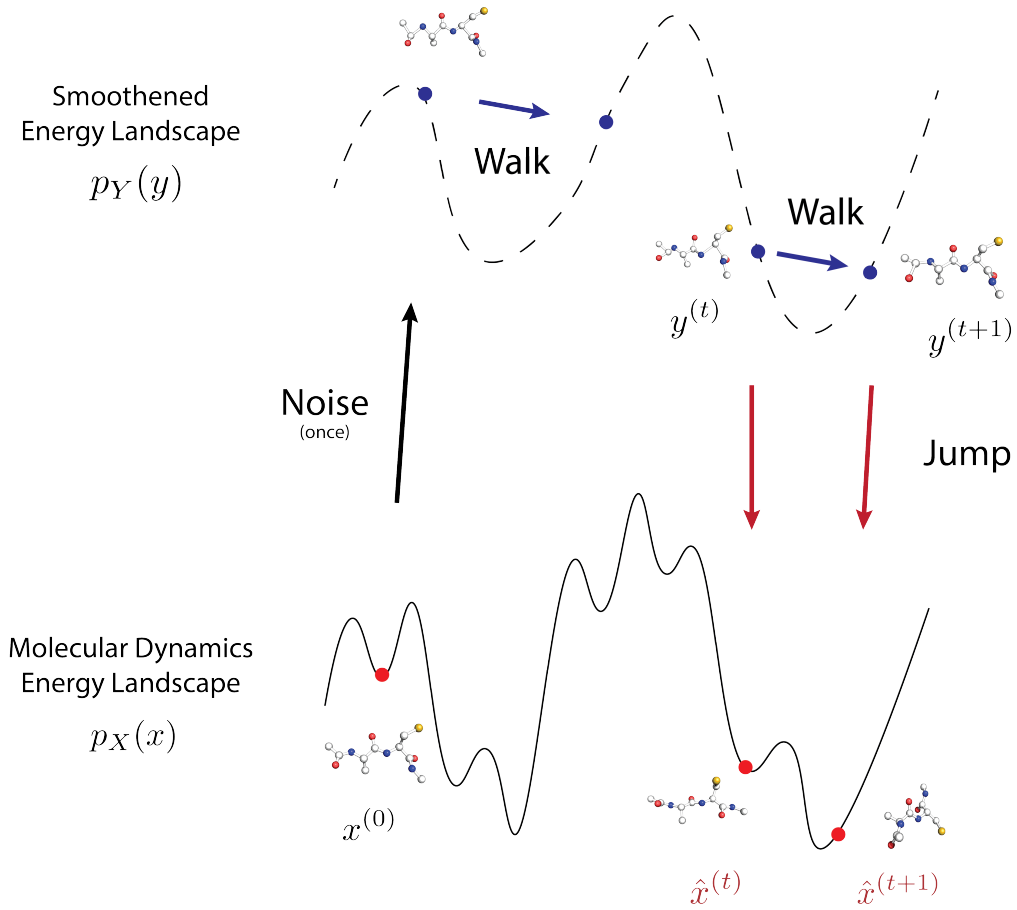


Figure 1: Overview of Walk-Jump Sampling.

to be transferable. They do not work well beyond their training data. (One notable exception is the recent Transferable Boltzmann Generators (Klein & Noé, 2024) model, with which we provide extensive comparison.) While the developments in this field have been immense, transferability remains a grand challenge. Without it 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). This framework is a natural fit for MD data. 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 partial noise, WJS simply smooths out the distribution enough to resolve sampling difficulties without fully destroying the information present in the data distribution. Moreover, the use of Langevin dynamics, the same algorithm commonly used for MD simulations, 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., 2024). 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., 2024b;a) and protein sequence generation (Frey et al., 2023). Our work is the first to our knowledge to apply WJS to point clouds. The equivariant denoising network we use is built with the e3nn library (Geiger & Smidt, 2022). For a survey of equivariant models for 3D atomic systems, see Duval et al. (2023).

## 3 Methods

### 3.1 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. Given an initial sample $x^{(0)}$ from the clean data distribution $p_X$ where $x^{(0)} \in \mathbb{R}^{N\times 3}$ represents the 3D coordinates of each of the $N$ atoms, Walk-Jump performs the following steps:

1. Construct initial sample $y^{(0)}$ from the noisy data distribution $p_Y$ by adding noise with magnitude $\sigma$ drawn from the normal distribution $\mathcal{N}$:
$$y^{(0)} = x^{(0)} + \sigma\eta^{(0)} \text{ where } \eta^{(0)} \sim \mathcal{N}(0, \mathbb{I}_{N\times 3}) \tag{1}$$
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):
$$dy = v_y dt \tag{2}$$
$$dv_y = \nabla_y \log p_Y(y)dt - \gamma v_y dt + M^{-\frac{1}{2}}\sqrt{2}dB_t \tag{3}$$
where $v_y$ represents the particle velocity, $\nabla_y \log p_Y(y)$ is the gradient of the log of the probability density function (the learned score function), $\gamma$ 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 G) to integrate Equation 2 numerically.

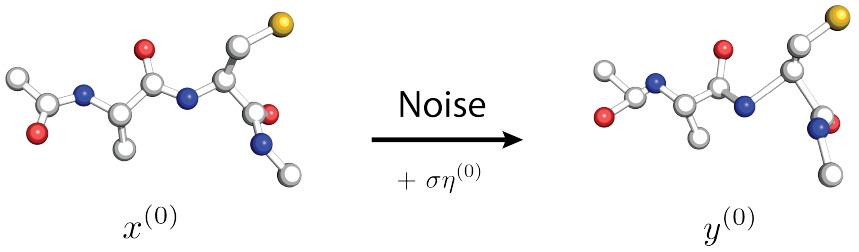


Figure 2: Adding noise to an initial conformation $x^{(0)}$ to obtain $y^{(0)} \sim p_Y$.

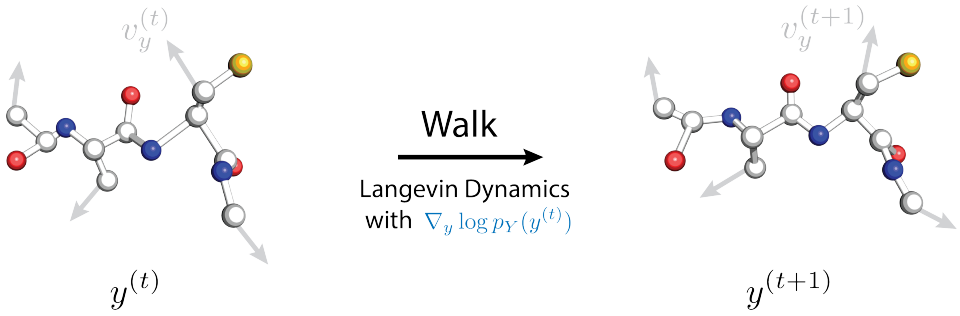


Figure 3: One iteration of BAOAB-discretized Langevin dynamics (Equation 2 and Equation 67) starting from $y^{(t)} \sim p_Y$ leads to a new sample $y^{(t+1)} \sim p_Y$.

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

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

$\hat{x}(\cdot) \equiv \mathbb{E}[X \mid Y = \cdot]$ is called the denoiser. It corresponds to the minimizer (Appendix E) of the $\ell_2$-loss between clean samples $X$ and samples denoised back from $Y = X + \sigma\eta$.

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

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

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

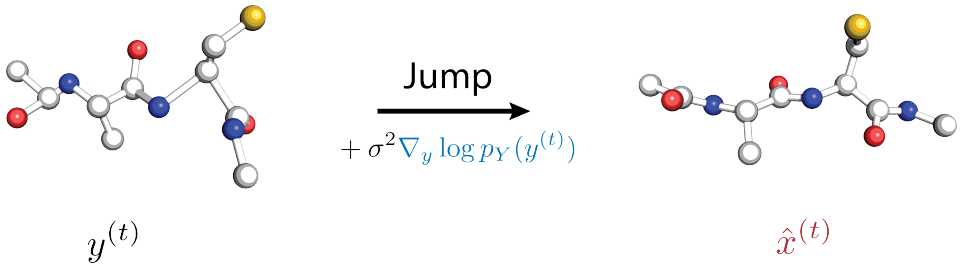


Figure 4: Denoising of $y^{(t)}$ according to Equation 6 gives us new samples $\hat{x}^{(t)}$.

Importantly, the score function $\nabla_y \log p_Y$ shows up in both the **walk** and **jump** steps.

### 3.2 Learning to Denoise

In order to run Walk-Jump Sampling as outlined above, we must train a parameterized denoising model which takes in noisy samples $y$ and outputs clean samples $\hat{x}$. 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), originally developed for diffusion models in images to model the denoiser as a neural network $\hat{x}_\theta(y, \sigma) \approx \hat{x}(y)$ parameterized by model parameters $\theta$. We appropriately modify their construction for the point cloud context as detailed in Section 3.3. Note that while the normalization is applicable for any noise level, we only need to learn a model at a **single, fixed noise level** $\sigma$. This is unlike training diffusion and flow-matching models where a large range of noise levels are required for sampling.

