Tutorials on How to Build Non-Markovian Dynamic Models from Molecular Dynamics Simulations for Studying Protein Dynamics

Yue Wu[1], Siqin Cao[1], Yunrui Qiu[1], Xuhui Huang[1,2*]

[1]Department of Chemistry, Theoretical Chemistry Institute, University of Wisconsin-Madison, Madison, WI, 53706, USA

[2]Data Science Institute, University of Wisconsin-Madison, Madison, WI, 53706, USA

[*]To whom correspondence should be addressed. E-mail: xhuang@chem.wisc.edu

Abstract

Protein conformational changes play crucial roles in their biological functions. In recent years, the Markov State Model (MSM) constructed from extensive Molecular Dynamics (MD) simulations has emerged as a powerful tool for modeling complex protein conformational changes. In MSMs, dynamics are modeled as a sequence of Markovian transitions among metastable conformational states at discrete time intervals (called lag time). A major challenge for MSM is that the lag time must be long enough to allow transitions among states to become memoryless (or Markovian). However, this lag time is constrained by the length of individual MD simulations available to track these transitions. To address this challenge, we have recently developed Generalized Master Equation (GME)-based approaches, encoding non-Markovian dynamics using a time-dependent memory kernel. In this tutorial, we introduce the theory behind two recently developed GME-based non-Markovian dynamic models: the Quasi Markov State Model (qMSM) and the Integrative Generalized Master Equation (IGME). We subsequently outline the procedures for constructing these models and provide a step-by-step tutorial on applying qMSM and IGME to study two peptide systems: the alanine dipeptide and villin headpiece. This tutorial is available at https://github.com/xuhuihuang/GME_tutorials. The protocols detailed in this tutorial aim to be accessible for non-experts interested in studying the biomolecular dynamics using these non-Markovian dynamic models.

1. Introduction

Protein functions heavily rely on dynamic transitions between conformational states or conformational changes[1, 2]. For example, RNA polymerase II needs to repeatedly oscillate between the pre-translocation state and post-translocation state to translocate along the doble-strand DNA during transcription elongation[3, 4]. Therefore, studying conformational changes is crucial for understanding the molecular mechanisms underlying many biological processes[4] and facilitating drug design targeting these conformational changes[5, 6].

Molecular dynamics (MD) simulations have emerged as a valuable tool that can complement experimental approaches by providing atomistic details of protein dynamics. However, the timescales of conformational changes for complex biological molecules often exceed the feasible simulation length. For example, the opening of the DNA loading gate of RNA polymerase occurs at millisecond timescales, while it still remains challenging for all-atom MD simulations of RNA polymerase (the simulation box contains around half a million atoms) to reach a millisecond[7]. Markov State Model (MSM)[8-19] has become a popular approach to bridge this timescale gap by modeling the long timescale dynamics based on many short MD simulations.

In MSMs, the high-dimensional conformational space is partitioned into a set of discrete and coarse-grained metastable states. Simultaneously, through the coarse-graining of time (using discrete time intervals or lag time), transitions among these states can be modelled as Markovian jumps. The transition probabilities after a lag time between pairs of states can then be estimated from short MD simulations. As a result, MSMs can predict long timescale dynamics based on a large ensemble of short MD simulations. In recent years, MSMs have been widely applied to study protein folding[18, 20-25], protein-ligand binding[26-29], and functional conformational changes of biomolecules[17, 30-47].

One critical condition for MSMs to have predicting power is that the lag time must be long enough to allow transitions among states to become memoryless (or Markovian), and the memory of these transitions is mainly determined by dynamic relaxations within each metastable state. This imposes a major challenge for MSM studies of protein dynamics, as the lag time is bound by the length of individual MD simulations available to estimate transition probabilities. To achieve Markovian transitions, one often needs to construct MSMs containing a large number of states, so that each state is sufficiently small and has relatively fast relaxation dynamics to allow affordable lag times. For example, Pande and coworkers showed that they need an MSM containing 2,000 states (with a lag time of 12ns) to model the millisecond folding of the NTL9 peptide[48]. Our previous work suggested that 10,000 state is needed for a MSM to be Markovian (with a lag time of 5ns) for a 37-residue intrinsically disordered peptide[20]. More recently, our work on the RNA Polymerase II backtracking also showed that MSMs consisting of 800 states are needed to reach Markovian[33]. MSMs containing hundreds of states are useful to make quantitative predictions to be tested by experiments, but often hinder the comprehension of biological mechanisms. In recent years, various methods, such as non-Markovian dynamic approaches[49] based on Generalized Master Equation (GME) [50-52] or Generalized Langevin Equation (GLE)[53-55], and Hidden Markov State Models[56] have been developed to address this challenge.

The GME has particularly emerged as a promising approach to address the aforementioned challenge of studying protein dynamics[49]. The GME method (also called quasi-MSM or qMSM) is a non-Markovian dynamic model and explicitly considers the memory kernel and propagates dynamics using a discretized GME[50]. The qMSM method is shown to greatly improve upon MSMs by accurately predicting long-timescale dynamics while being built from significantly shorter MD simulations[7, 50, 57]. The Transfer Tensor Method (TTM) provides an analogous approach to qMSM that utilizes a discretization of the integrated GME using the time-dependent transfer tensor[58]. In addition, Dill and co-authors introduced the memory kernel (NMMK) method, where they applied the maximum entropy principle to obtain the memory kernels from experimental data[59].

GME represents a new and promising approach to studying biomolecular dynamics. However, a major challenge persists in the robustness of the computed time-dependent memory kernels when applying GME to investigate complex conformational changes[49]. This challenge arises because memory kernels are estimated based on probabilities of transitions among states at a series of time points, and the fluctuations encountered when extracting these time-dependent transition probabilities from MD trajectories could induce numerical instability in complex systems. To address this issue, we have recently developed the Integrative GME (IGME) method[51]. In this method, we first derived an analytical solution for the GME under the condition that the memory kernels are fully decayed. Subsequently, we determine the hyper-parameters in this analytical solution through fitting to MD simulations. As IGME deals with the condition that the memory kernels have already been decayed, it employs only the time integrations of memory kernels, thereby avoiding the numerical instability associated with the explicit computation of time-dependent memory kernels in qMSM. When applied to the study of peptide dynamics, the IGME models demonstrate significantly reduced fluctuations in both memory kernels and predicted long-term dynamics compared to qMSM[50]. In addition to IGME, the time-convolutionless GME (TCL-GME) provides another noise-resilient GME-based approach, where the non-Markovian time evolutions of dynamics are modeled by a time-dependent rate matrix[52].

In this tutorial, we present a step-by-step guide on how to build qMSM and IGME models to study protein dynamics. This tutorial is accompanied by our most recent implementation of these two GME-based methods, available on GitHub (https://github.com/xuhuihuang/GME_tutorials). We hope this provides a detailed tutorial for researchers in the computational chemistry and biophysics community to learn how to build GME methods for studying conformational dynamics of proteins and other macromolecules. Our paper is organized as follows: we first introduce the GME (for the qMSM method) and IGME theories, followed by outlining detailed and step-by-step protocols for building these models from MD simulations. Finally, we will present two detailed examples (alanine dipeptide and the villin headpiece) along with associated Python code (presented as Schemes) to demonstrate how to build qMSM and IGME models from MD simulations.

2. Theories of non-Markovian dynamic models for protein dynamics

2.1. Liouville equation and dynamic operators

The dynamics in the phase space follows the Liouville equation:

$$\partial\rho(t,\boldsymbol{\Gamma})/\partial t = \mathcal{L}\rho(t,\boldsymbol{\Gamma}) \quad (1)$$

where the Liouville operator $\mathcal{L}$ encapsulates all pertinent information of the dynamic system and $\rho(t, \boldsymbol{\Gamma})$ represents the probability distribution function across the entire phase space $\boldsymbol{\Gamma}$ at time t.

Based on the Liouville equation, the evolution of the probability density after the time interval τ follows $\rho(t+\tau, \boldsymbol{\Gamma}) = e^{\mathcal{L}\tau}\rho(t, \boldsymbol{\Gamma})$, regardless of the starting time t or any travelling history. This property is called Markovian, or memoryless. When studying protein dynamics in reversible system, several dynamic operators have been used to describe the propagation of the distribution function, as listed below:

- Propagator $\mathcal{P}(\tau)$: $\rho(t+\tau, \boldsymbol{\Gamma}) = \mathcal{P}(\tau)\rho(t, \boldsymbol{\Gamma})$
- Transfer operator $\mathcal{T}(\tau)$ [9]: $\rho(t+\tau, \boldsymbol{\Gamma})/\rho_{eq} = \mathcal{T}(\tau)\big(\rho(t, \boldsymbol{\Gamma})/\rho_{eq}\big)$, where ρ_{eq} represents the equilibrium probability distribution function. Transfer operator can only be applied in reversible system.

2.2. Markov State Model (MSM) theory

In MSMs, the configurational space is partitioned into n states, represented as $\{X_i\}_{i=1}^{n}$, and simultaneously the continuous time is coarse-grained into discrete time intervals (τ, called lag times). The system's dynamics is then modelled as Markovian transitions among these states. Consequently, the probability density function $\rho(t)$ can be expressed as a vector containing n elements: $\boldsymbol{p}(t) = [p_1(t) \quad \cdots \quad p_n(t)]^{\mathrm{T}}$ where $p_i(t)$ represents the population in state X_i at time t. The Markovian property requires that the propagation of $\boldsymbol{p}(t)$ after the time interval τ can be seen as under the operation of the transition probability independent of t : $p_j(t+\tau) = \sum_{i=1}^{n} p_i(t)\,\mathbb{P}\big(\boldsymbol{x}(t+\tau) \in X_j \big| \boldsymbol{x}(t) \in X_i\big)$. Here, $\boldsymbol{x}$ represents the configuration and the transition probability is described as the conditional probability of jumping to state j after a lag time of τ, given the initial state i. By defining the transition probability matrix (TPM), $\boldsymbol{T}(\tau)$, as T_{ij}=$\mathbb{P}\big(\boldsymbol{x}(t+\tau) \in X_j \big| \boldsymbol{x}(t) \in X_i\big)$, the time propagation of $\boldsymbol{p}(t)$ can be rewritten as:

$$\boldsymbol{p}(t+\tau)^{\mathrm{T}} = \boldsymbol{p}(t)^{\mathrm{T}}\boldsymbol{T}(\tau) \tag{2}$$

We can then define the equilibrium density $[\pi_1 \quad \cdots \quad \pi_n]^{\mathrm{T}}$, where $\pi_i = \int_{\boldsymbol{x}\in X_i} \rho_{eq}(\boldsymbol{x})\mathrm{d}\boldsymbol{x}$. For equilibrium dynamics, the detailed balance condition imposes the relationship: $\pi_i^{\mathrm{T}} T_{ij} = \pi_j^{\mathrm{T}} T_{ji}$. If we further define $\boldsymbol{u}(t) = \boldsymbol{p}(t)/\boldsymbol{\pi} = [p_1(t)/\pi_1 \quad \cdots \quad p_n(t)/\pi_n]^{\mathrm{T}}$, the expression can be rewritten as:

$$\boldsymbol{u}(t+\tau) = \boldsymbol{T}(\tau)\boldsymbol{u}(t) \tag{3}$$

Where $\boldsymbol{T}(\tau)$ represents an approximation of the transfer operator $\mathcal{T}(\tau)$ at reduced dimensions (i.e., transitions among n discrete states). For the row normalized TPM, the leading left eigenvectors provide information of the population flux of the slowest dynamic processes, which correspond to transitions between metastable regions of the conformational space. The timescales of these dynamic processes are related to the eigenvalues of the TPM, as represented by their implied timescales (ITS)[60]:

$$\mathrm{ITS}_i(\tau) = -\frac{\tau}{\ln \lambda_i(\tau)} \tag{4}$$

Where $i = 1,2,3$ .... When the reduced dynamics in the state space with the lag time of τ are Markovian, we can use the first-order master equation to propagate the dynamics:

$$\boldsymbol{T}(n\tau) = \boldsymbol{T}(\tau)^n \tag{5}$$

Where $\boldsymbol{\pi}^{\mathrm{T}} = \boldsymbol{\pi}^{\mathrm{T}}\boldsymbol{T}(\tau)$ and $\mathbf{1} = \boldsymbol{T}(\tau)\mathbf{1}$. The first eigenvalue of $\boldsymbol{T}(\tau)$ always equals to 1 and its corresponding eigenvector corresponds to the equilibrium state populations. The lag time (τ) must be

long enough to allow the dynamics in the reduced state space to become Markovian or memoryless, otherwise the memory of these transitions needs to be considered.

2.3. Generalized Master Equation (GME) theory

A major challenge in constructing MSMs for protein dynamics is building a Markovian model of state dynamics. To achieve a Markovian model, the lag time must be long enough to allow the fully relaxation of memory effects in dynamic transitions among states. As the lag time is bound by the length of individual short MD simulations, a large number of states are often needed for MSMs to achieve Markovian behavior. However, MSMs containing hundreds and even thousands of states can impede the comprehension of biological mechanisms.

