Algorithm 1 Timewarp MCMC with batched proposals

Require: Initial state $X_0 = (X_0^p, X_0^v)$, chain length M, proposal batch size B.
  $m \leftarrow 0$
  while $m < M$ **do**
    Sample $\tilde{X}_1, \ldots, \tilde{X}_B \sim p_\theta(\cdot | X_m^p)$ {Batch sample}
    for $b = 1, \ldots, B$ **do**
      $\epsilon \sim \mathcal{N}(0, I)$ {Resample auxiliary variables}
      $X_b \leftarrow (X_m^p, \epsilon)$
      Sample $I_b \sim \text{Bernoulli}(\alpha(X_b, \tilde{X}_b))$
    end for
    if $S := \{b : I_b = 1, 1 \leq b \leq B\} \neq \emptyset$ **then**
      $a = \min(S)$ {First accepted sample}
      $(X_{m+1}^p, \ldots, X_{m+a-1}^p) \leftarrow (X_m^p, \ldots, X_m^p)$
      $X_{m+a}^p \leftarrow \tilde{X}_a^p$
      $m \leftarrow m + a$
    else
      $(X_{m+1}^p, \ldots, X_{m+B}^p) \leftarrow (X_m^p, \ldots, X_m^p)$
      $m \leftarrow m + B$
    end if
  end while
output $X_0^p, \ldots X_M^p$

3.5. Fast exploration of the state space without MH

Although the MH correction ensures that Timewarp provides asymptotically unbiased samples, it can lead to much slower exploration of the state space due to the rejected proposals. For some of the peptides we consider, the acceptance probabilities are too low to apply Algorithm 1 effectively. Instead, we can apply Timewarp in a simple *exploration* algorithm, where we ignore the MH correction and accept all proposals with an energy change lower than some cutoff. This allows much faster exploration of the state space, and in Section 6 we show that the algorithm, although technically biased, often leads to qualitatively accurate free energy estimates. It also succeeds in discovering all metastable states of a peptide orders of magnitude faster than Algorithm 1 and standard MD. Timewarp applied in exploration mode can be used to efficiently find the metastable states of a new molecule, which could be used, *e.g.*, to provide initialisation states for a subsequent MSM method, although we do not pursue this here. We provide pseudocode for the exploration algorithm in Algorithm 2 in Appendix C.

4. Model architecture

We now describe the architecture of our conditional normalising flow $f_\theta(z^p, z^v; x^p(t))$, which is shown in Figure 2.

RealNVP coupling flow Our architecture is based on RealNVP (Dinh et al., 2017), which consists of a stack of *coupling layers* which affinely transform subsets of the dimensions of the latent variable based on the other dimensions. Specifically, we transform the position variables based on the auxiliary variables, and vice versa. In the ℓth coupling layer of the flow, the following transformations are implemented:

$$z_{\ell+1}^p = s_{\ell,\theta}^p(z_\ell^v; x^p(t)) \odot z_\ell^p + t_{\ell,\theta}^p(z_\ell^v; x^p(t)), \quad (9)$$

$$z_{\ell+1}^v = s_{\ell,\theta}^v(z_{\ell+1}^p; x^p(t)) \odot z_\ell^v + t_{\ell,\theta}^v(z_{\ell+1}^p; x^p(t)). \quad (10)$$

Going forward, we suppress the coupling layer index ℓ. Here $\odot$ is the element-wise product, and $s_\theta^p : \mathbb{R}^{3N} \to \mathbb{R}^{3N}$ is our *atom transformer*, a neural network based on the transformer architecture (Vaswani et al., 2017) that takes the auxiliary latent variables z^v and the conditioning state $x(t)$ and outputs scaling factors for the position latent variables z^p. The function $t_\theta^p : \mathbb{R}^{3N} \to \mathbb{R}^{3N}$ is implemented as another atom transformer, which uses z^v and $x(t)$ to output a translation of the position latent variables z^p. The affine transformations of the position variables (in Equation (9)) are interleaved with similar affine transformations for the auxiliary variables (in Equation (10)). Since the scale and translation factors for the positions depend only on the auxiliary variables, and vice versa, the Jacobian of the transformation is lower triangular, allowing for efficient computation of the density. The full flow f_θ consists of N_{coupling} stacked coupling layers, beginning from $z \sim \mathcal{N}(0, I)$ and ending with a sample from $p_\theta(x(t+\tau)|x(t))$. This is depicted in Figure 2, Left. Note that there is a skip connection such that the output of the flow predicts the *change* $x(t+\tau) - x(t)$, rather than outputting $x(t+\tau)$ directly.

Atom transformer We now describe the *atom transformer* network. Let $x_i^p(t), z_i^p, z_i^v$, all elements of $\mathbb{R}^3$, denote respectively the position of atom i in the conditioning state, the position latent variable for atom i, and the auxiliary latent variable for atom i. To implement an atom transformer which takes z^v as input (such as $s_\theta^p(z^v, x^p(t))$ and $t_\theta^p(z^v, x^p(t))$ in Equation (9)), we first concatenate the variables associated with atom i. This leads to a vector $a_i := [x_i^p(t), h_i, z_i^v]^\mathsf{T} \in \mathbb{R}^{H+6}$, where z_i^p has been excluded since s_θ^p, t_θ^p are not allowed to depend on z^p. Here $h_i \in \mathbb{R}^H$ is a learned embedding vector which depends only on the atom type. The vectors a_i are fed into an MLP $\phi_{\text{in}} : \mathbb{R}^{H+6} \to \mathbb{R}^D$, where D is the feature dimension of the transformer. The vectors $\phi_{\text{in}}(a_1), \ldots, \phi_{\text{in}}(a_N)$ are then fed into $N_{\text{transformer}}$ stacked transformer layers. After the transformer layers, they are passed through another atom-wise MLP, $\phi_{\text{out}} : \mathbb{R}^D \to \mathbb{R}^3$. The final output is in $\mathbb{R}^{3N}$ as required. This is depicted in Figure 2, Middle. When implementing s_θ^v and t_θ^v from Equation (10), a similar procedure is performed on the vector $[x_i^p(t), h_i, z_i^p]^\mathsf{T}$, but now including z_i^p and excluding z_i^v. There are two key differences between the atom transformer and the architecture in Vaswani et al. (2017). First, to maintain permutation equivariance, we do