As described in Algorithm 1, we use an $SE(3)$-equivariant model to parametrize the denoiser Thomas et al. (2018). This is 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), $E(3)$ models are equivariant under parity, which means that are forced to transform mirrored structures identically. When we experimented with such 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.

Training the denoiser $\hat{x}_\theta$ consists of solving the following optimization problem:

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

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

**Algorithm 1** Operations of our $SE(3)$-Equivariant Denoiser $F_\theta$

**Require:** Sample $y$, Noise Level $\sigma$, Message Passing Iterations $T$, Cutoff $d_{\text{max}}$, Spherical Harmonic Degree $\ell$, Tensor Product $\otimes$

Compute neighbor lists for each atom in $y$:

$$(u, v) \in E \iff \|y_u - y_v\| \leq d_{\text{max}}$$

**for** $v \in V$ **do**:
  $h_v^{(0)} \leftarrow \textsc{InitialAtomEmbedding}(v)$
**for** $t = 1, 2, \ldots, T$ **do**:
  **for** $v \in V$ **do**:
    $h_v^{(t)} \leftarrow \frac{1}{|\mathcal{N}(v)|} \sum_{u \in \mathcal{N}(v)} \text{MLP}(\|\mathbf{y}_u - \mathbf{y}_v\|) \times \textsc{Linear}(h_u^{(t-1)} \otimes Y_\ell(\mathbf{r}_u - \mathbf{r}_v))$
    $h_v^{(t)} \leftarrow \textsc{Gate}(h_v^{(t)})$
    $h_v^{(t)} \leftarrow \textsc{Concatenate}([h_v^{(t-1)}, h_v^{(t)}])$
    $h_v^{(t)} \leftarrow \textsc{Linear}(h_v^{(t)})$
    $h_v^{(t)} \leftarrow \textsc{NoiseConditionalScaling}(h_v^{(t)}, \sigma)$
**for** $v \in V$ **do**:
  $f_v \leftarrow \textsc{Linear}(h_v^{(T)})$
**return** $\{f_v\}_{v \in V}$

Similar to Klein & Noé (2024), the initial embedding of each atom (represented by INITIALATOMEMBEDDING in Algorithm 1) is given by embedding its atomic number, the atom type as provided by the PDB, the type of its associated residue, and the sequence index of the associated residue.

### 3.3 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 $\sigma$ here, we describe the general construction for an arbitrary noise level $\sigma$,

which could be useful for training $E(3)$ 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:

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

where $F_\theta$ is the $SE(3)$-equivariant graph neural network (GNN) model, parameterized by $\theta$. $c_{\text{skip}}(\sigma), c_{\text{out}}(\sigma), c_{\text{in}}(\sigma), c_{\text{noise}}(\sigma)$ are fixed functions from $\mathbb{R}^+$ to $\mathbb{R}$, chosen to normalize the effective inputs and outputs to $F_\theta$. Further, these coefficients encourage re-use of the input $y$ at low noise levels, but the opposite at high noise levels. Next, we describe the exact form of these functions.

### 3.4 Normalization

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

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

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

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

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 $F_\theta$ 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 $\sigma$.

To achieve this, we compute:

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

Where $D$ is the number of particles and $C = \mathbb{E}_{(i,j)\sim\text{Uniform}(E)} \|x_i - x_j\|^2$ can be easily estimated from the true data distribution. The full derivation can be found in Appendix C.

As the input is now appropriately normalized, the target output of the network $F_\theta$ should also be appropriately normalized. A full derivation, found in Appendix D, leads to:

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

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

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

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

### 3.5 Rotational Alignment

As described in Algorithm 2, we use the Kabsch-Umeyama algorithm (Kabsch, 1976; Umeyama, 1991) to rotationally align $y$ to $x$ before calling the denoiser.

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

$$\sum_{i=1}^{N} y_i = \vec{0} \in \mathbb{R}^3 \tag{15}$$

$$\sum_{i=1}^{N} x_i = \vec{0} \in \mathbb{R}^3 \tag{16}$$

so there is no net translation.

**Algorithm 2** 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** $y(\mathbf{R}^*)^T$

## 4 DATASET

We demonstrate our method on two datasets: (i) the uncapped amino acids used in Timewarp (Klein et al., 2024a), specifically the 2AA-huge dataset consisting of 380 diamines split into 200 train, 80 validation and 100 test diamines, and (ii) capped amino acids simulated in water following the same splits used in (i). The molecules in (i) have a single peptide bond between amino acids of varying identities. This results in a distribution well represented by two dihedral angles, the $\phi$ angle of the second residue and the $\psi$ 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, for consistency in benchmarking, we run all experiments on these to compare against Transferable Boltzman Generators (Klein & Noé, 2024).

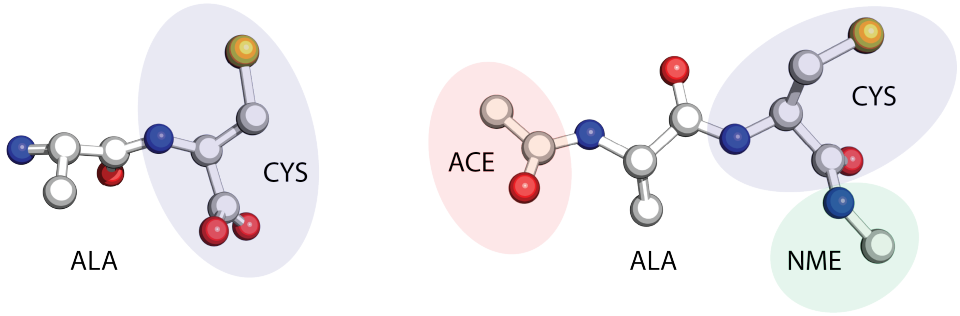


Figure 5: A side-by-side comparison of uncapped (left) compared to capped (right) ALA-CYS. The acetyl (ACE) and N-methyl (NME) capping groups provide steric hindrance and prevent local charge interactions on the N-terminal and C-terminal ends respectively.

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. These are essentially peptide bonds with the first and last residue, bonded to methyl groups. 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 $\phi$ and $\psi$ angles of both residues. We find this is a significantly richer set of distributions. 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.

## 5 RESULTS

We find that JAMUN samples conformational ensembles of short peptides faster than conventional MD while remaining reasonably faithful to thermodynamics. It does this despite being quite small by ML model standards (8.2M parameters). We note it requires no a priori knowledge or hand-crafted featurization for transferable exploration–for instance, it generates diversity in $\phi$ and $\psi$ dihedrals without explicit inputs regarding these variables. We also show a remarkable speedup relative to transferable Boltzmann generators. However, unlike the transferable Boltzmann generator, we only have access to the score of the distribution, and thus cannot precisely reweight it, making this a Boltzmann emulator.

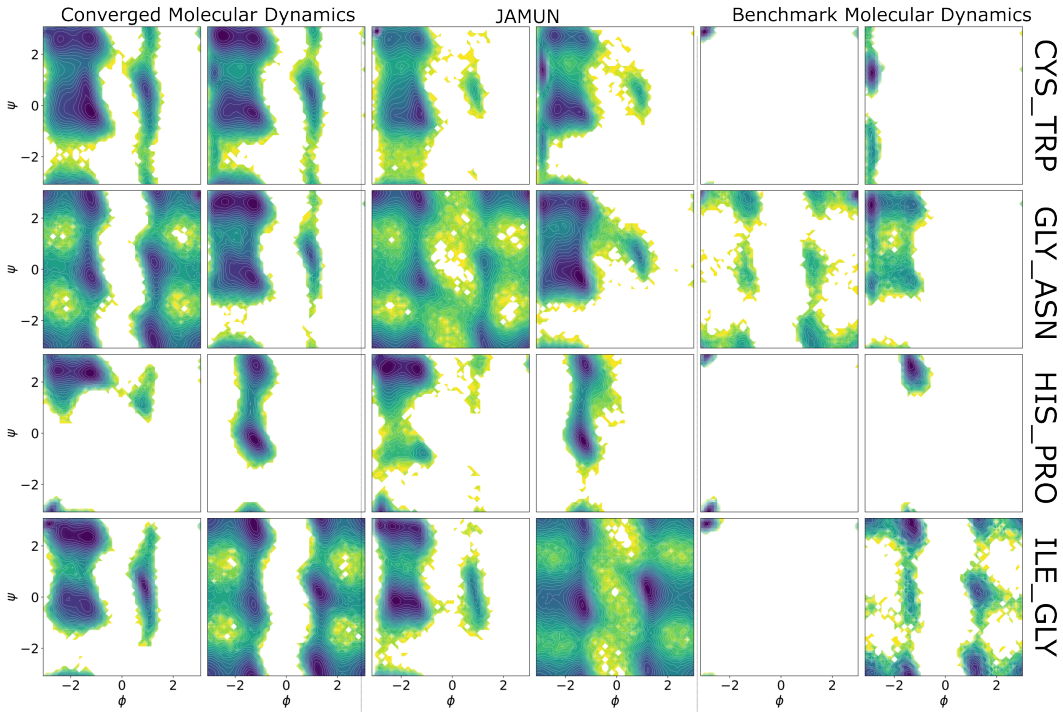


Figure 6: -log probabilities of outputs from four example diamines from the test set. These, along with Figure 9, are chosen to represent the diversity of states as well as subtle perturbations from single-point amino acid changes in the dataset. Each pair of Ramachandran plots show the distribution for a single diamine, with each row corresponding to diamine identity. The first column shows the fully converged molecular dynamics distribution. The second shows sixty four thousand samples from JAMUN. The third column shows a distribution obtained from an MD trajectory run for the exact same amount of time as the JAMUN sampler.