Recently developed non-Markovian dynamic models aim to address the aforementioned challenge in MSM. These models go beyond the Markovian assumption for interstate transitions and utilize the GME framework to explicitly account for the memory of protein dynamics. The GME is derived from the Liouville's equation Eq. (1). In the Liouville's equation, the dynamics in the high-dimensional phase space $\boldsymbol{\Gamma}$ is Markovian. In reduced dimensionality (e.g., the collective variable (CV) space discussed in Sec. **3.3** or the state space discussed in Sec. **3.4**-**3.5)**, the dynamics are projections of the phase-space dynamics given by Eq. (1) onto these reduced dimensions. For a state model, the projection from phase space to the state space satisfies the following generalized Hummer-Szabo projection operator:

$$\mathbb{P} = \sum_i \left|\chi_i(\boldsymbol{x})\rho_{eq}(\boldsymbol{x})\right\rangle \pi_i^{-1} \langle \chi_i(\boldsymbol{x})| \tag{6}$$

Here χ is an indicator function: $\chi_i(\boldsymbol{x}) = 1$ when $\boldsymbol{x} \in X_i$ and $\chi_i(\boldsymbol{x}) = 0$ otherwise. $\rho_{eq}(\boldsymbol{x})$ represents the equilibrium probability density, and π_i represents the equilibrium population of the state i.

Upon the projection, the dynamics follow the Nakajima-Zwanzig equation:

$$\frac{\partial}{\partial t}\mathbb{P}\rho(t) = \mathbb{P}\mathcal{L}\mathbb{P}\rho(t) + \mathbb{P}\mathcal{L}e^{\mathbb{Q}\mathcal{L}t}\mathbb{Q}\rho(0) + \int_0^t \mathbb{P}\mathcal{L}\, e^{\mathbb{Q}\mathcal{L}(t-s)}\mathbb{Q}\mathcal{L}\mathbb{P}\rho(s)\mathrm{d}s \tag{7}$$

With $\mathbb{Q} = \mathbf{I} - \mathbb{P}$ ($\mathbf{I}$ is an identity matrix). In Eq. (7), the second term on the right-hand side vanishes when $\rho(0)$ is initiated from an equilibrium distribution. Thus, the Nakajima-Zwanzig equation can be rewritten as the following General Master Equation (GME)[50, 51]:

$$\dot{\boldsymbol{T}}(t) = \boldsymbol{T}(t)\dot{\boldsymbol{T}}(0) - \int_0^t \boldsymbol{T}(t-\tau)\boldsymbol{K}(\tau)d\tau \tag{8}$$

Here, $\boldsymbol{T}(t)$ is the row-normalized transition probability matrix (TPM) with lag time t, following the same convention in **Sec. 2.2**. Each element of the TPM ($T_{ij}(t) = \left\langle \chi_j(\boldsymbol{x})\middle| e^{\mathcal{L}t}\middle|\chi_i(\boldsymbol{x})\rho_{eq}(\boldsymbol{x})\right\rangle \pi_i^{-1}$) corresponds to the conditional probability for the system visit state $\boldsymbol{X_j}$ after a lag time of t, given its initial state at $\boldsymbol{X_i}$. While $\boldsymbol{K}(t)$ with each element $K_{ij}(t) = -\left\langle \chi_j(\boldsymbol{x})\middle|\mathcal{L}e^{\mathbb{Q}\mathcal{L}t}\mathbb{Q}\mathcal{L}\middle|\chi_i(\boldsymbol{x})\rho_{eq}(\boldsymbol{x})\right\rangle \pi_i^{-1}$ is referred as the memory kernel matrix.

An important feature of the memory kernel is the time scale τ_K, namely, the memory kernel relaxation time, when the memory kernel $\boldsymbol{K}(\tau)$ decays to zero: $\boldsymbol{K}(t \geq \tau_K) = \mathbf{0}$. In biomolecular systems, where the separation of timescales often occurs, we have demonstrated that memory kernel relaxation time (τ_K) is often significantly shorter than the Markovian lag time (τ_M)[50]. Under this condition, the GME can be rewritten as

$$\dot{\boldsymbol{T}}(t) = \boldsymbol{T}(t)\dot{\boldsymbol{T}}(0) - \int_0^{\min[\tau_K,t]} \boldsymbol{T}(t-s)\boldsymbol{K}(s)ds \tag{9}$$

where the convolution term of the right-hand side only needs to be computed up to τ_K when predicting dynamics at a long lag time ($t \geq \tau_K$). This provides us with the opportunity to build GME with short MD trajectories. In the GME-based method, we can use the short MD trajectories to compute the time-dependent TPMs, $\boldsymbol{T}(t)$. These short-time $\boldsymbol{T}(t)$ can be employed to compute the memory kernels $\boldsymbol{K}(s)$ with Eq. (9). Subsequently, the long-time dynamics $\boldsymbol{T}(t)$ at any lag time longer than τ_K can be computed from the GME, as given by Eq. (9). In **Sec. 3.6**, we demonstrate a brute-force method utilizing Eq. (9) to construct GME, namely the quasi-MSM (qMSM). qMSM is theoretically rigorous, but it also involves the computation of the time-dependent memory kernel tensor $\boldsymbol{K}(s)$, which is challenging due to the numerical instability induced by the fluctuations of MD simulations especially for complex systems. In **Sec. 2.4** below, we will introduce the Integrative-GME (or IGME) to improve the numerical instability of qMSM by analytical solving the GME at $t \geq \tau_K$.

2.4. Integrative GME (IGME) theory

To enhance numerical stability, the IGME theory[51] adopts the time integration of memory kernels $\boldsymbol{M}_n(t)$ instead of the memory kernel tensor $\boldsymbol{K}(t)$. When $t \geq \tau_K$, the GME can be rewritten as an ordinary differential equation after applying the Taylor series of $\boldsymbol{T}(t-s)$ in the convolution term of Eq. (7):

$$\begin{gathered}\boldsymbol{T}(t)^{-1}\frac{d}{dt}\boldsymbol{T}(t) = \dot{\boldsymbol{T}}(0) - \boldsymbol{M}_0 - \sum_{n=1}^{\infty}\left[\frac{(-1)^n}{n!}\boldsymbol{T}(t)^{-1}\frac{d^n}{dt^n}\boldsymbol{T}(t)\right]\boldsymbol{M}_n \\ \boldsymbol{M}_n = \int_0^{\tau_K}\boldsymbol{K}(s)s^n ds\end{gathered} \tag{10}$$

Here $\boldsymbol{M}_n$ are the time integrals of memory kernels at order n. At the zeroth order, $\boldsymbol{M}_0$ corresponds to the time-integrated memory kernel matrix. Eq. (10) contains $\boldsymbol{M}_n$ instead of the memory kernel tensor $\boldsymbol{K}(s)$, which avoids the numerical computation of $\boldsymbol{K}(s)$. In the IGME theory, we have obtained the analytical solution to the above ordinary differential equation for $T(t)$ as:

$$\begin{gathered}\boldsymbol{T}(t \geq \tau_K) = \boldsymbol{A}\widehat{\boldsymbol{T}}^t \\ \ln\widehat{\boldsymbol{T}} = \dot{\boldsymbol{T}}(0) - \boldsymbol{M}_0 - \sum_{n=1}\frac{(-1)^n}{n!}\left(\ln\widehat{\boldsymbol{T}}\right)^n\boldsymbol{M}_n\end{gathered} \tag{11}$$

Here $\widehat{\boldsymbol{T}}$ and $\boldsymbol{A}$ are two constant matrices. $\widehat{\boldsymbol{T}}$ describes the dynamics at an infinitely long lag time, which can be used to estimate the timescales of the slowest dynamical modes and transition rates between pairs of states. $\widehat{\boldsymbol{T}}$ is obtained by fitting $\boldsymbol{A}\widehat{\boldsymbol{T}}^t$ with short-time MD simulation trajectories $\boldsymbol{T}(t)$ (see **Sec. 3.6** and **Sec. A1** for details).

The second equation of Eq. (11) can also be utilized to compute the time-integrated memory kernel matrix $\boldsymbol{M}_0$. According to Eq. (11), the rigorous computation of $\boldsymbol{M}_0$ involves $\boldsymbol{M}_n$ at higher orders. However, for biomolecular systems where there are separations of timescales (i.e., $-t/\ln\lambda_i(t) \gg \Delta t$, $\lambda(t)$ are eigenvalues of $\boldsymbol{T}(t)$, and Δt is the saving interval of the input $\boldsymbol{T}(t)$ in IGME[51]), we have shown that:

$$\boldsymbol{M}_0 \approx \dot{\boldsymbol{T}}(0) - \ln\widehat{\boldsymbol{T}} \tag{12}$$

Therefore, $\boldsymbol{M}_0$ can be obtained from the IGME.

3. Protocol for constructing GME Models to study protein dynamics

In this section, we will introduce a detailed protocol (**Figure 1**) to construct qMSM and IGME models to study protein dynamics based on MD simulation trajectories.

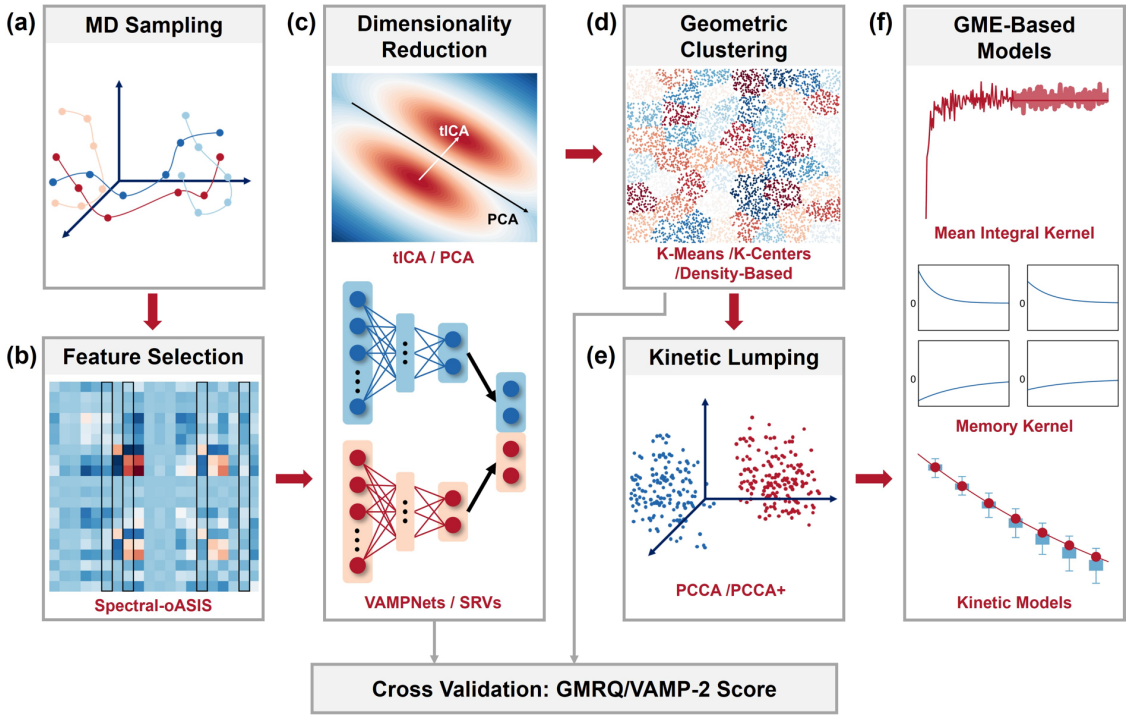

Figure 1. Our recommended protocol for constructing GME Models to study protein dynamics.

3.1 Feature selection

The input to our protocol is an MD simulation dataset containing an ensemble of MD trajectories that sample protein conformational changes of interest (**Figure 1(a)**). An MD trajectory consists of the time evolution of MD snapshots, each containing the positions of all the atoms in the simulation box. However, these Cartesian coordinates are often unsuitable for the analysis of protein conformational changes because they are typically of high dimensions (e.g., a typical protein system contains tens of thousands of Cartesian coordinates). Additionally, it is challenging to separate protein internal motions from their external motions. In contrast, internal coordinates (e.g., inter-residue distances or backbone torsional angles) offer a better description of protein conformational changes and are often used as input features to construct MSM or GME models.

Many biologically relevant conformational changes are localized, involving structural transitions in only a subset of the system, such as translocation of a motor protein on dsDNA[61], and the gate opening of RNA polymerase[7]. Even for global dynamic processes like protein folding, one could identify a subset of structural features sufficient to describe the slowest dynamics of these conformational changes. Therefore, the first step in our protocol is to select a subset of structural features that can describe the slowest dynamics of the system (**Figure 1(b)**). In this section, we will discuss several algorithms for automatic feature selection, including Accelerated Sequential Incoherent Selection (oASIS)[62], spectral oASIS[63], the Force Distribution Analysis (FDA)[64], and molecular systems automated identification of correlation (MoSAIC)[65].

Both the oASIS[62] and spectral oASIS[63] aim to find a subset of features to reconstruct the original

feature space based on the Nyström method. The primary focus of these two methods is to minimize the errors of diagonal elements between the original correlation matrix $C = X^{\mathrm{T}}X$ (X_{ij} is the value of feature j at simulation frame i) and reconstructed correlation matrix $\tilde{C} = C_k W_k^{\dagger} C_k^{\mathrm{T}}$ (C_k is subset columns of C, and W_k is the correlation matrix of subset features) reconstructed based on the Nyström method. The oASIS method employs a selection strategy that targets the column indexed with the highest diagonal error, which is less effective when selecting more than one column in each iteration. The Spectral Oasis, a modified version of oASIS, maintains the effectiveness of batch selection by considering both reconstruction errors and eigenvector differences. The FDA method involves the analysis of pair-wise forces and the mechanical strain distribution alongside the MD simulations. This method focuses on residue pairs that exhibit significant force variations in the simulations. Finally, MoSAIC[65] is a correlation-based feature selection method based on the physical insights that crucial dynamic processes involve many features changing in a concerted manner. MoSAIC makes use of the Leiden community detection algorithm to block-diagonalize the correlation matrix, thereby creating coherent and distinct feature clusters. These clusters are subsequently ranked according to their size. Larger clusters are presumed to be linked to significant dynamic processes, while smaller clusters are eliminated as noise during the feature selection process. In this tutorial, we choose the Spectral oASIS[63] as the feature selection method.