In Figure 6 and Figure 9 we show, for eight test peptides in the challenging capped dataset, distributions generated using both molecular dynamics and JAMUN. 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 amount of GPU time as JAMUN requires to sample 640000 points. This number was chosen to reflect reasonable sampling for validation diamines. This can be variable, for the specific 8 diamines we choose here, the sampling time with JAMUN is: 1) ASP-TRP: 38 minutes, 2) GLU-THR: 27 minutes, 3) PHE-ALA: 28 minutes, 4) ASN-GLU: 29 minutes, 5) CYS-TRP: 34 minutes, 6) GLY-ASN: 22 minutes, 7) HIS-PRO: 30 minutes, and 8) ILE-GLY: 22 minutes. In terms of MD simulation time, this is between $100ps$ and $300ps$.

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 6 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.

While the capped dataset represents higher diversity and fidelity to true peptide thermodynamics, we also wish to compare against the transferable Boltzmann generator, as it has already shown success for short peptides. To this end, we also train our model on the timewarp dataset, although please note that our hyperparameters are optimized for the capped diamines. Nevertheless, we show in Figure 7 a significant improvement in sampling. For the two machine learning methods, we simulate using an equal amount of compute-time, including post-processing steps such as chirality checks. It is clear, by visual inspection, that JAMUN samples all states, where transferable Boltzmann Generators misses a basin in the given time.

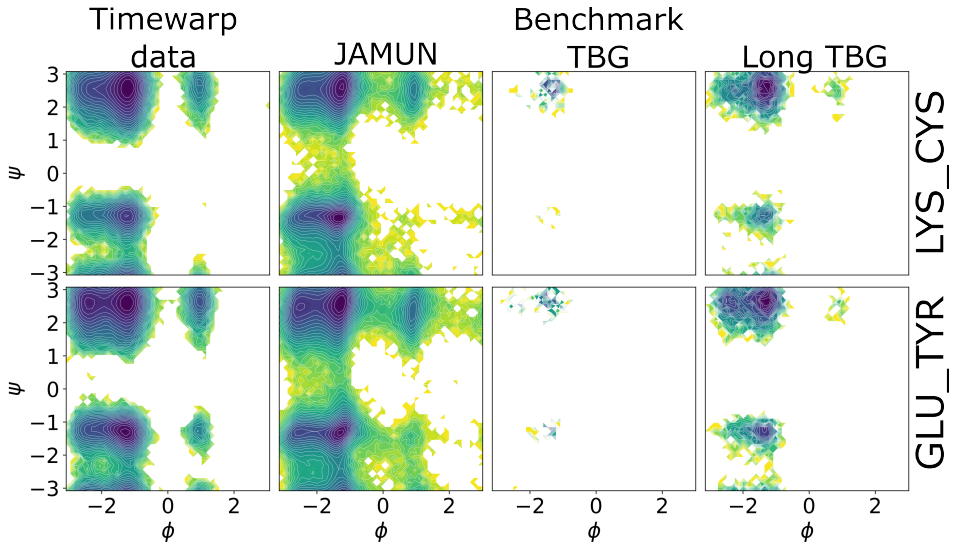


Figure 7: -log probabilities of outputs from two example diamines from the uncapped test set. These are chosen to show two examples of different distributions from this dataset which has limited conformational diversity. Each Ramachandran plot shows the distribution for a single diamine, with each row corresponding to diamine identity. The first column shows the full distribution from the timewarp dataset. The second shows sixty four thousand samples from JAMUN. The third column shows a distribution obtained from a transferable Boltzman generator run for the exact same amount of time as the JAMUN sampler. The fourth shows results from the transferable Boltzman generator run for five thousand samples (run for roughly 10 times as long as JAMUN).

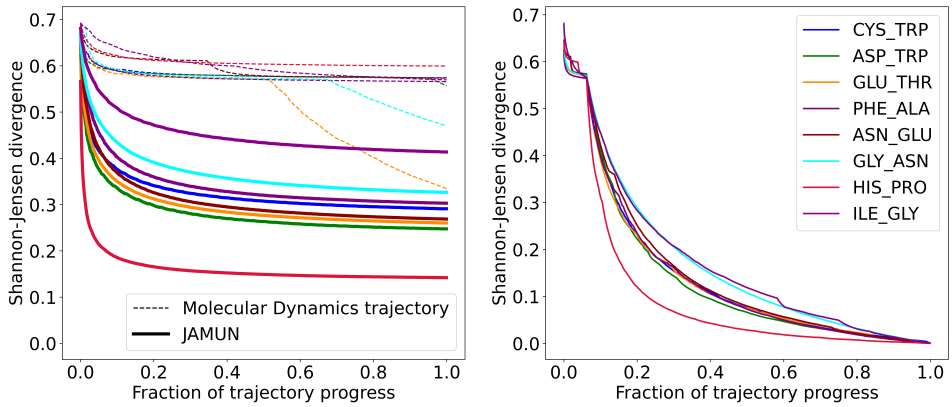


Figure 8: Shannon Jensen divergences plotted along time. (left) A comparison between JAMUN and molecular Dynamics where the x-axis is a trajectory progress coordinate such that all JAMUN runs are for 640,000 samples while molecular Dynamics runs are truncated to be the same GPU time. (right) Shannon-Jensen divergence of a long fully converged molecular dynamics trajectory.

As a convergence metric, we look at Shannon-Jensen divergence in Figure 8. This allows us to compare the rate at which JAMUN and MD converge to the "correct" distribution. We calculate the Shannon-Jensen divergence for molecular dynamics and JAMUN runs of equal GPU time (using the data generated for Figure 6 and Figure 9). We also analyze fully converged molecular dynamics runs for reference. It appears that while JAMUN converges quicker 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 re-

produce 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 Shannon-Jensen 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 diamines with thermodynamic fidelity. We demonstrate this with the use of Markov State Models trained on converged MD data in Appendix B. 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.

## 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 Walk-Diffuse, 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.

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.

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.

Alexandre Duval, Simon V Mathis, Chaitanya K Joshi, Victor Schmidt, Santiago Miret, Fragkiskos D Malliaros, Taco Cohen, Pietro Lio, Yoshua Bengio, and Michael Bronstein. A Hitchhiker's Guide to Geometric GNNs for 3D Atomic Systems. *arXiv preprint arXiv:2312.07511*, 2023.

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.

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*, 2024.

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.

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 Code Availability

We will be releasing all the code and data necessary to reproduce this paper.

## B Distribution analyses

We demonstrate the quantitative gain in sampling through metrics like the Shannon-Jensen 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 figures 10-16, 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 $\phi$, small metastability in the $(-\pi, \pi)$ area and even the specific shapes of commonly occuring basins can be signatures particular to specific diamine pairs and are well-emulated by JAMUN.

## C Input Normalization

Fix an $(i, j) \in E$ from Equation 10. As $\eta_i, \eta_j \overset{\text{iid}}{\sim} \mathcal{N}(0, \mathbb{I}_D)$, we have $\eta_i - \eta_j \sim \mathcal{N}(0, 2\mathbb{I}_D)$ from the closure of the multivariate Gaussian under linear combinations. Thus, for each component