3.2. Dimensionality reduction

The feature selection in the previous step provides a representative subset of internal coordinates, but their number is large and often at the order of hundreds to thousands of features. In our protocol, we will further perform a dimensionality reduction based on these features to identify several collective variables (CVs) that describe the slowest dynamics of the system (**Figure 1(c)**). In this step, various dimensionality reduction algorithms can be utilized, e.g., Principal Component Analysis (PCA)[66], time-lagged Independent Components Analysis (tICA)[67-69], Variational Approach for Markovian Process (VAMP) based neural networks (VAMPnets)[70] or its combination with graph neural networks (GraphVAMPnets)[71, 72], and State-free Reversible VAMPnets (SRVs)[73]. In this tutorial, both tICA and SRVs will be applied.

PCA[66] can find a small number of principal components by maximizing the variance across the spatial scale of the principal components. Alternatively, tICA[67-69] can find a set of collective variables representing the slowest dynamics of biomolecules by maximizing the time-lagged autocorrelation of the transformed components. In tICA, the self-correlation matrix $\boldsymbol{C}_{00}$ and time-lagged autocorrelation matrix $\boldsymbol{C}_{01}$ are constructed from the high-dimensional feature space $\boldsymbol{x}(t) = [x_1(t), \dots, x_d(t)]^{\mathrm{T}}$, where $x_1(t), \dots x_d(t)$ are d features at simulation time t:

$$\begin{aligned} \boldsymbol{C}_{00} &= \mathbb{E}_t[\boldsymbol{x}(t)\boldsymbol{x}(t)^{\mathrm{T}}] \\ \boldsymbol{C}_{01} &= \mathbb{E}_t[\boldsymbol{x}(t)\boldsymbol{x}(t+\tau)^{\mathrm{T}}] \end{aligned} \tag{13}$$

Here τ is the chosen lag time and the d features can be picked by intuition or generated from feature selection methods (e.g., spectral oASIS[63]). The CVs corresponding to the slowest dynamic modes can then be obtained by solving the generalized eigenvalue problem:

$$\boldsymbol{C}_{01}\boldsymbol{U} = \boldsymbol{C}_{00}\boldsymbol{U}\boldsymbol{\Lambda} \tag{14}$$

These slowest CVs (called TICs) can then be constructed from dimension reduction of the input features that utilizes a sub-matrix of $\boldsymbol{U}$ consisting of the top columns corresponding to the largest values in the diagonal matrix $\boldsymbol{\Lambda}$**.** Both PCA and tICA can generate uncorrelated collective variables by using linear combinations of input features. PCA tends to assign importance to large-amplitude

motions, even if they are typically irrelevant to the actual function of a protein. A similar challenge can be encountered when using tICA, especially when the coordinates involve slow but unimportant motions. For example, when tICA is applied to dihedral angles (ϕ, ψ) to investigate the folding of a helical protein, HP35, it identifies transitions between right- and left-handed helices as the slowest processes[65, 74, 75].

The VAMPnets[70] was developed based on the variational approach of Markov processes (VAMP) theorem. VAMPnets allows for the independent training of two parallel encoders (E_0 and E_1) to find the low-dimensional latent space representation of the collective variables: $\boldsymbol{y}_0(t) = E_0(\boldsymbol{x}(t))$ and $\boldsymbol{y}_1(t+\tau) = E_1(\boldsymbol{x}(t+\tau))$. Minimizing the loss function in VAMPNets is equivalent to maximize the VAMP-2 score that is defined as:

$$\begin{aligned} \boldsymbol{C}_{00} &= \mathbb{E}_t[\boldsymbol{y}_0(t)\boldsymbol{y}_0(t)^{\mathrm{T}}] \\ \boldsymbol{C}_{01} &= \mathbb{E}_t[\boldsymbol{y}_0(t)\boldsymbol{y}_1(t+\tau)^{\mathrm{T}}] \\ \boldsymbol{C}_{11} &= \mathbb{E}_t[\boldsymbol{y}_1(t+\tau)\boldsymbol{y}_1(t+\tau)^{\mathrm{T}}] \\ R_{\mathrm{VAMP2}} &= \left\| \boldsymbol{C}_{00}^{-1/2}\, \boldsymbol{C}_{01} \boldsymbol{C}_{11}^{-1/2} \right\|_F^2 \end{aligned} \tag{15}$$

Where the subscript *F* represents the Frobenius norm. The VAMPnets can work with the input of molecular coordinates, and finally yield a state model that has better performance than previous Markov modeling approaches.[70] State-free Reversible VAMPnets[76] (SRVs) method is largely similar to VAMPnets, while the major difference lies in the reversible-dynamics assumption of the SRVs. Unlike the VAMPnets that adopts two independent neural networks to train $\boldsymbol{y}_0(t)$ and $\boldsymbol{y}_1(t+\tau)$, the SRVs uses the shared neural network for $\boldsymbol{y}(t) = E(\boldsymbol{x}(t))$ and $\boldsymbol{y}(t+\tau) = E(\boldsymbol{x}(t+\tau))$. In the training, the SRVs utilize a slightly different loss function (VAMP-2 like loss function):

$$\begin{aligned} \boldsymbol{C}_{01}\boldsymbol{s}_i &= \tilde{\lambda}_i \boldsymbol{C}_{00}\boldsymbol{s}_i \\ L &= -\sum_i \left(\tilde{\lambda}_i\right)^2 \end{aligned} \tag{16}$$

Where $\boldsymbol{C}_{00}$ and $\boldsymbol{C}_{01}$ are computed in the same way as Eq. (15) and $\tilde{\lambda}_i$ are the generalized eigenvalues. The SRVs can achieve higher success rate for training in numerical experiments[76]. In addition, the SRVs only yields the collective variables, unlike the VAMPnets that yield few-state kinetic models.

Besides the above-mentioned methods, many other algorithms are also available for dimension reduction, such as kernel-tICA[77], deep-tICA[78], time-lagged autoencoder[79], variational dynamics encoder[80], past-future information bottleneck[81], state predictive information bottleneck[82], transition manifold methods[83], reaction coordinate flow[84], and relaxation mode analysis[85], etc.

3.3. Geometric clustering to generate microstates

Geometric clustering (**Figure 1(d)**) involves partitioning the reduced-dimensional conformational space spanned by the CVs into a large number of discrete clusters (called microstates). The most widely used clustering methods in the MSM/GME construction are the centroid-based algorithms[86] such as K-Means[87], K-Centers[88, 89] and K-Medoids[90]. In K-Means clustering, the primary objective is to minimize the sum of squared distances between data points and the centroid of the cluster that the data point belongs to, which is calculated as the mean of the data points assigned to that cluster. K-Centers clustering aims to minimize the maximum distance or dissimilarity between a data point and the nearest cluster center. The cluster centers are evenly distributed. K-Medoids clustering also seeks to minimize the sum of distances like K-Means, but it uses actual data points (medoids) as representatives of the

clusters. In addition to the centroid-based algorithms, other clustering methods are also available, such as Automatic state partitioning for multibody systems (APM)[91], shape-Gaussian mixture models (shape-GMM)[92], Density-Based Spatial Clustering of Applications with Noise (DBSCAN), and Adaptive partitioning by local density-peaks (APLoD)[93].

3.4. Kinetic Lumping to produce metastable macrostates

In this step, we will further lump microstates that can interconvert quickly into a small set of metastable microstates (**Figure 1(e)**). The most widely used kinetic lumping methods[94-98] are the Perron-Cluster Cluster Analysis (PCCA)[94] and Robust Perron Cluster Analysis (PCCA+)[95, 96]. The PCCA uses the structure of the eigenvectors of microstate TPMs to find the metastable states. In PCCA, the sign pattern of the eigenvectors corresponding to the largest eigenvalues of TPM is used for finding the metastable states that can form a nearly uncoupled Markov chain. Unlike the PCCA that uses a crisp assignment of state boundary, the PCCA+[96] utilizes almost invariant sets by almost characteristic functions $\tilde{\chi}$, which is a soft assignment for the microstates. $\tilde{\chi}$ is obtained by maximizing the metastability of microstate TPM: $\mathrm{tr}(\widetilde{W}) = \sum_i \widetilde{w}_{ii}$, where $\widetilde{W} = (\mathrm{diag}[\pi])^{-2}\langle\tilde{\chi}|\boldsymbol{T}^{\mathrm{micro}}|\tilde{\chi}\rangle_\pi$ (π are the stationary populations, and T^{micro} is the microstate TPM). In practice, the PCCA+ employs perturbation of characteristic functions to optimize the invariant sets $\tilde{\chi}$. In our tutorial, we utilize the PCCA+ algorithm for kinetic lumping.

3.5. Constructing Markov State Models

With a set of micro- and macro-states, we can then estimate transition probabilities between pairs of states after the lag time τ from MD simulations:

$$T_{ij}(\tau) = p[x(t+\tau) \in j \mid x(t) \in i] = \frac{C_{ij}(\tau)}{\sum_j C_{ij}(\tau)} \tag{17}$$

Where C is the transition count matrix (TCM), and $C_{ij}(\tau)$ corresponds to the number of transitions that begin from state i and end at state j after the lag time τ. For equilibrium sampling, the TCM should be theoretically symmetric to satisfy the detailed balance: $C_{ij}(\tau) = C_{ji}(\tau)$. However, for realistic applications, one often needs to symmetrize the TCM using the following equation:

$$\boldsymbol{C}^{\mathrm{sym}}(\tau) = \frac{\boldsymbol{C}(\tau) + \boldsymbol{C}(\tau)^T}{2} \tag{18}$$

Alternatively, when there are large differences between $C_{ij}(\tau)$ and $C_{ji}(\tau)$, the Maximum likelihood estimator (MLE)[99] can be employed to enforce the detailed balance condition using the following likelihood function[9]:

$$p(T|C^{obs}) \propto \prod_{i,j=1}^{n} T_{ij}^{C_{ij}^{\mathrm{prior}}+C_{ij}^{\mathrm{obs}}} = \prod_{i,j=1}^{n} T_{ij}^{C_{ij}} \tag{19}$$

The resulting MLE algorithm can be written as[9, 99]:

$$\pi_i = \sum_j \frac{c_{ij} + c_{ji}}{\frac{N_i}{\pi_i} + \frac{N_j}{\pi_j}}, \qquad T_{ij} = \frac{(c_{ij} + c_{ji})\pi_j}{N_j\pi_i + N_i\pi_j} \tag{20}$$

Where N_i represents the total number of transition counts starting from state i. In an MSM, the eigenvectors and eigenvalues of TPMs ($T(\tau)$) represent the slowest dynamic modes. Starting from the second eigenmode, the $(i+1)$-th eigenvector corresponds to the i-th slowest dynamic mode, while the corresponding eigenvalue λ_{i+1} describes the fraction of molecules that have not undergone the

transition after the lag time τ. The first eigenvalue is always 1, and the first eigenvector corresponds to the stationary populations of states.

In various steps of the MSM construction (**Figure 1(c,d)**), we need to determine several hyperparameters, including the optimal number of CVs, lag time in tICA or SRVs, and number of microstates. To choose the optimal values for these hyperparameters, we recommend using the generalized matrix Rayleigh quotient (GMRQ)[100] in our protocol. The GMRQ employed cross-validation to avoid possible overfitting induced by the violation of variational bounds due to statistical uncertainty[100]. In GMRQ, the whole dataset is divided into a training set and a test set, and the Rayleigh quotient is computed from the eigenvectors and correlation matrices of these two parts:

$$R = \mathrm{Tr}(\boldsymbol{V}^{\mathrm{T}}\boldsymbol{C}\boldsymbol{V}(\boldsymbol{V}^{\mathrm{T}}\boldsymbol{S}\boldsymbol{V})^{-1}) \tag{21}$$

Where $\boldsymbol{V}$ is the right eigenvector of TPM computed from the training set data, $\boldsymbol{S}$ and $\boldsymbol{C}$ are the diagonal matrix of stationary population and TCM computed from the test set data, respectively. In practice, the best model should have the highest GMRQ score.

3.6. Constructing qMSM and IGME models

Using TPMs and their time derivatives at different lag times as input, one could also construct qMSM, a non-Markovian GME-based dynamic model (**Figure 1(f)**). Given a series of short-time $\boldsymbol{T}(t)$, qMSM employs a brute force approach to numerically compute the memory kernel tensor $\boldsymbol{K}(t)$ and predict the long-time dynamics based on Eq. (9).

In qMSM, a discrete-time GME at $t = n\Delta t$ is employed as follows:

$$\dot{\boldsymbol{T}}(n\Delta t) = \boldsymbol{T}(n\Delta t)\dot{\boldsymbol{T}}(0) - \Delta t \sum_{m=1}^{\min[n,\, \tau_K/\Delta t]} \boldsymbol{T}\big((n-m)\Delta t\big)\boldsymbol{K}(m\Delta t) \tag{22}$$

Where τ_K is the memory relaxation time when the memory kernel $\boldsymbol{K}(\tau)$ decays to zero: $\boldsymbol{K}(t \geq \tau_K) = \boldsymbol{0}$). Therefore, a straightforward method to compute memory kernel can be derived from the above time-discrete GME:

$$\boldsymbol{K}(n\Delta t) = -\frac{\dot{\boldsymbol{T}}(n\Delta t) - \boldsymbol{T}(n\Delta t)\dot{\boldsymbol{T}}(0)}{\Delta t} + \sum_{m=1}^{n} \boldsymbol{T}\big((n-m)\Delta t\big)\boldsymbol{K}(m\Delta t) \qquad (n\Delta t \leq \tau_K) \tag{23}$$

To find τ_K, the qMSM employs the mean integral kernel (MIK) to visualize the relaxation of memory kernel tensor:

$$\mathrm{MIK}(t) = \frac{1}{N}\sqrt{\sum_{i,j=1}^{N}\left(\int_0^t K_{ij}(\tau)\mathrm{d}\,\tau\right)^2} \tag{24}$$

When $\boldsymbol{K}(t)$ fully relaxes, the MIK will become independent of time. Therefore, the MIK can act as an indicator for τ_K in qMSM. Finally, the long-time dynamics can be predicted from Eq. (22) with the memory kernels computed from Eq. (23) and τ_K obtained from Eq. (24). In the applications of qMSM, we observe that the fluctuations encountered when obtaining $\boldsymbol{T}(t)$ and $\dot{\boldsymbol{T}}(t)$ from MD trajectories can induce numerical instability in $\boldsymbol{K}(t)$, especially for complex systems. To address this challenge, we have recently developed the IGME method[51].

The IGME method utilizes $\boldsymbol{T}(t)$ at $t \geq \tau_K$ to compute the two matrices $\boldsymbol{A}$ and $\widehat{\boldsymbol{T}}$ in Eq. (11). In this tutorial, we introduce a least-square fitting (LSF) method to compute $\boldsymbol{A}$ and $\widehat{\boldsymbol{T}}$ with the row-sum restriction of $\boldsymbol{T}(t)$: $\sum_j T_{ij}^{\mathrm{MD}}(t) \equiv 1$. The least-square fitting method in this tutorial adopts the following Lagrangian (see **Sec. A1** for details):

$$L = \frac{1}{2}\sum_t \left|\ln \boldsymbol{T}^{\mathrm{MD}}(t) - \ln \boldsymbol{A} - t \ln \widehat{\boldsymbol{T}}\right|_F^2 + \sum_i \left(\gamma_i \sum_j \left[\ln \widehat{\boldsymbol{T}}\right]_{ij}\right) \tag{25}$$

Where γ_i is the Lagrange multiplier to guarantee the row-sum rule of $\boldsymbol{T}^{\mathrm{MD}}(t)$. With the Lagrange method of the above Lagrangian, we can derive a straightforward least-square fitting method to fit $\boldsymbol{A}$ and $\widehat{\boldsymbol{T}}$ (see Eq. (A6) for details).

In practice, the LSF fitting is performed on a subset of the input TPMs: $\left[T^{\mathrm{MD}}\left(\tau_K^{\mathrm{trial}}\right), T^{\mathrm{MD}}\left(\tau_K^{\mathrm{trial}} + \Delta t\right), \ldots, T^{\mathrm{MD}}\left(\tau_K^{\mathrm{trial}} + L\right)\right]$, where τ_K^{trial} is the time of the first frame used in LSF, and L is the length of input data used in LSF. For each fitting, we can compute the resulting $\boldsymbol{A} = \boldsymbol{A}\left(\tau_K^{\mathrm{trial}}, L\right)$ and $\widehat{\boldsymbol{T}} = \widehat{\boldsymbol{T}}\left(\tau_K^{\mathrm{trial}}, L\right)$. A single run of LSF fitting may be susceptible to numerical fluctuations of simulation data. Therefore, we adopted a systematic search for τ_K^{trial} and L to obtain the best IGME models that match the MD simulation data. To quantify the errors of IGME models in reproducing the simulation data, we have followed our previous work to utilize the time-averaged root mean squared error (RMSE)[51]:

$$\mathrm{RMSE} = \sqrt{\frac{\sum_{n=1}^{L_x}\sum_{i,j=1}^{N}\left[\pi_i \boldsymbol{T}_{ij}^{\mathrm{MD}} - \pi_i \boldsymbol{T}_{ij}^{IGME}(t)\right]^2 dt}{N^2 L_x}} \tag{26}$$

Where $\boldsymbol{T}^{IGME}$ corresponds to the TPMs predicted by IGME, represented as $\boldsymbol{T}^{IGME} = \boldsymbol{A}\widehat{\boldsymbol{T}}$. Additionally, L_x denotes the maximum lag time of $\boldsymbol{T}^{IGME}(t_x)$ used for computing the RMSE ($t_x = L_x \Delta t$, where Δt is the saving interval of MD simulations). By substituting $\boldsymbol{T}^{IGME}$ with $\boldsymbol{T}^{qMSM}$ or $\boldsymbol{T}^{MSM}$ in Eq. (26) the same RMSE metric can be employed to evaluate the accuracy of qMSM or MSM[50], respectively. Specifically, for qMSMs, $L_x = t_x/\Delta t$, while for MSMs, $L_x = t_x/\tau_M^{trial}$ (where τ_M^{trial} represents the lag time of an MSM).

In our implementation of IGME, we perform the systematic search for possible values of two hyperparameters, τ_K^{trial} and L, $\tau_K^{\mathrm{trial}} = \Delta t, 2\Delta t, \ldots, L_0$, and $L = \Delta t, 2\Delta t, \ldots, L_0 - \tau_K^{\mathrm{trial}}$ (L_0 is a pre-determined scanning range). Finally, the IGME models with the smallest RMSE will be used.

4. Tutorial examples

In this section, we will provide a detailed tutorial on how to construct non-Markovian dynamic models (i.e., qMSM and IGME) from MD simulation datasets using two examples: the alanine dipeptide and villin headpiece. To perform the tasks in this tutorial, we will utilize both MSMBuilder[101, 102] and PyEMMA[103] software. All the relevant Python codes for this tutorial are available on GitHub: https://github.com/xuhuihuang/GME_tutorials. This tutorial employs MSMBuilder version 2022 (https://github.com/msmbuilder/msmbuilder2022) and the PyEMMA version 2.5.12 (https://github.com/markovmodel/PyEMMA/).

4.1. Alanine Dipeptide

The first example is the conformational dynamics of the alanine dipeptide in explicit solvent (**Figure**

2(a)). The MD simulation dataset of alanine dipeptide consists of 100 MD trajectories, and each trajectory is performed for 10 ns under the NVT ensemble at 310 K. AMBER99SB force field[104] is used for alanine dipeptide and the TIP3P model[105] for water. The snapshots of these MD trajectories are stored every 0.1 ps.

As shown in **Scheme 1**, we utilize all 45 pair-wise distances among 10 heavy atoms of the alanine dipeptide as the input features in the featurization step. Given the relatively small size of this system, there is no need for feature selection.

Scheme 1 | Featurization of alanine dipeptide

```
from msmbuilder.featurizer import AtomPairsFeaturizer
from msmbuilder.dataset import dataset
import numpy as np
xyz = dataset(MDtrajs_dir + "*.xtc",
              topology = MDtrajs_dir + "ala2.pdb", stride=1)
atom_pair_list = np.loadtxt(MDtrajs_dir + 'heavy_atom_pairs_list.txt')
featurizer = AtomPairsFeaturizer(atom_pair_list)
ftrajs = featurizer.fit_transform(xyz)
```

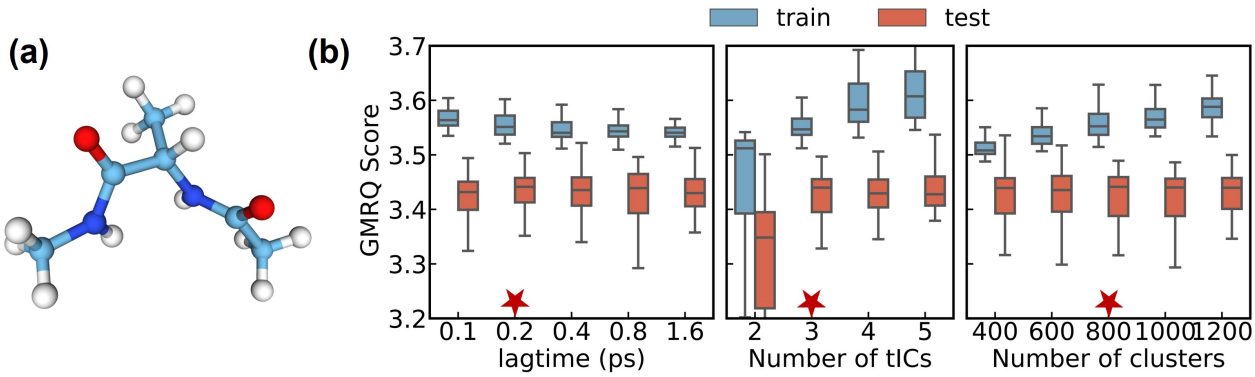

Figure 2. Featurization and cross-validation of microstate tICA-MSM for the alanine dipeptide. (a) Structure of alanine dipeptide. The atom distances between the 10 heavy atoms are selected for features. **(b)** Cross-validation based on GMRQ score to select the optimal hyperparameters, including the lag time of tICA, number of tICs and number of microstate clusters for K-Centers clustering. Based on the GMRQ scores, we chose a 0.2 ps lag time, 3 tICs, and 800 microstates.

With the input features, the dimensionality reduction was performed with tICA. Specifically, the original 45 distance features are transformed into a reduced set of time-lagged Independent Components (tICs). In our tutorial, the tICA model is built with 3 tICs at the tICA lag time 0.2 ps. A sample Python code for performing the tICA analysis is shown in **Scheme 2**. Next, we apply the K-Centers algorithm to generate a microstate model with 800 states based on the top three tICs (see **Scheme 3**).

Scheme 2 | Dimensionality Reduction using tICA

```
from msmbuilder.decomposition import tICA
tica = tICA(n_components=3, lag_time=2, kinetic_mapping=True)
tica_trajs = tica.fit_transform(ftrajs)
```

Scheme 3 | Geometric Clustering

```
from msmbuilder.cluster import KCenters
cluster = KCenters(n_clusters=800)
clustered_trajs = cluster.fit_transform(tica_trajs)
```

In the dimensionality reduction (**Figure 1(c)**) and geometric clustering step (**Figure 1(d)**), we have applied the cross-validation tool, GMRQ, to select the optimal values for several hyperparameters, including the lag time of tICA (τ_{tICA}), the number of tICs (n_{tICs}), and the number of geometric clusters ($n_{\mathrm{microstates}}$). In the GMRQ analysis, we perform a 5-fold cross-validation (i.e., 100 MD trajectories are randomly divided into 5 folds, and we use 4 folds for training and the remaining one for validation) and repeat it 10 times by changing the random number when splitting the dataset (see **Scheme 4** for the Python code). Based on these runs, we report GMRQ scores as box plots (**Figure 2(b)**). We then determine the optimal values (those with the highest median GMRQ score in the box plots) of the following three hyperparameters sequentially: $\tau_{\mathrm{tICA}} = 0.2$ ps, $n_{\mathrm{tICs}} = 3, n_{\mathrm{microstates}} = 800$.

Scheme 4 | GMRQ Cross-Validation based on the tICA analysis

```
# The function TICA_CV is used for cross-validation.
# More details can be found on the github.

# Cross-validation for lag time
lt_list = [1,2,4,8,16]
Parallel(n_jobs=5)(delayed(TICA_CV)\
                       (lt=i, n_tics=3, n_clusters=800, para='lagtime') for i in lt_list)

# Cross-validation for number of tICs
ntics_list = [2,3,4,5]
Parallel(n_jobs=4)(delayed(TICA_CV)\
                       (lt=2, n_tics=i, n_clusters=800, para='n_tics') for i in ntics_list)

# Cross-validation for number of microstates (clusters)
nc_list = [400, 600, 800, 1000, 1200]
Parallel(n_jobs=5)(delayed(TICA_CV)\
                       (lt=2, n_tics=3, n_clusters=i, para=' n_clusters') for i in nc_list)
```

Next, we construct the microstate MSM and calculated the error bar of both ITS and residence probabilities of the most populated 8 microstates following the code in **Scheme 5**. As shown in **Figure 3(a)**, the slowest three ITS curves reach plateaus at the lag time of 10 ps, indicating that the microstate MSM reaches Markovian at $\tau_M \geq 10$ ps. To further validate the microstate MSM, we perform the Chapman-Kolmogorov test (according to Eq. (5)), and show that an MSM with a lag time of 10 ps can predict dynamics in reasonable agreement with the MD simulations (**Figure 3(b)**).