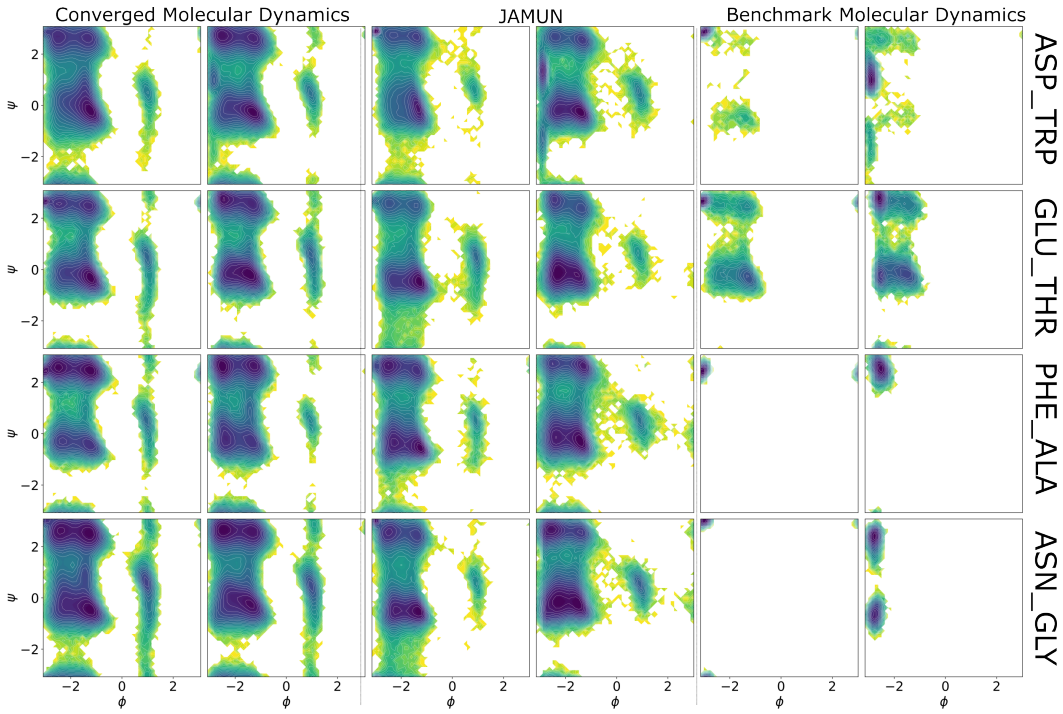


Figure 9: -log probabilities of outputs from four example diamines from the test set. These, along with 6, are chosen to represent the diversity of states as well as subtle perturbations from single-point amino acid changes in the dataset. Each pair of Ramachandran plots show the distribution for a single diamine, with each row corresponding to diamine identity. The first column shows the fully converged molecular dynamics distribution. The second shows sixty four thousand samples from JAMUN. The third column shows a distribution obtained from an MD trajectory run for the exact same amount of time as the JAMUN sampler.

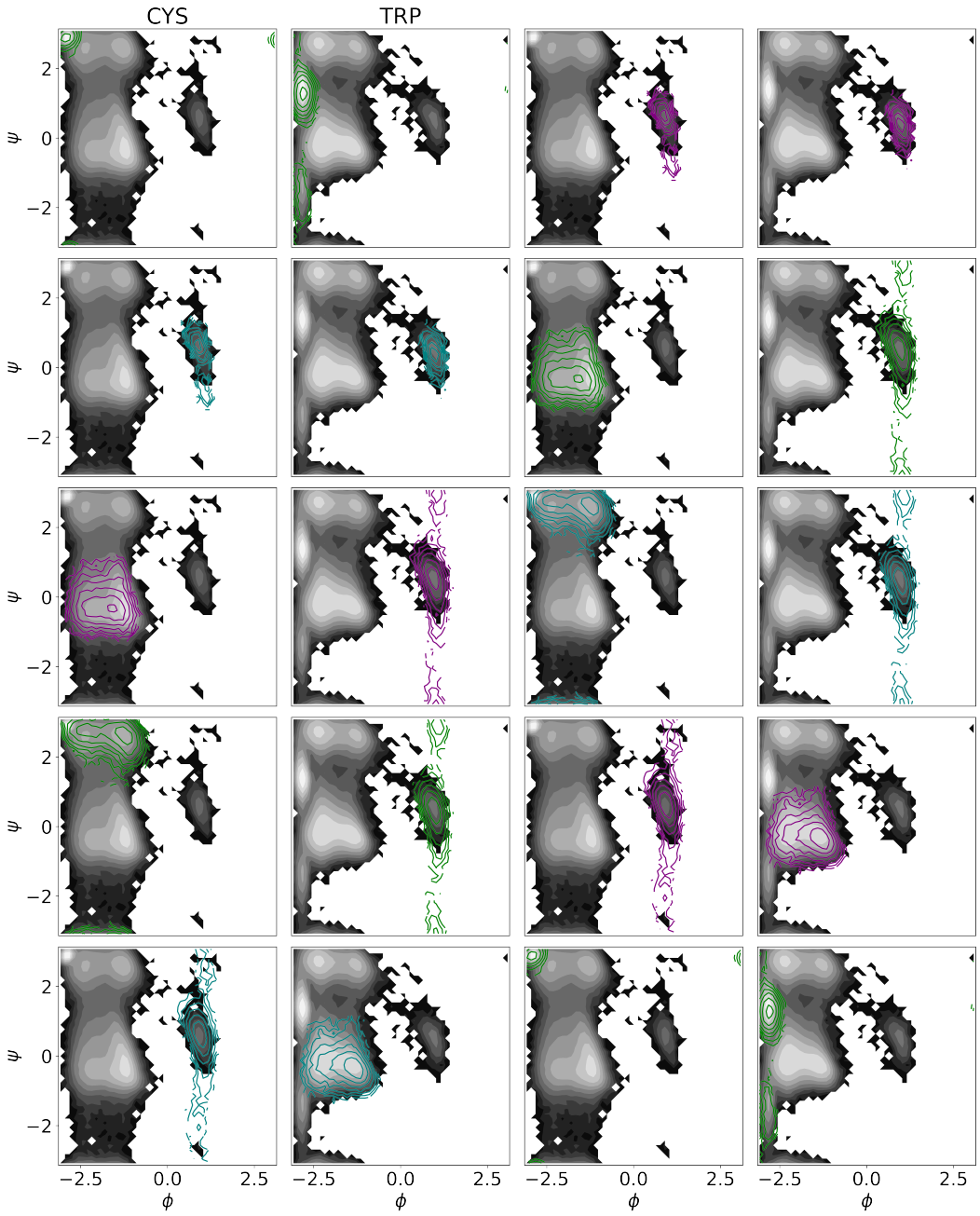


Figure 10: Markov states superimposed on sampled PMFs for CYS-TRP. This figure shows the results of running JAMUN with Markov state model negative log histograms superimposed on them. The MSMs are trained on fully converged MD data and represent metastable states. Each pair of Ramachandran plots correspond to a single metastable basin. Note that the gray-scale represents JAMUN samples while the colored is from molecular dynamics.

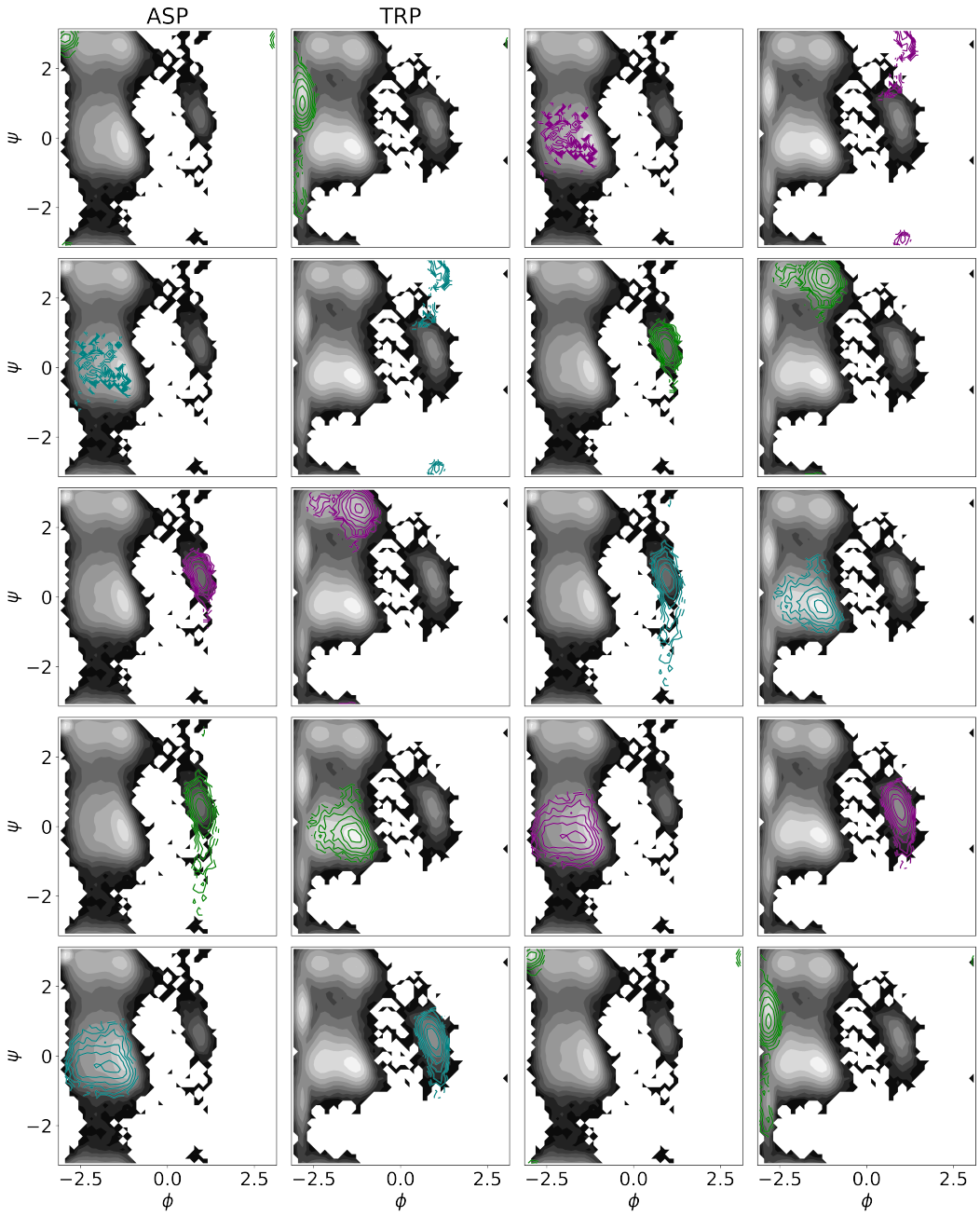


Figure 11: Markov states superimposed on sampled PMFs for ASP-THR. This figure shows the results of running JAMUN with Markov state model negative log histograms superimposed on them. The MSMs are trained on fully converged MD data and represent metastable states. Each pair of Ramachandran plots correspond to a single metastable basin. Note that the gray-scale represents JAMUN samples while the colored is from molecular dynamics.