Scheme 5 | Microstate-MSM Validation

```
# Construct the microstate MSM
msm = MarkovStateModel(n_timescales=10, lag_time=100, ergodic_cutoff='off',
                          reversible_type='transpose', verbose=False)
msm.fit(clustered_trajs)
# Generate the microstate TPM
micro_TPM = msm.transmat_
# Find the most populated 8 microstates
states_idx = np.argsort(msm.populations_)[:-9:-1]
```

```
# Bootstrapping of clustered trajectories for 50 runs
# More details of the function bootstrapMSM and CK_test can be found on the github.
num_runs = 50
lagtime = np.arange(1, 501)
bootstrap_ITS = np.zeros((num_runs, len(lagtime), 10))
bootstrap_RP = np.zeros((num_runs, len(lagtime), 8))
for run in range(num_runs):
    bootstrap_ITS[run], bootstrap_RP[run] = bootstrapMSM(trajs=clustered_trajs, lagtime=lagtime,
                                                         n_timescales=10, n_RP=8,
                                                         RP_idx=states_idx, n_states=800,
                                                         num_samples_per_run=100)
# Calculate the error bar of bootstrapped data
ITS_std = np.std(bootstrap_ITS, axis=0)
RP_std = np.std(bootstrap_RP, axis=0)

# Generate the MSM-predicted TPMs used for CK-test
MSM_time, MSM_TPM = CK_test(lag=100, TPM= micro_TPM, delta_t=0.1, length=500)
```

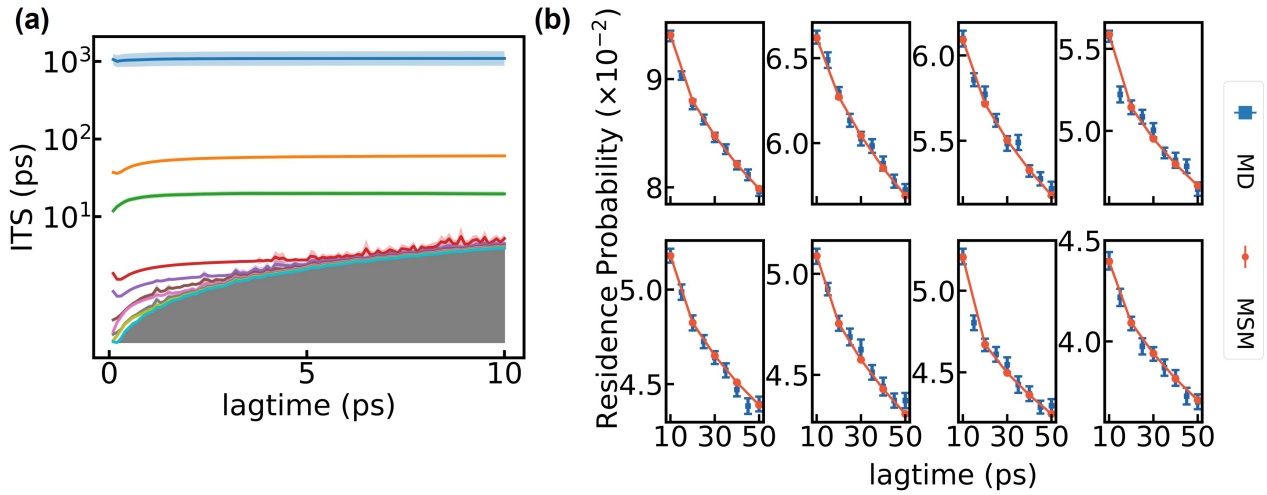

Figure 3. Construction and validation of microstate-MSM of alanine dipeptide. (a) Implied timescales (or ITS) for the first ten dynamic modes as a function of lag time. **(b)** Chapman-Kolmogorov test for the 8 most populated microstates. We choose the lag time to be 10 ps. The error bars in the implied timescale and the residence probability plots are calculated by bootstrapping with replacement 100 trajectories for 50 times.

With a validated microstate-MSM, we next apply PCCA+ to lump 800 microstates into 4 metastable states (see **Figure 1(e)** and the sample code in **Scheme 6**). The number of macrostates is an input parameter for PCCA+, and we determine its values as 4 because there exists a persistent and major gap between the third and fourth eigenmodes in the ITS plots, as shown in **Figure 3(a)**. The four macrostates can be visualized by the projections of their MD snapshots onto two backbone torsion angles of the alanine dipeptide (see **Figure 4(a)**).

Scheme 6 | Kinetic Lumping

```
import numpy as np
from msmbuilder.lumping import PCCAPlus
from msmbuilder.msm import MarkovStateModel
```

```
msm = MarkovStateModel(n_timescales=10, lag_time=5, reversible_type='transpose',
                       verbose=False, ergodic_cutoff='off')
msm.fit(clustered_trajs)
pcca = PCCAPlus.from_msm(msm, n_macrostates=4)
lumped_trajs = pcca.fit_transform(clustered_trajs)

# Use transpose method to get the symmetrized TPM
lagtime = np.arange(1, 501)
TPM = np.zeros((len(lagtime), 4, 4))
for i in range(len(lagtime)):
    msm_macro = MarkovStateModel(n_timescales=3, lag_time=lagtime[i],
                                 ergodic_cutoff='off',
                                 reversible_type='transpose',
                                 verbose=False)
    msm_macro.fit(lumped_trajs)
    TPM[i] = msm_macro.transmat_
```

Based on these four macrostates, we next compute the memory kernels ($\boldsymbol{K}(t)$) and built a qMSM, as illustrated in **Figure 1(f)** and the accompanying sample code in **Scheme 7**. In **Figure 4(c)**, various elements of $\boldsymbol{K}(t)$ exhibit noticeable fluctuations. To determine the memory kernel relaxation time (τ_K) for propagating GME (Eq. (9) and Eq.(22)) we apply the mean integral of memory kernel elements (or MIK, see Eq (24), following our previous work[50]. The MIK plot, as shown in **Figure 4(b)**, reaches a plateau at ~1.5 ps, leading us to select $\tau_K = 1.5$ ps for building the qMSM. The RMSE (Eq. (26)) between our qMSM and the original MD dataset is very small (only 1.5×10^{-3}, see **Figure 5(a)**), and the qMSM model has also successfully passed the Chapman-Kolmogorov test (see blue curves in **Figure 5(b)**). It's worth noting that for any choice of τ greater than $\tau_K = 1.5$ ps, the memory kernel theoretically has fully decayed, and the propagation of dynamics using the GME (Eq. (9)) should yield the same accuracy. However, due to numerical fluctuations, qMSMs constructed at different lag times when τ exceeds 1.5 ps still exhibit varying errors. We will further discuss this issue in the next subsection when constructing IGME models. Finally, using GME in our qMSM, we could obtain TPMs at any lag times. When the lag time is sufficiently long so that the model reaches Markovian, the GME will be reduced to an MSM. Indeed, when $\tau = 10$ ps, the TPM from our qMSM yields the same slowest ITS of 1.14 ns as an MSM for the alanine dipeptide.

Scheme 7 | Constructing qMSMs for alanine dipeptide

```
import sys
sys.path.insert(0, './scripts')
from qmsm import QuasiMSM
delta_t =0.1 # the unit: ps

# Calculate MIK
qmsm = QuasiMSM()
qmsm.fit(TPM, tau_k=55, delta_t=delta_t, rmse=False)
qmsm_mik = qmsm.mik # MIK list with tau_k from 1 to 55

# Build qMSM
qmsm = QuasiMSM()
qmsm.fit(TPM, tau_k=15, delta_t=delta_t)
qmsm_time, qmsm_tpm = qmsm.predict(TPM) # Use qMSM for prediction
qmsm_rmse = qmsm.rmse # RMSE list of the qMSM models with tau_k from 1 to 15
qmsm_its = qmsm.timescales(TPM, ITS_t=100) # ITS predicted by qMSM
```

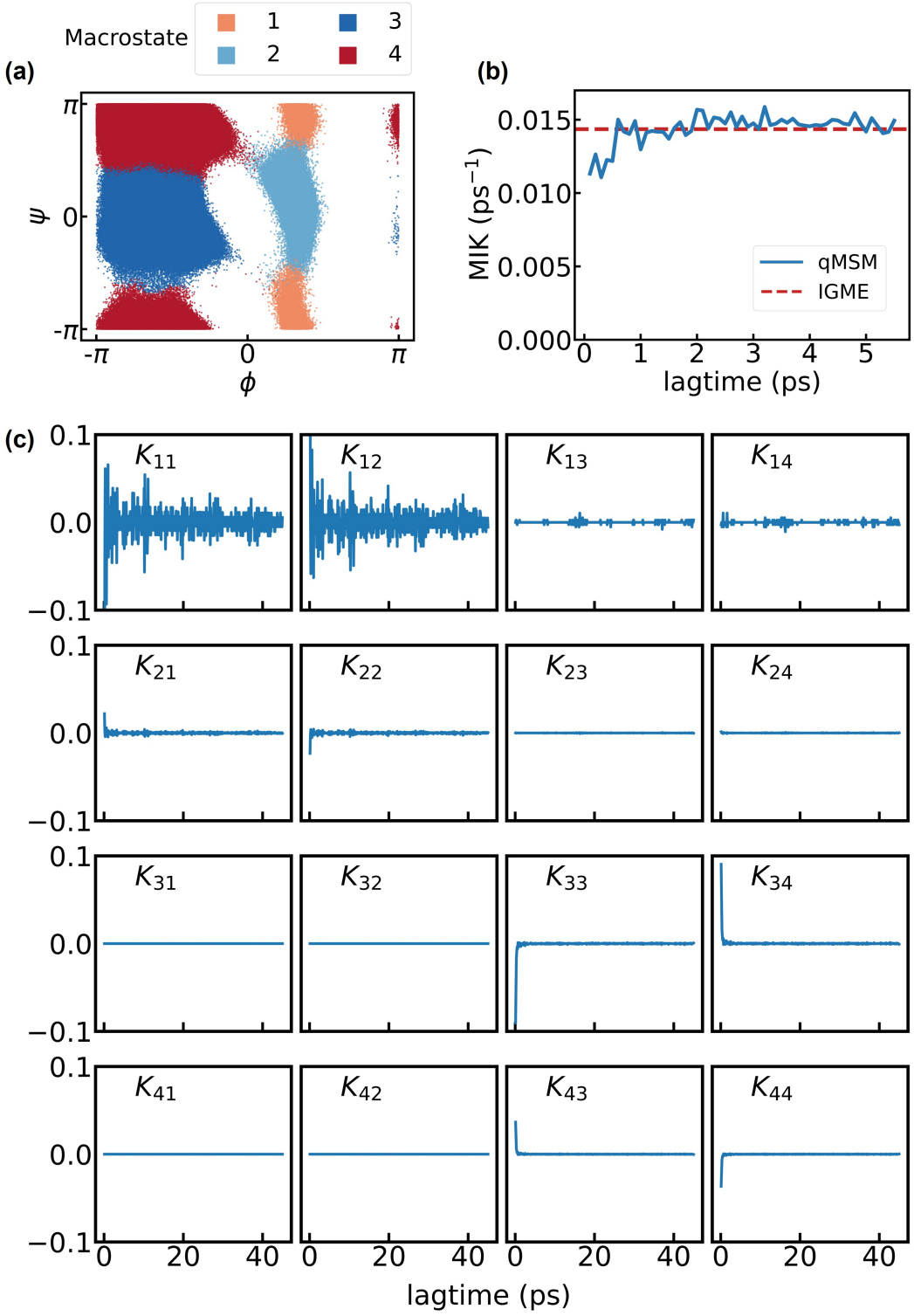

Figure 4. The calculation of memory kernels, $\boldsymbol{K}(t)$. (a) The projections of MD snapshots onto two backbone dihedral angles (ψ, ϕ) of the alanine dipeptide. The four macrostates (1 to 4) are color-coded as orange, cyan, blue, and red, respectively. **(b)** The mean integral kernel (or MIK) calculated from qMSM (blue) and IGME (red) with $\tau_K = 1.5$ ps and $L = 0.1$ ps. **(c)** The full (4×4) memory kernel matrix ($\boldsymbol{K}(t)$) shown as a function of lag time.

Scheme 8 | Constructing IGME models for alanine dipeptide

```
import sys
sys.path.insert(0, './scripts')
from igme import IGMENS, IGME
delta_t =0.1 # the unit: ps

# Scan tau_k and L
igme = IGME()
scan_output = igme.scan(input_data=TPM, begin=1, end=16)

# Build top IGME
igme_top = igme.top_model(scan_output, 1) # Select the top IGME model
igme_tpm = np.array(igme_top.predict(1,len(TPM))) # Use IGME for prediction
igme_top_rmse = igme_top.rmse # RMSE of the top IGME model
igme_mik = igme_top.mik / delta_t
igme_its = igme_top.timescales * delta_t # ITS predicted by top IGME
```

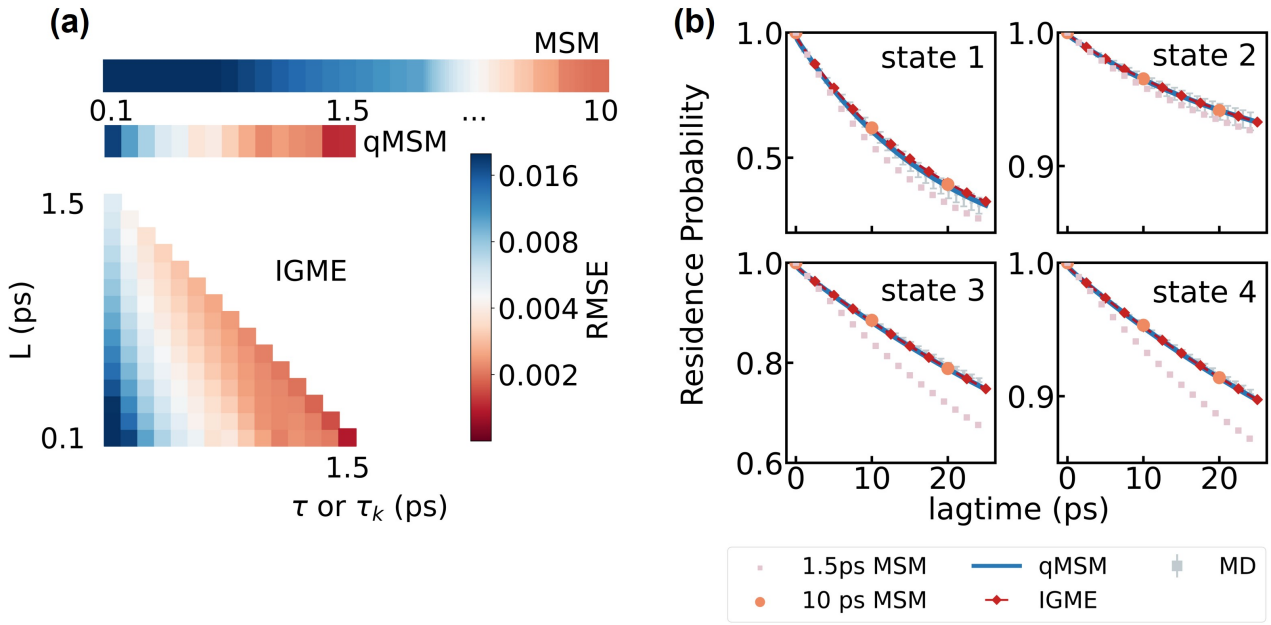

Figure 5. Building qMSM and IGME models for the alanine dipeptide. (a) The RMSE map of the IGME, qMSM and MSM. We applied Eq. (26) to compute RMSE and chose $L_x = 500$ for IGME and qMSM, while $L_x = 50\text{ns}/\tau$ for MSM with a lag time of τ. The saving internal of MD simulations is $\Delta t = 0.1ps$. **(b)** Chapman-Kolmogorov test on the 4 macrostates for the selected models. MSM has been tested when lag time $\tau = 1.5$ ps and $\tau = 10$ ps. qMSM has been tested with $\tau_K = 1.5$ ps. IGME has been tested when $\tau_K = 1.5$ ps and $L = 0.1$ ps. The error bars in the residence probability plots of MD data are calculated by bootstrapping 100 lumped trajectories 50 times, with repeated trajectories allowed.

In this section, we illustrate the process of constructing IGME models (see **Figure 1(f)** and **Scheme 8**). As discussed in **Sec 3.6**, we need to determine two hyperparameters, τ_K and L, when constructing IGME models. In this system, we conducted a systematic scan to identify their optimal values, resulting in IGME models with the minimized RMSE error (Eq. (26)). **Figure 5(a)** illustrates that the

RMSE of the optimal IGME reaches 1.4×10^{-3} at $\tau_K = 1.5$ ps and $L = 0.1$ ps. The Chapman-Kolmogorov test for this IGME model also demonstrates strong consistency between IGME predictions and MD simulations (**Figure 5(b)**). In addition, the slowest dynamics can be directly derived from the $\hat{T}$ matrix (Eq. (11)). Based on $\hat{T}$, we identify that the slowest ITS for the alanine dipeptide is ~1.15 ns, consistent with predictions from qMSM and MSM. Moreover, the MIK computed from the optimal IGME model (see Eq. (12)) aligns well with that obtained from qMSM (**Figure 4(b)**).

For the relatively simple alanine dipeptide system with sufficient sampling, all three methods, MSM, qMSM, and IGME, yield consistent results. However, MSM requires a significantly longer lag time ($\tau_M = 10$ ps) to achieve a similar RMSE compared to qMSM ($\tau_K = 1.5$ ps) and IGME ($\tau_K = 1.5$ ps and $L = 0.1$ ps). Consequently, qMSM and IGME can be constructed with shorter MD trajectories than MSM. Both qMSM and IGME consistently perform well for the alanine dipeptide. In the subsequent example of villin headpiece, we demonstrate that IGME outperforms qMSM by substantially reducing numerical instability, providing a more robust approach to constructing non-Markovian dynamic models for studying protein dynamics.

B. Villin Headpiece (HP35)

HP35 is a 35-residue peptide that exhibits ultrafast folding[106, 107], making it a suitable benchmark system for MD simulations of protein folding.[92, 108-110] The HP35 simulation dataset provided by the D. E. Shaw Research[111] consists of a single ~ 300μs all-atom MD trajectory of Nle/Nle mutant of HP35 (PDB ID: 2f4k) saved at 0.2 ns interval.

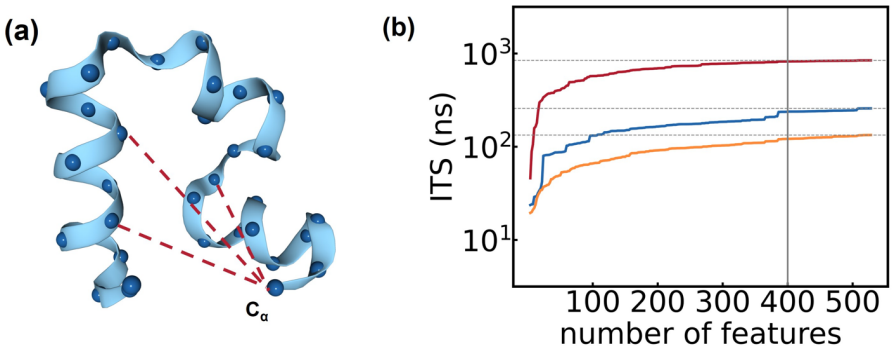

Figure 6. Feature selection of Villin headpiece. (a) Structure of villin headpiece. There are 35 residues and 528 residue-residue distances based on the distances between their alpha carbon atoms with a minimum separation of 3 residues. We use all these 528 pairwise distances as raw input features. (b) Feature selection using Spectral oASIS. 400 out of 528 features are selected.

Scheme 9 | Featurization of villin headpiece

```
from msmbuilder.featurizer import ContactFeaturizer
Cfeaturizer = ContactFeaturizer(scheme='ca')
ftrajs = Cfeaturizer.fit_transform(xyz)
```

Scheme 10 | Feature Selection using Spectral oASIS

(a)
```
sys.path.insert(0, './scripts')
```

```
import FeatureSelection

# Use self-covariance matrix for feature selection
featureselect =\
    FeatureSelection.spectral_oasis(num_select=400, num_every_iter=2,
                                    method='spectral-oasis', covariance=True)
select_columns = featureselect.select_columns
```

(b)

```
# Use time-lagged-autocorrelation matrix for feature selection
featureselect =\
    FeatureSelection.spectral_oasis(num_select=400, num_every_iter=2,
                                    method='spectral-oasis', covariance=False)
featureselect.select(ftrajs, lagtime=100)
select_columns = featureselect.select_columns
for key in range(len(ftrajs)):
    oasis_trajs.append(ftrajs[key][:, select_columns)
```

In the featurization step (**Figure 1(b)**), we first employ all 528 pairwise distances between C-alpha atoms with a minimum separation of 3 residues as raw input features (**Figure 6(a)**). This step is conducted using the "*ContactFeaturizer*" function in MSMBuilder[101] (see the sample code in **Scheme 9**). Next, we select 400 features from these 528 pairwise distances that can best describe the slowest dynamics of protein folding using Spectral oASIS[63] in the PyEMMA package[103] (see the sample code in **Scheme 10**). In the feature selection step, we employ the time-lagged autocorrelation matrix[63] with a lag time of 20 ns rather than the self-covariance matrix to achieve better performance. As shown in **Figure 6(b)**, the ITS plot demonstrates that the 400 selected features are sufficient to capture the slowest dynamics of HP35.

Following the feature selection, we proceed with the dimensionality reduction and geometric clustering (**Figure 1(c, d)**). We use two different approaches for the dimensionality reduction: tICA and SRVs. Following the sample code in **Scheme 2** and **Scheme 11**, we performed tICA and SRVs analysis to transform the 400 features into a specific number of CVs (or tICs for TICA) with a designated lag time, respectively. Utilizing the obtained CVs, we apply the K-Centers algorithm (**Scheme 3**) to partition the MD dataset into a given number of microstates. Several hyperparameters associated with this step, including the lag time, number of CVs (or tICs), and number of microstates, need to be determined. We employed the GMRQ cross-validation tool for both tICA (**Scheme 4**) and SRVs (**Scheme 12**) to sequentially choose their optimal values, starting with the lag time of 40 ns (left panels in **Figure 7**), followed by the selection of 4 tICs for tICA as well as 3 CVs for SRVs(middle panels in **Figure 7**), and concluding with 200 microstates (right panels in **Figure 7**).

Scheme 11 | SRVs for dimensionality reduction

```
import torch
import torch.nn as nn
import sys
sys.path.insert(0, './scripts')
from VAMPNet_SRVNet import *

if torch.cuda.is_available():
    device = torch.device("cuda")
    torch.backends.cudnn.benchmark = True
else:
    device = torch.device("cpu")
```

```
network_lobe = nn.Sequential(
    nn.BatchNorm1d(400),
    nn.Linear(400, 200), nn.ELU(),
    nn.Linear(200, 100), nn.ELU(),
    nn.Linear(100, 50), nn.ELU(),
    nn.Linear(50, 20), nn.ELU(),
    nn.Linear(20, 10), nn.ELU(),
    nn.Linear(10, 5), nn.ELU(),
    nn.Linear(5, 3))

network_lobe = network_lobe.to(device=device)

projector = deep_projector(network_type='SRVNet', lobe=network_lobe,
                           epsilon=1e-6, learning_rate=1e-4, device=device)

past, future = TimeLaggedDataset(trajs=oasis_trajs, lagtime=200, normalize=False)
# oasis_trajs: generated from Spectral oASIS

train_loader, validation_loader \
    =  split_train_validate_data(pastdata=past, futuredata=future,
                                 validation_ratio=0.2, train_batchsize=50000)
projector.fit(train_loader=train_loader, num_epochs=15,
              validation_loader=validation_loader)
```

Scheme 12 | GMRQ Cross-Validation based on the SRVs analysis

```
# The function TICA_CV is used for cross-validation.
# More details can be found on the github.

# Cross-validation for lag time
lt_list = [50, 100, 200, 300, 400, 500]
Parallel(n_jobs=3)(delayed(SRV_CV)(trajs=oasis_trajs,
                                   lt=i, n_cvs=3, n_clusters=200,
                                    para='lagtime') for i in lt_list)

# Cross-validation for number of CVs
ncvs_list = [2, 3, 4, 5]
Parallel(n_jobs=4)(delayed(SRV_CV)(trajs=oasis_trajs,
                                   lt=200, n_cvs=i, n_clusters=200,
                                   para='n_cvs') for i in ncvs_list)

# Cross-validation for number of microstates (clusters)
nc_list = [50, 100, 200, 300, 400, 500]
Parallel(n_jobs=3)(delayed(SRV_CV)(trajs=oasis_trajs,
                                   lt=200, n_cvs=3, n_clusters=i,
                                   para='n_clusters') for i in nc_list)
```

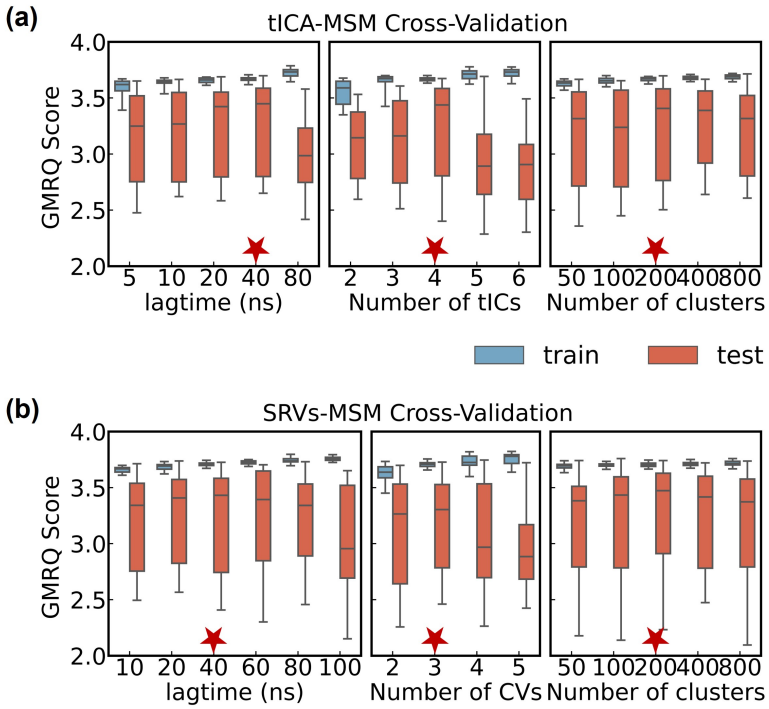

Figure 7. Cross-Validation of villin headpiece with different methods for dimensionality reduction. **(a)** Cross-validation has been conducted based on GMRQ scores to select the optimal parameters for lag time in tICA, the number of tICs, and the number of microstates (or clusters) for K-Centers clustering. The optimal parameters chosen are a 40 ns tICA lag time, 4 tICs, and 200 microstates. **(b)** Cross-validation has been performed based on GMRQ scores to select the optimal parameters for the lag time in SRVs, the number of CVs and the number of microstates (or clusters) for K-Centers clustering. The optimal parameters chosen are a 40 ns SRV lag time, 3 CVs, and 200 microstates.