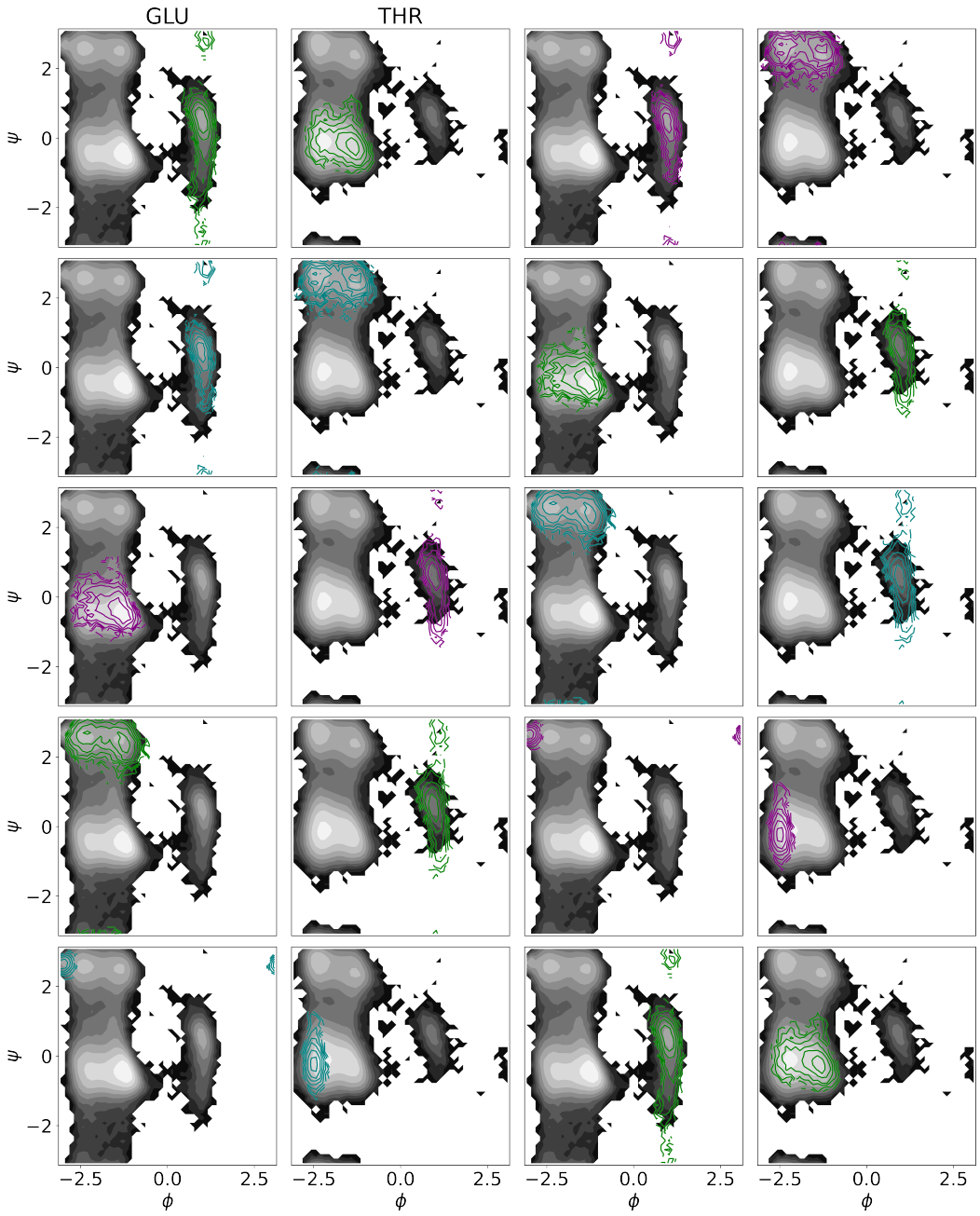


Figure 12: Markov states superimposed on sampled PMFs for GLU-THR. This figure shows the results of running JAMUN with Markov state model negative log histograms superimposed on them. The MSMs are trained on fully converged MD data and represent metastable states. Each pair of Ramachandran plots correspond to a single metastable basin. Note that the gray-scale represents JAMUN samples while the colored is from molecular dynamics.

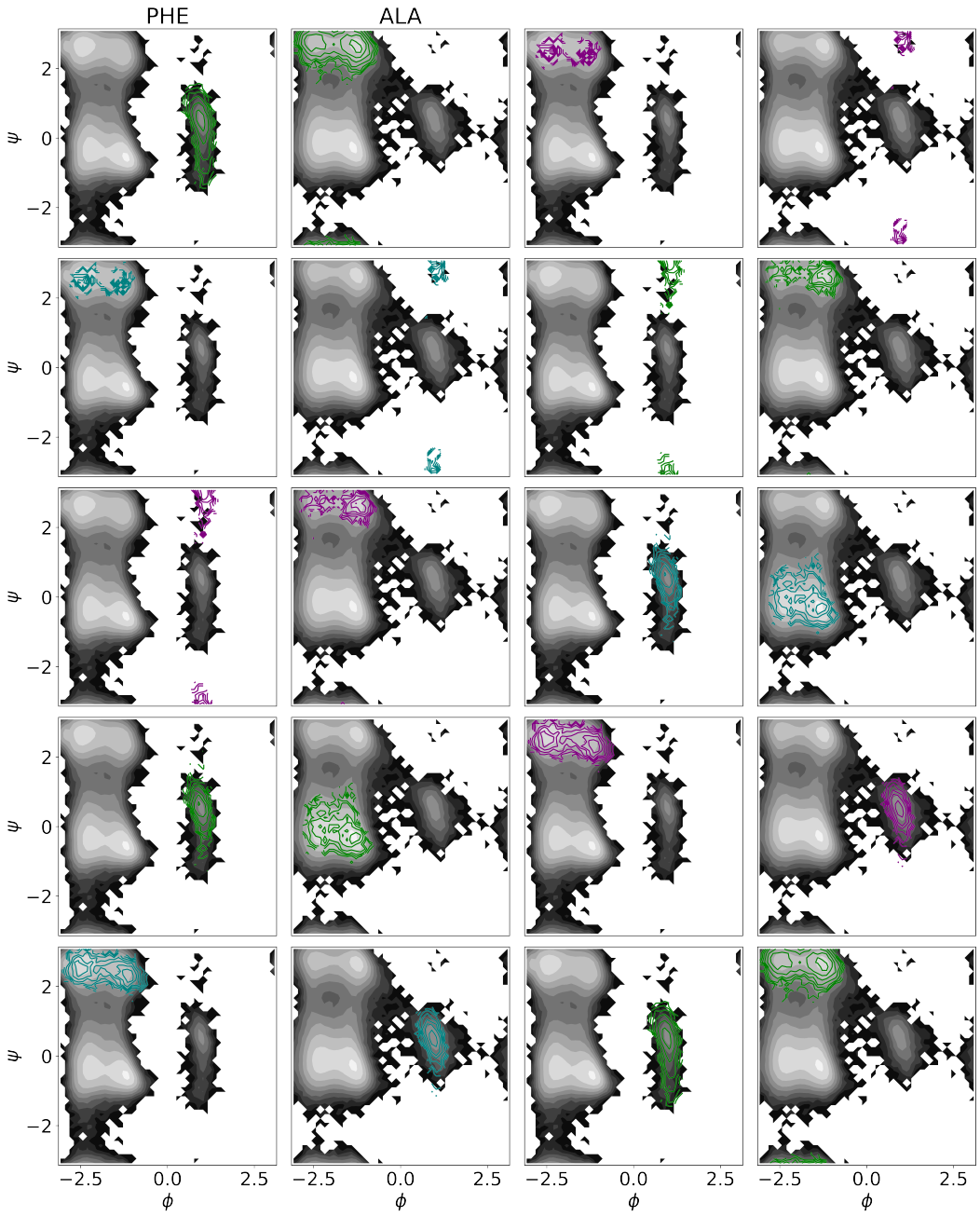


Figure 13: Markov states superimposed on sampled PMFs for PHE-ALA. This figure shows the results of running JAMUN with Markov state model negative log histograms superimposed on them. The MSMs are trained on fully converged MD data and represent metastable states. Each pair of Ramachandran plots correspond to a single metastable basin. Note that the gray-scale represents JAMUN samples while the colored is from molecular dynamics.