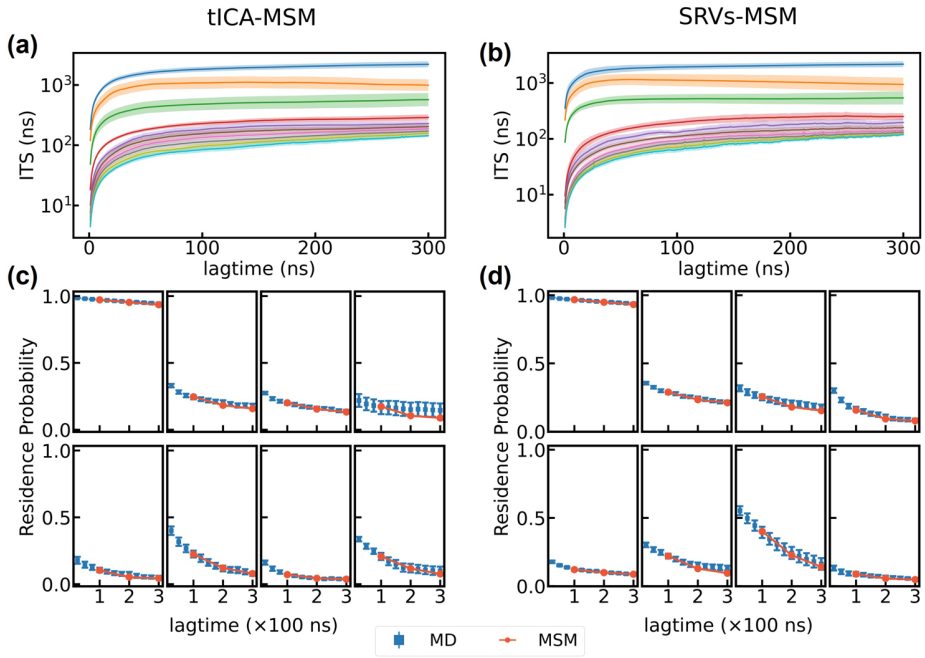

Figure 8. Construction and validation of microstate-MSMs for villin headpiece. (a, b) Implied timescales

(or ITS) for the ten slowest dynamic modes as a function of lag time. **(c, d)** Chapman-Kolmogorov test for the 8 most populated microstates. We choose the lag time to be 100 ns. The error bars in the implied timescale and the residence probability plots are calculated by bootstrapping with replacement 150 trajectories for 50 times. The results from tICA-MSM and SRV-MSM are displayed in the left and right panels, respectively.

We next validate the microstate MSMs constructed based on both tICA (denoted as tICA-MSM) and SRVs approaches (denoted as SRVs-MSM). Specifically, we followed **Scheme 5** to perform the ITS and Chapman-Kolmogorov test. For both tICA-MSM and SRVs-MSM, the ITS plots reach a plateau at ~100 ns (**Figure 8(a, b)**). Furthermore, both tICA-MSM and SRVs-MSM constructed at this lag time successfully pass Chapman-Kolmogorov test (**Figure 8(c, d)**).

With validated microstate-MSMs, we next perform kinetic lumping (**Figure 1(e)**) to group 200 microstates into 4 metastable macrostates states using PCCA+ (**Scheme 6**). We chose 4 macrostates as there exists a clear separation between the third and fourth eigenmodes based on the ITS plots, indicating there are four dominant metastable dynamic processes (see **Figure 8(a, b)**). As shown in **Figure 9**, State 1 and State 3 are partially folded states. State 2 corresponds to the unfolded state, while the most populated State 4 is the folded state. These representative conformations in **Figure 9** are chosen from the kinetic lumping results based on the microstate tICA-MSM, and similar state decomposition can be obtained using the microstate SRVs-MSM.

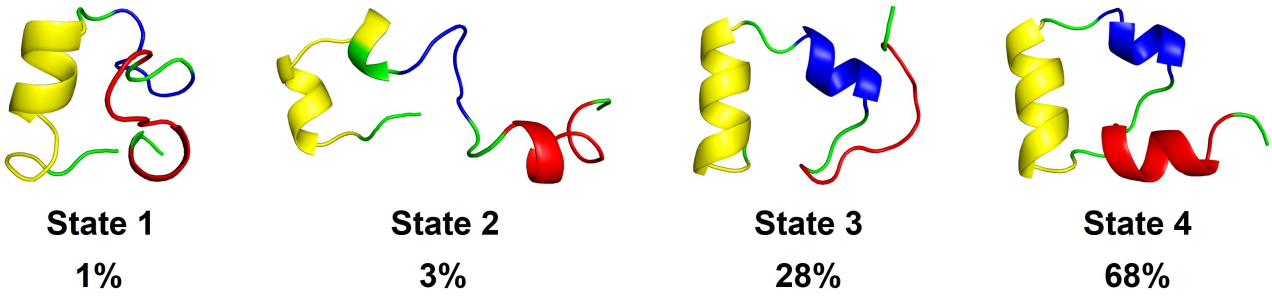

Figure 9. Representative conformations from the 4 macrostates. These conformations were chosen from the kinetic lumped model from the microstate tICA-MSM.

We proceed by computing memory kernels and compare the qMSMs constructed using the 4 macrostates based on tICA (referred to as tICA-qMSM) and SRV (referred to SRVs-qMSM) approaches (see **Figure 1(f)** and the sample code in **Scheme 13**). For tICA-qMSM, we examine the MIK plots (computed according to Eq (24)) to determine the value of $\tau_K = 30$ ns, where the integral of memory kernels has already reached a plateau (**Figure 10(a)**). The resulting tICA-qMSM exhibits a small deviation in reproducing the original MD simulation dataset (with the RMSE as low as 6×10^{-4}, see **Figure 10(c)**). Utilizing this 4-state tICA-qMSM, we also predicte the slowest ITS to be at ~1.87 μs, consistent with the value obtained from the validated 200-microstate MSMs (**Figure 8(a)**). To achieve this, we use our tICA-qMSM to obtain the TPM ($\boldsymbol{T}(n\Delta t)$) at $n\Delta t = 500$ ns (Eq (22)) to compute the slowest ITS. For the SRVs-qMSM, it takes a slightly shorter lag time ($\tau_K = 25$ ns) for the MIK plot to reach a plateau (**Figure 10(b)**). The RMSE of this model is 1.0×10^{-3}, and the predicted slowest ITS, as computed from $\boldsymbol{T}(n\Delta t)$ at $n\Delta t = 500$ ns, is ~1.64 μs.

Scheme 13 | Constructing qMSMs for villin headpiece

```
import sys
sys.path.insert(0, './scripts')
from qmsm import QuasiMSM
delta_t = 1 # the unit: ns

# Calculate MIK
qmsm = QuasiMSM()
qmsm.fit(TPM, tau_k=100, delta_t=delta_t, rmse=False)
qmsm_mik = qmsm.mik

# Build qMSM
qmsm = QuasiMSM()
qmsm.fit(TPM, tau_k=30, delta_t=delta_t) # tau_k = 25 if SRVs-based
qmsm_time, qmsm_tpm = qmsm.predict(TPM) # Use qMSM for prediction
qmsm_its = qmsm.timescales(TPM, ITS_t=500) # ITS predicted by qMSM
```

In the final step of this tutorial, we constructed tICA-IGME and SRVs-IGME models for HP35 (see **Figure 1(f)** and the sample code in **Scheme 14**). To determine the values of the two hyperparameters (τ_K and L), we conduct a systematical scan within the range of 1 ns to 50 ns (**Figure 10(c, d)**). For tICA-IGME and SRVs-IGME, we identify the best models with the smallest RMSE error at $\{\tau_K = 31\ \text{ns}, L = 1\ \text{ns}\}$ and $\{\tau_K = 22\ \text{ns}, L = 1\ \text{ns}\}$, respectively. In the Chapman-Kolmogorov test, both models accurately predict the time evolutions of state residence probabilities, aligning well with the original MD simulations (**Figure 10(e-f)**). Furthermore, we notice that all top 5% of the IGME models exhibited RMSE errors below 1.0×10^{-3}, indicating high accuracy. Based on these models, we calculate the average value and standard deviations of the slowest ITS (based on $\hat{T}$, see Eq. (11)) to be 1.95 ± 0.30 μs and 1.87 ± 0.41 μs for tICA-IGME and SRVs-IGME, respectively. As expected, the MIK obtained from IGME (Eq. (12)) is consistent with that from qMSM (**Figure 10(a, b)**). For comparisons, we also constructed macrostate MSMs for HP35. For both macrostate tICA-MSM and SRVs-MSM, the lag time needs to be as long as $\tau = 150$ ns for the models to achieve Markovian behavior and pass the Chapman-Kolmogorov test (**Figure 10(e-f)**). This Markovian lag time ($\tau_M = 150\ \text{ns}$) is several times longer than τ_K for qMSM and IGME models. Furthermore, MSMs consistently exhibit larger RMSE errors compared to qMSM and IGME models (**Figure 10(c-d)**).

Scheme 14 | Constructing IGME models for villin headpiece

```
import sys
sys.path.insert(0, './scripts')
from igme import IGMENS, IGME
delta_t = 1 # the unit: ns

# Scan tau_k and L
igme = IGME()
scan_output = igme.scan(input_data=TPM, begin=1, end=50)

# Build top IGME for CK-test
igme_top = igme.top_model(scan_output, 1) # Select the top IGME model
igme_tpm = np.array(igme_top.predict(1,len(TPM))) # Use IGME for prediction

# Compute MIK and ITS based on top 5% IGME models
top_outputs = igme.top_outputs(scan_output, n=0.05, max_its=1e5)
```

```
igme_mik = np.array(top_outputs['mik'])[:,1] / delta_t
igme_its = np.array(top_outputs['timescales']) * delta_t
```

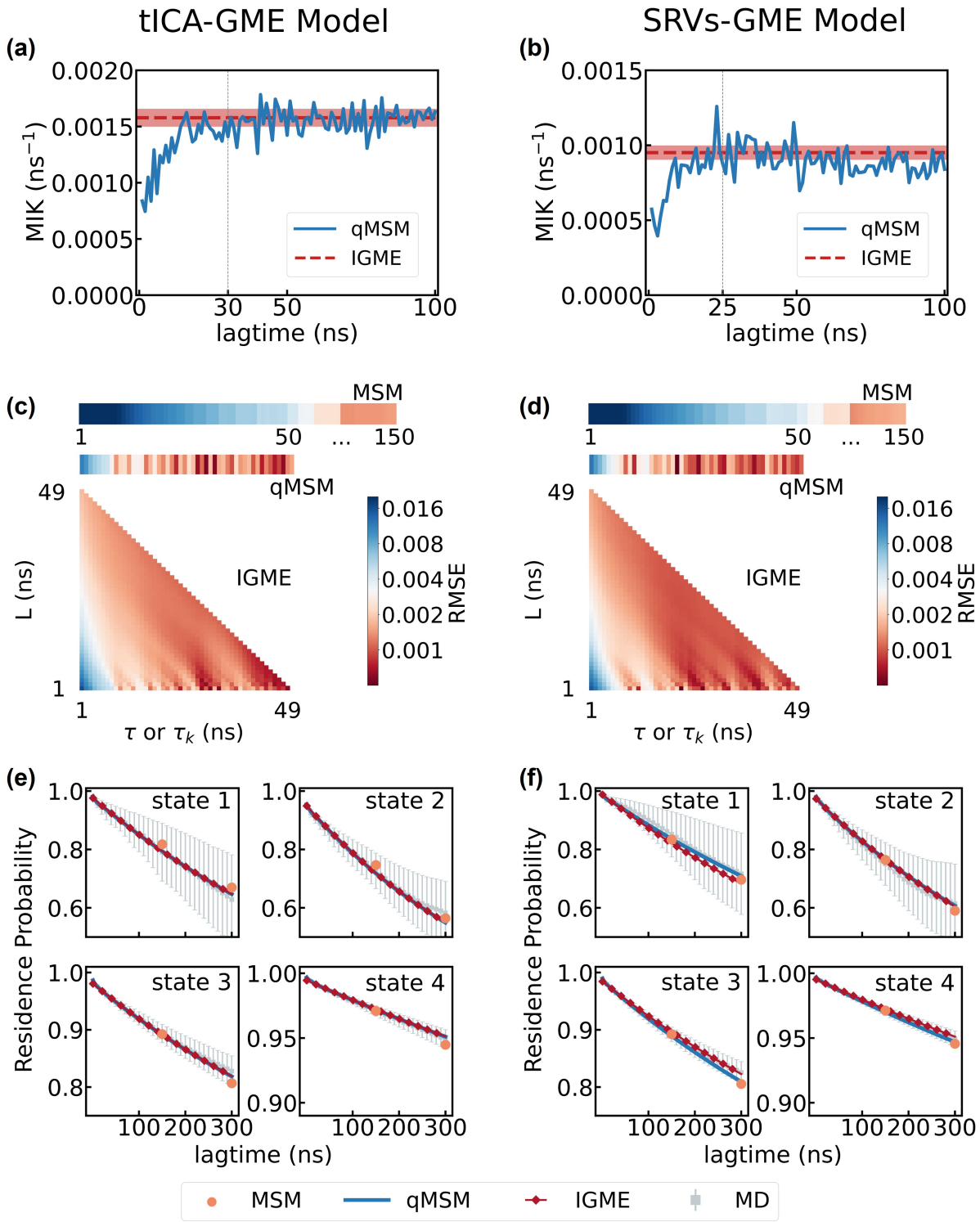

Figure 10. Building qMSM and IGME models for the villin headpiece. (a). The mean integral kernel (MIK) calculated from qMSM (blue) and IGME (red) for the tICA-qMSM and tICA-IGME models. The red dash line is the mean MIK of the top 5% IGME models and the shade area indicates their standard deviations. **(b).** The same as **(a)** except that the results from SRVs-qMSM and SRVs-IGME models are shown. **(c)** The RMSE map

of the tICA-IGME, tICA-qMSM and macrostate tICA-MSM. We applied **Eq. (26)** to compute RMSE and chose $L_x = 300$ for IGME and qMSM, while $L_x = 300\text{ns}/\tau$ for MSM with a lag time of τ. The saving internal of MD simulations is $\Delta t = 1ns$. **(d).** The same as **(c)** except that the results from SRVs-IGME, SRVss-qMSM and macrostate SRV-MSM are shown. (**e**) Chapman-Kolmogorov test on the tICA-based macrostates models. Specifically, the lag time of $\tau_M = 150$ ns, $\tau_K = 30$ ns, and $\tau_K = 31$ ns, $L = 1$ ns are used for the MSM, qMSM and IGME models, respectively. (**f**) Chapman-Kolmogorov test on the SRVs-based macrostates models. Specifically, the lag time of $\tau_M = 150$ ns, $\tau_K = 25$ ns, and $\tau_K = 22$ ns, $L = 1$ ns are used for the MSM, qMSM and IGME models, respectively. In (**e, f**), the error bars are calculated by bootstrapping 150 MD trajectories 50 times with replacement.