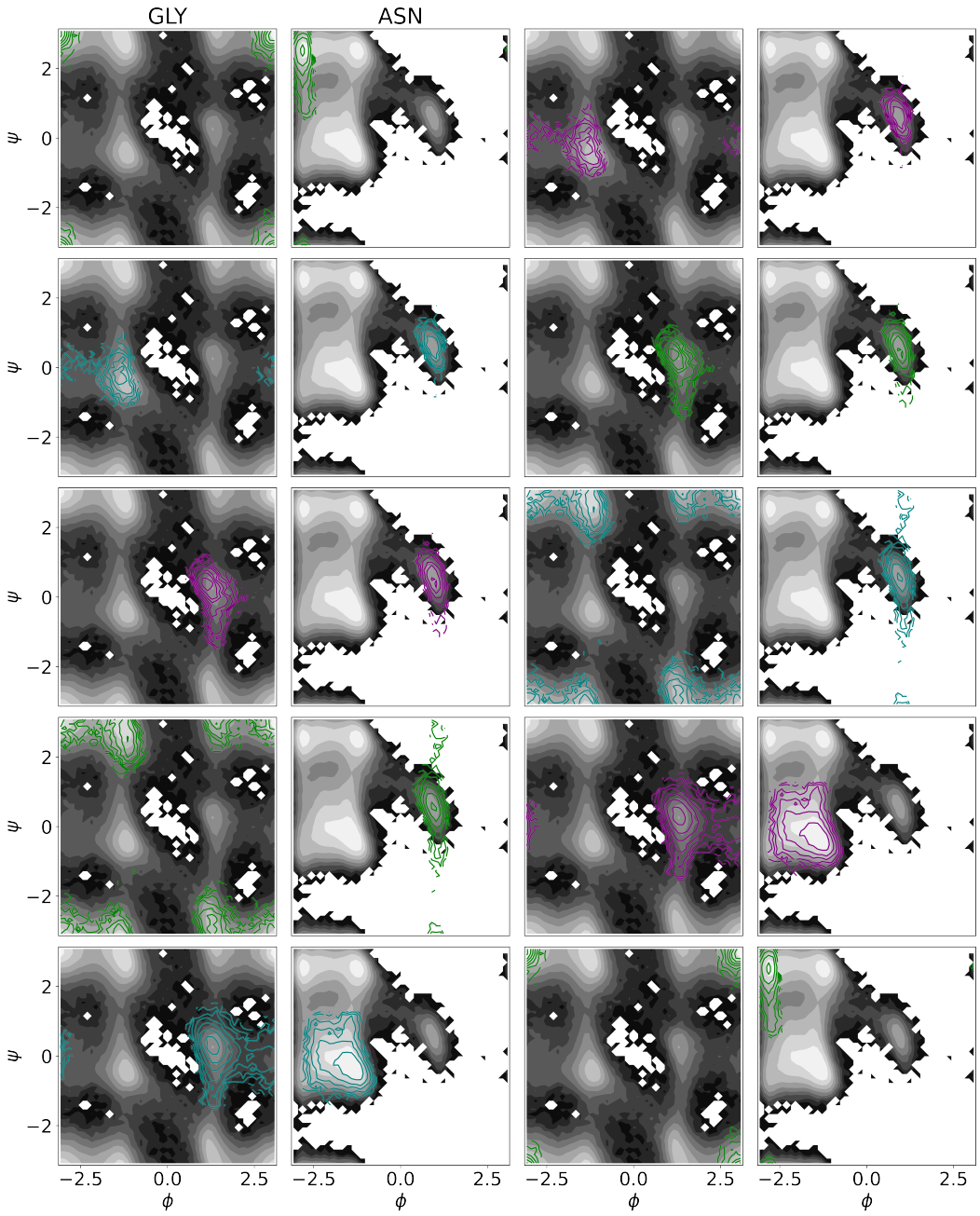


Figure 14: Markov states superimposed on sampled PMFs for GLY-ASN. This figure shows the results of running JAMUN with Markov state model negative log histograms superimposed on them. The MSMs are trained on fully converged MD data and represent metastable states. Each pair of Ramachandran plots correspond to a single metastable basin. Note that the gray-scale represents JAMUN samples while the colored is from molecular dynamics.

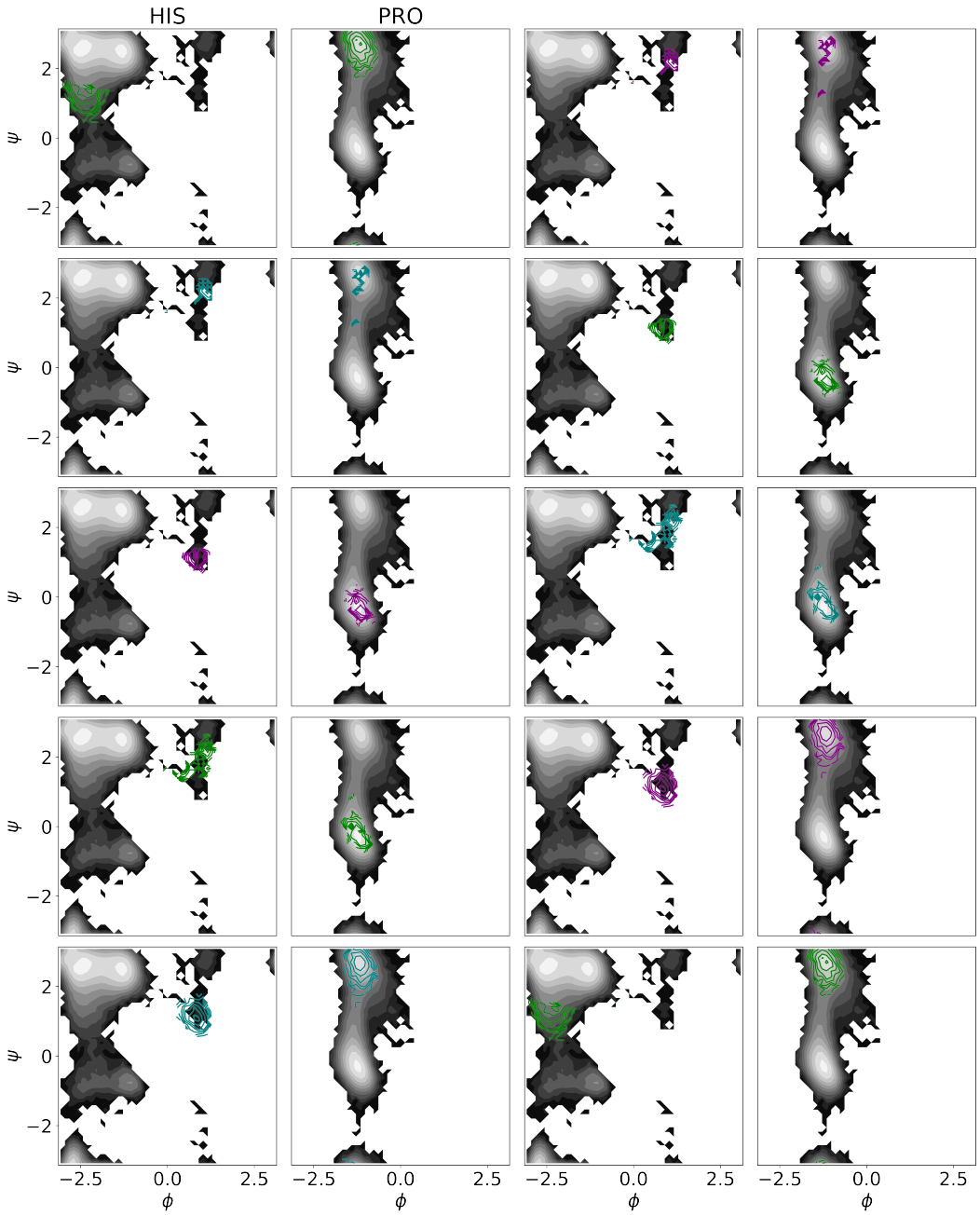


Figure 15: Markov states superimposed on sampled PMFs for HIS-PRO. This figure shows the results of running JAMUN with Markov state model negative log histograms superimposed on them. The MSMs are trained on fully converged MD data and represent metastable states. Each pair of Ramachandran plots correspond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

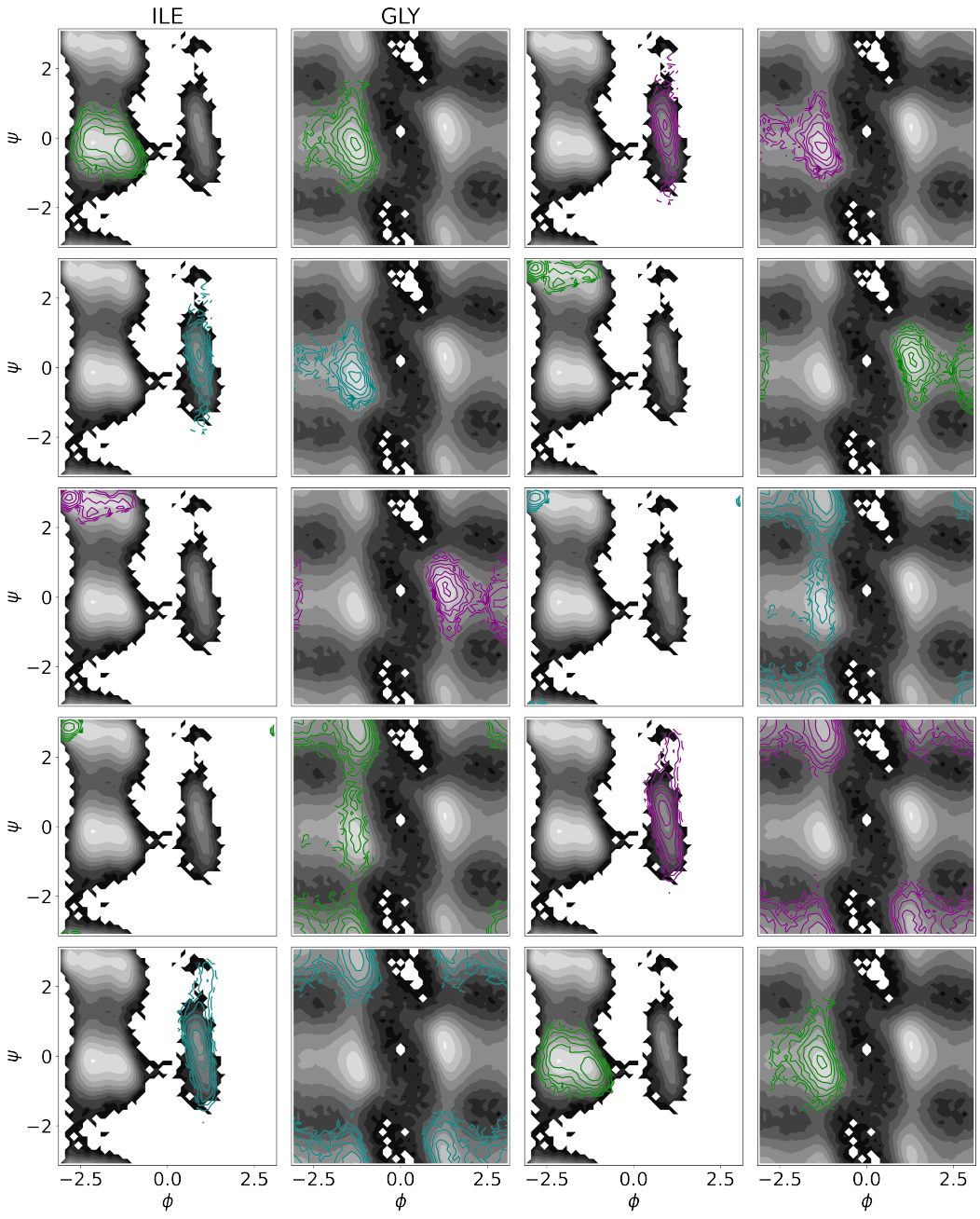


Figure 16: Markov states superimposed on sampled PMFs for ILE-GLY. This figure shows the results of running JAMUN with Markov state model negative log histograms superimposed on them. The MSMs are trained on fully converged MD data and represent metastable states. Each pair of Ramachandran plots correspond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

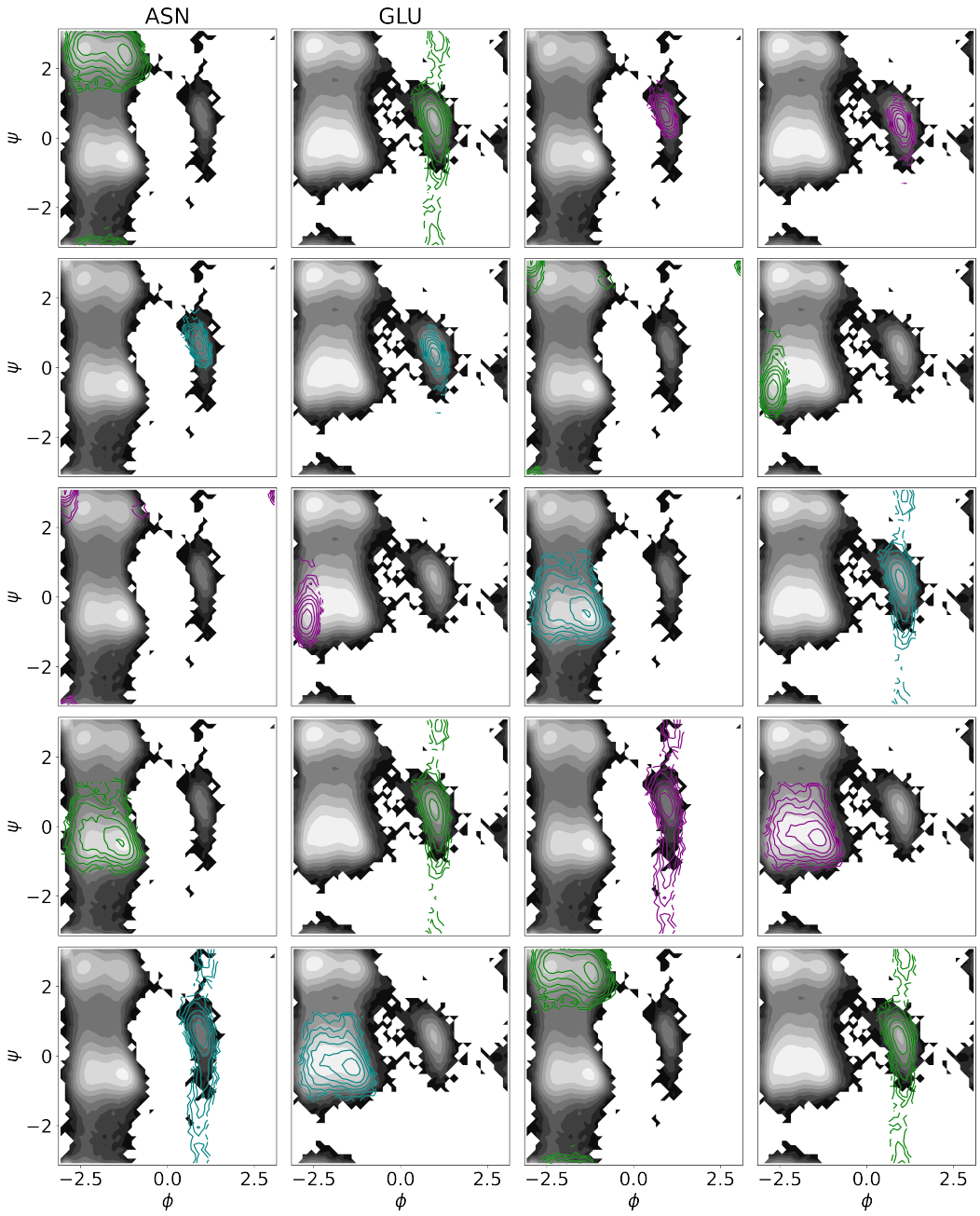


Figure 17: Markov states superimposed on sampled PMFs for ASN-GLU. This figure shows the results of running JAMUN with Markov state model negative log histograms superimposed on them. The MSMs are trained on fully converged MD data and represent metastable states. Each pair of Ramachandran plots correspond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

$d \in \{1, \dots D\}$, we have: $(\eta_i - \eta_j)_{(d)} \sim \mathcal{N}(0, 2)$ and hence:

$$\mathbb{E}_{\eta \sim \mathcal{N}(0, \mathbb{I}_{D \times N})}[(x_i - x_j)^T(\eta_i - \eta_j)] = \sum_{d=1}^{D} (x_i - x_j)_{(d)} \mathbb{E}[(\eta_i - \eta_j)_{(d)}] = 0 \tag{17}$$

$$\mathbb{E}_{\eta \sim \mathcal{N}(0, \mathbb{I}_{D \times N})}[\|\eta_i - \eta_j\|^2] = \sum_{d=1}^{D} \mathbb{E}[(\eta_i - \eta_j)_{(d)}^2] = 2D \tag{18}$$

We can now compute:

$$\begin{aligned}
&\mathbb{E}_z[\|\tilde{y}_i - \tilde{y}_j\|^2] \\
&= c_{\text{in}}(\sigma)^2 \mathbb{E}_z[\|y_i - y_j\|^2] \\
&= c_{\text{in}}(\sigma)^2 \mathbb{E}_z[\|x_i - x_j + \sigma(\eta_i - \eta_j)\|^2] \\
&= c_{\text{in}}(\sigma)^2 \left( \|x_i - x_j\|^2 + 2\sigma \mathbb{E}_z[(x_i - x_j)^T(\eta_i - \eta_j)] + \sigma^2 \mathbb{E}_z[\|\eta_i - \eta_j\|^2] \right) \\
&= c_{\text{in}}(\sigma)^2 \left( \|x_i - x_j\|^2 + \sigma^2 \mathbb{E}_z[\|\eta_i - \eta_j\|^2] \right) \\
&= c_{\text{in}}(\sigma)^2 \left( \|x_i - x_j\|^2 + 2D\sigma^2 \right).
\end{aligned} \tag{19}$$

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

$$\begin{aligned}
\mathbb{E}_{\substack{(i,j) \sim \text{Uniform}(E) \\ \eta \sim \mathcal{N}(0, \mathbb{I}_{D \times N})}}[\|\tilde{y}_i - \tilde{y}_j\|^2] &= \mathbb{E}_{(i,j) \sim \text{Uniform}(E)}[\mathbb{E}_z[\|\tilde{y}_i - \tilde{y}_j\|^2]] \\
&= c_{\text{in}}(\sigma)^2 \left( \mathbb{E}_{(i,j) \sim \text{Uniform}(E)} \|x_i - x_j\|^2 + 2D\sigma^2 \right)
\end{aligned} \tag{20}$$

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 20 and our intended normalization given by Equation 10:

$$c_{\text{in}}(\sigma) = \frac{1}{\sqrt{C + 2D\sigma^2}} \tag{21}$$

## D 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:

$$\mathcal{L}(\hat{x}_\theta, \sigma) = \mathbb{E}_{x \sim p_x} \mathbb{E}_{\eta \sim \mathcal{N}(0, \mathbb{I})}[\|\hat{x}_\theta(x + \sigma\eta, \sigma) - x\|^2] \tag{22}$$

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

$$\begin{aligned}
\mathcal{L}(\hat{x}_\theta) &= \mathbb{E}_{\sigma \sim p_\sigma}[\lambda(\sigma) \mathcal{L}(\hat{x}_\theta, \sigma)] \\
&= \mathbb{E}_{\sigma \sim p_\sigma} \mathbb{E}_{x \sim p_x} \mathbb{E}_{\eta \sim \mathcal{N}(0, \mathbb{I})}[\lambda(\sigma) \|\hat{x}_\theta(x + \sigma\eta, \sigma) - x\|^2] \\
&= \mathbb{E}_{\sigma \sim p_\sigma} \mathbb{E}_{x \sim p_x} \mathbb{E}_{y \sim \mathcal{N}(x, \sigma^2 \mathbb{I})}[\lambda(\sigma) \|\hat{x}_\theta(y, \sigma) - x\|^2] \\
&= \mathbb{E}_{\sigma \sim p_\sigma} \mathbb{E}_{x \sim p_x} \mathbb{E}_{y \sim \mathcal{N}(x, \sigma^2 \mathbb{I})}[\lambda(\sigma) \|c_{\text{skip}}(\sigma) y + c_{\text{out}}(\sigma) F_\theta(c_{\text{in}}(\sigma) y, c_{\text{noise}}(\sigma)) - x\|^2] \\
&= \mathbb{E}_{\sigma \sim p_\sigma} \mathbb{E}_{x \sim p_x} \mathbb{E}_{y \sim \mathcal{N}(x, \sigma^2 \mathbb{I})} \left[ \lambda(\sigma) c_{\text{out}}(\sigma)^2 \left\| F_\theta(c_{\text{in}}(\sigma) y, c_{\text{noise}}(\sigma)) - \frac{x - c_{\text{skip}}(\sigma) y}{c_{\text{out}}(\sigma)} \right\|^2 \right] \\
&= \mathbb{E}_{\sigma \sim p_\sigma} \mathbb{E}_{x \sim p_x} \mathbb{E}_{y \sim \mathcal{N}(x, \sigma^2 \mathbb{I})} \left[ \lambda(\sigma) c_{\text{out}}(\sigma)^2 \|F_\theta(c_{\text{in}}(\sigma) y, c_{\text{noise}}(\sigma)) - F\|^2 \right]
\end{aligned} \tag{23}$$

where:

$$F = \frac{x - c_{\text{skip}}(\sigma) y}{c_{\text{out}}(\sigma)} \tag{24}$$

is the effective training target for the network $F_\theta$. We want to normalize $F$ similarly as the network input:

$$\mathbb{E}_{\substack{(i,j) \sim \text{Uniform}(E) \\ \eta \sim \mathcal{N}(0, \mathbb{I}_{D \times N})}}[\|F_i - F_j\|^2] = 1 \text{ at all noise levels } \sigma. \tag{25}$$

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

$$\begin{aligned}
\mathbb{E}_z \left\| F_i - F_j \right\|^2 &= \frac{\mathbb{E}_z \left\| (x_i - x_j) - c_{\text{skip}}(\sigma)(y_i - y_j) \right\|^2}{c_{\text{out}}(\sigma)^2} \\
&= \frac{\mathbb{E}_z \left\| (1 - c_{\text{skip}}(\sigma))(x_i - x_j) - c_{\text{skip}}(\sigma)\sigma \cdot (\eta_i - \eta_j) \right\|^2}{c_{\text{out}}(\sigma)^2} \\
&= \frac{(1 - c_{\text{skip}}(\sigma))^2 \left\| x_i - x_j \right\|^2 + c_{\text{skip}}(\sigma)^2 \cdot 2D\sigma^2}{c_{\text{out}}(\sigma)^2}
\end{aligned} \tag{26}$$

and hence:

$$\begin{aligned}
&\mathbb{E}_{\substack{(i,j)\sim\text{Uniform}(E) \\ \eta\sim\mathcal{N}(0,\mathbb{I}_{D\times N})}}[\left\| F_i - F_j \right\|^2] = 1 \\
\implies &\frac{(1 - c_{\text{skip}}(\sigma))^2 \cdot C + c_{\text{skip}}(\sigma)^2 \cdot 2D\sigma^2}{c_{\text{out}}(\sigma)^2} = 1 \\
\implies &c_{\text{out}}(\sigma)^2 = (1 - c_{\text{skip}}(\sigma))^2 \cdot C + c_{\text{skip}}(\sigma)^2 \cdot 2D\sigma^2
\end{aligned} \tag{27}$$

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):

$$\begin{aligned}
&\frac{d}{dc_{\text{skip}}(\sigma)} c_{\text{out}}(\sigma)^2 = 0 \\
\implies &-2(1 - c_{\text{skip}}(\sigma)) \cdot C + 2c_{\text{skip}}(\sigma) \cdot 2D\sigma^2 = 0 \\
\implies &c_{\text{skip}}(\sigma) = \frac{C}{C + 2D\sigma^2}
\end{aligned} \tag{28}$$

Substituting into Equation 27, we get after some routine simplification:

$$c_{\text{out}}(\sigma) = \sqrt{\frac{C \cdot 2D\sigma^2}{C + 2D\sigma^2}} \tag{29}$$

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 23, we set $\lambda(\sigma) = \frac{1}{c_{\text{out}}(\sigma)^2}$.

## E THE DENOISER MINIMIZES THE EXPECTED LOSS

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

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

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

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

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

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

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

and hence, any minimizer $a^*$ must minimize the local denoising loss $l(a^*, y)$ at each point $y \in \mathbb{R}^{N\times 3}$. For a fixed $y \in \mathbb{R}^{N\times 3}$, the loss $l(a, y)$ is convex, and hence, the global minimizer can be

found by finding the critical points of $l(a, y)$ as a function of $a(y)$:

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

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

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

Rearranging:

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

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

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

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

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

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

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

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

as claimed.

## F 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.

## G Numerical Solvers for Langevin Dynamics

As mentioned in Section 3.1, 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 H.

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

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

## H THE ORNSTEIN-UHLENBECK PROCESS

For completeness, we discuss the distributional solution of the Ornstein-Uhlenbeck process, taken directly from the excellent Leimkuhler & Matthews (2015). 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:

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

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

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

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

The analysis above is performed in one dimension, but readily applies to the $N \times 3$ dimensional case as the Wiener processes are all independent of each other:

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

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

## I HYPERPARAMETERS

All the models trained in this work are $SE(3)$-equivariant networks

We use a uniform distribution of noise with a range of .01 to .1, There are four hidden layers. Graphs are constructed using bonds as well as neighbors within radius of five. We use ADAM with learning rate .002. For the capped diamines we use 320,000 snapshots for 200 training diamines, whereas for the uncapped we use the timewarp-Large dataset. Models are trained with six parallel processing GPUs and a batch size of forty-two. The model has 8.2 million parameters. For the hidden layers of the network we use 120 scalar and 32 pseudovector features per node and spherical harmonics up to $l = 1$ The output is 1 vector per node.

For sampling capped diamines, we use $\sigma = .04$, whereas for uncapped we use $\sigma = .06$. We always use friction of $\gamma = 0.1$ and a Langevin dynamics step size of $\delta = \sigma$.