5. Conclusions

In this tutorial, we offer a comprehensive, step-by-step guide on constructing non-Markovian dynamics models, specifically qMSM and IGME, for investigating protein dynamics. Using two MD simulation datasets—alanine dipeptide and villin headpiece—we provide detailed instructions along with associated sample codes covering the entire model construction protocol (see **Figure 1**). This protocol includes feature selection, dimensionality reduction, geometric clustering, kinetic lumping, and the creation of qMSM and IGME models. All the steps and Python codes can be accessed on our GitHub repository (https://github.com/xuhuihuang/GME_tutorials). We believe that this tutorial will prove valuable to a broad audience in computational biophysics who are interested in exploring the dynamics of proteins and other biological macromolecules.

6. Acknowledgements

XH acknowledges the support from NIH/NIGMS under award number R01GM147652-01A1. XH also acknowledges the support from the Hirschfelder Professorship Fund.

7. Appendix

A1. The least-square fitting method to fit hyperparameters in IGME

In our implementation, we utilized the following form of Eq (11):

$$\ln \boldsymbol{T}(t) \approx \ln \boldsymbol{A} + t \ln \widehat{\boldsymbol{T}} \tag{A1}$$

A simple Lagrangian to minimize the error of the above equation can be defined as:

$$L_u = \frac{1}{2}\sum_t \left|\ln \boldsymbol{T}(t) - \ln \boldsymbol{A} - t \ln \widehat{\boldsymbol{T}}\right|_F^2 \tag{A2}$$

Where the subscribe "F" represents the Frobenius norm. We next introduced an additional constrain to the above Lagrangian. As the TPM, $T(t)$, shall satisfy the row-sum rule, i.e., $\sum_j T_{ij}(t) = 1$. To consider this constrain, we utilized the logarithm form of the row-sum rule as derived below:

$$\begin{aligned}
&\sum_j T_{ij}(t) = 1 \Rightarrow T(t)\begin{bmatrix}1\\1\\\dots\\1\end{bmatrix} = \begin{bmatrix}1\\1\\\dots\\1\end{bmatrix}\\
&\Rightarrow \lambda_1\big(T(t)\big) = 1, v_1\big(T(t)\big) = [1,1,\dots,1]^T\\
&\Rightarrow \lambda_1(\ln T(t)) = 0, v_1(\ln T(t)) = [1,1,\dots,1]^T\\
&\Rightarrow [\ln T(t)]\begin{bmatrix}1\\1\\\dots\\1\end{bmatrix} = 0 \Rightarrow \sum_j [\ln T(t)]_{ij} = 0
\end{aligned} \tag{A3}$$

Where λ_1 and v_1 refer to the first eigenvalue and eigenvector, respectively. In IGME, the input $T(t)$ (i.e., $T(\tau_K), T(\tau_K + \Delta t), \dots, T(\tau_K + L)$) generated by MSMBuilder already satisfy the row-sum rule. However, predicted TPMs, $T(t > \tau_K + L)$, may not satisfy the row-sum rule if this constraint is not explicitly included in the least-square fitting. In order to guaranteed that all predicted TPMs satisfy the row-sum rule, we need to ensure both A and $\hat{T}$ satisfy the row-sum rule. However, the row-sum rule of A is automatically satisfied when the input $T(t)$ and $\hat{T}$ satisfy the row-sum rule (since $\sum_j [\ln A]_{ij} = \sum_j \left[\overline{\ln T(t)} - \bar{t}\ln \hat{T}\right]_{ij}$, see Eq. (A5)), thus in the least-square fitting we only need to ensure $\hat{T}$ follows the row-sum rule: $\sum_j \hat{T}_{ij} = 1$ or $\sum_j [\ln \hat{T}]_{ij} = 0$. Therefore, the complete Lagrangian with the row-sum constraint of $T(t)$ can be written as follows:

$$L = \frac{1}{2}\sum_t \left|\ln \boldsymbol{T}(t) - \ln \boldsymbol{A} - t \ln \hat{\boldsymbol{T}}\right|_F^2 + \sum_i \left(\gamma_i \sum_j [\ln \hat{\boldsymbol{T}}]_{ij}\right) \tag{A4}$$

Here γ_i are Lagrange multipliers for each row of $\ln \hat{\boldsymbol{T}}$. Taking $\partial/\partial_{\hat{T}_{ij}}$, $\partial/\partial_{A_{ij}}$ and $\partial/\partial_{\gamma_i}$ to L of Eq. (A4), the solution of these Lagrange equations are:

$$\begin{aligned}
&\left(\bar{t}[\ln A]_{ij} + \overline{t^2}[\ln \hat{T}]_{ij} + \gamma_i - \overline{t[\ln T(t)]_{ij}}\right)[\ln \hat{T}]_{ij} = 0\\
&\left([\ln A]_{ij} + \bar{t}[\ln \hat{T}]_{ij} - \overline{[\ln T(t)]_{ij}}\right)[\ln A]_{ij} = 0\\
&\sum_j [\ln \hat{T}]_{ij} = 0
\end{aligned} \tag{A5}$$

The above solution can be rewritten into a matrix form for the least-square fitting. For each row i, the least-square fitting is done by solving the following matrix equation to get $\ln A$, $\ln \hat{T}$ and γ:

$$\begin{bmatrix}
\bar{t} & & & \overline{t^2} & & & 1\\
& \bar{t} & & & \overline{t^2} & & 1\\
& & \dots & & & \dots & \dots\\
1 & & & \bar{t} & & & 0\\
& 1 & & & \bar{t} & & 0\\
& & \dots & & & \dots & \dots\\
0 & 0 & \dots & 1 & 1 & \dots & 0
\end{bmatrix}
\begin{bmatrix}
[\ln A]_{i1}\\
[\ln A]_{i2}\\
\dots\\
[\ln \hat{T}]_{i1}\\
[\ln \hat{T}]_{i2}\\
\dots\\
\gamma_i
\end{bmatrix}
=
\begin{bmatrix}
\overline{t[\ln T(t)]_{i1}}\\
\overline{t[\ln T(t)]_{i2}}\\
\dots\\
\overline{[\ln T(t)]_{i1}}\\
\overline{[\ln T(t)]_{i2}}\\
\dots\\
0
\end{bmatrix} \tag{A6}$$

8. References

1. K. Henzler-Wildman and D. Kern, Nature **450** (7172), 964-972 (2007).
2. I. Bahar, T. R. Lezon, L. W. Yang and E. Eyal, Annu Rev Biophys **39**, 23-42 (2010).
3. F. Brueckner, J. Ortiz and P. Cramer, Current opinion in structural biology **19** (3), 294-299 (2009).
4. L. Zhang, F. Pardo-Avila, I. C. Unarta, P. P. Cheung, G. Wang, D. Wang and X. Huang, Acc Chem Res **49** (4), 687-694 (2016).
5. G. R. Bowman, E. R. Bolin, K. M. Hart, B. C. Maguire and S. Marqusee, Proc Natl Acad Sci USA **112** (9), 2734-2739 (2015).
6. J. R. Wagner, C. T. Lee, J. D. Durrant, R. D. Malmstrom, V. A. Feher and R. E. Amaro, Chem Rev **116** (11), 6370-6390 (2016).
7. I. C. Unarta, S. Cao, S. Kubo, W. Wang, P. P. Cheung, X. Gao, S. Takada and X. Huang, Proc Natl Acad Sci USA **118** (17), e2024324118 (2021).
8. J. D. Chodera, N. Singhal, V. S. Pande, K. A. Dill and W. C. Swope, J Chem Phys **126** (15), 155101 (2007).
9. J. H. Prinz, H. Wu, M. Sarich, B. Keller, M. Senne, M. Held, J. D. Chodera, C. Schutte and F. Noe, J Chem Phys **134** (17), 174105 (2011).
10. K. A. Konovalov, I. C. Unarta, S. Cao, E. C. Goonetilleke and X. Huang, JACS Au **1** (9), 1330-1341 (2021).
11. L. Zhang, H. Jiang, F. K. Sheong, F. Pardo-Avila, P. P.-H. Cheung and X. Huang, Methods in Enzymology **578**, 343–371 (2016).
12. W. Wang, S. Cao, L. Zhu and X. Huang, Wires Comput Mol Sci **8**, e1343 (2018).
13. A. C. Pan and B. Roux, J Chem Phys **129** (6), 064107 (2008).
14. B. W. Zhang, W. Dai, E. Gallicchio, P. He, J. C. Xia, Z. Q. Tan and R. M. Levy, J Phys Chem B **120** (33), 8289-8301 (2016).
15. F. Morcos, S. Chatterjee, C. L. McClendon, P. R. Brenner, R. López-Rendón, J. Zintsmaster, M. Ercsey-Ravasz, C. R. Sweet, M. P. Jacobson, J. W. Peng and J. A. Izaguirre, Plos Comput Biol **6** (12), e1001015 (2010).
16. X. H. Huang, G. R. Bowman, S. Bacallado and V. S. Pande, P Natl Acad Sci USA **106** (47), 19765-19769 (2009).
17. R. D. Malmstrom, C. T. Lee, A. T. Van Wart and R. E. Amaro, J Chem Theory Comput **10** (7), 2648-2657 (2014).
18. N. V. Buchete and G. Hummer, J Phys Chem B **112** (19), 6057-6069 (2008).
19. C. Lorpaiboon, E. H. Thiede, R. J. Webber, J. Weare and A. R. Dinner, J Phys Chem B **124** (42), 9354-9364 (2020).
20. Q. Qiao, G. R. Bowman and X. H. Huang, J Am Chem Soc **135** (43), 16092-16101 (2013).
21. F. Noé, C. Schütte, E. Vanden-Eijnden, L. Reich and T. R. Weikl, P Natl Acad Sci USA **106** (45), 19011-19016 (2009).
22. G. R. Bowman, V. A. Voelz and V. S. Pande, Current opinion in structural biology **21** (1), 4-11 (2011).
23. N. J. Deng, W. Dai and R. M. Levy, J Phys Chem B **117** (42), 12787-12799 (2013).
24. H. B. Wan, Y. H. Ge, A. Razavi and V. A. Voelz, J Chem Theory Comput **16** (2), 1333-1348 (2020).
25. Y. Qiu, M. S. O'Connor, M. Xue, B. Liu and X. Huang, J Chem Theory Comput **19** (14), 4728-4742 (2023).
26. I. Buch, T. Giorgino and G. De Fabritiis, P Natl Acad Sci USA **108** (25), 10184-10189 (2011).
27. M. Lawrenz, D. Shukla and V. S. Pande, Sci Rep **5**, 7918 (2015).
28. D. A. Silva, G. R. Bowman, A. Sosa-Peinado and X. H. Huang, Plos Comput Biol **7** (5), e1002054 (2011).
29. N. Plattner and F. Noé, Nat Commun **6**, 7653 (2015).
30. H. Jiang, F. K. Sheong, L. Zhu, X. Gao, J. Bernauer and X. Huang, Plos Comput Biol **11** (7), e1004404 (2015).
31. D. A. Silva, D. R. Weiss, F. Pardo Avila, L. T. Da, M. Levitt, D. Wang and X. Huang, Proc Natl Acad Sci USA **111** (21), 7665-7670 (2014).
32. K. J. Kohlhoff, D. Shukla, M. Lawrenz, G. R. Bowman, D. E. Konerding, D. Belov, R. B. Altman and V. S. Pande, Nat Chem **6** (1), 15-21 (2014).

33. L. T. Da, F. Pardo-Avila, L. Xu, D. A. Silva, L. Zhang, X. Gao, D. Wang and X. Huang, Nat Commun **7**, 11244 (2016).
34. L. T. Da, E. Chao, B. G. Duan, C. B. Zhang, X. Zhou and J. Yu, Plos Comput Biol **11** (11), e1004624 (2015).
35. L. T. Da, D. Wang and X. H. Huang, J Am Chem Soc **134** (4), 2399-2406 (2012).
36. R. D. Malmstrom, A. P. Kornev, S. S. Taylor and R. E. Amaro, Nat Commun **6**, 7588 (2015).
37. B. B. Wang, R. E. Sexton and M. Feig, Bba-Gene Regul Mech **1860** (4), 482-490 (2017).
38. M. Khaled, A. Gorfe and A. Sayyed-Ahmad, J Phys Chem B **123** (36), 7667-7675 (2019).
39. E. P. Barros, Ö. Demir, J. Soto, M. J. Cocco and R. E. Amaro, Chem Sci **12** (5), 1891-1900 (2021).
40. J. Y. Feng, B. Selvam and D. Shukla, Structure **29** (8), 922-933 (2021).
41. C. Y. Son, A. Yethiraj and Q. Cui, P Natl Acad Sci USA **114** (42), E8830-E8836 (2017).
42. L. T. Da, F. Pardo Avila, D. Wang and X. Huang, Plos Comput Biol **9** (4), e1003020 (2013).
43. Y. R. Qiu, M. S. O'Connor, M. Y. Xue, B. J. Liu and X. H. Huang, J Chem Theory Comput **19** (14), 4728-4742 (2023).
44. B. J. Liu, M. Y. Xue, Y. R. Qiu, K. A. Konovalov, M. S. O'Connor and X. H. Huang, J Chem Phys **159** (9), 094901 (2023).
45. A. K.-H. Yik, Y. Qiu, I. C. Unarta, S. Cao and X. Huang, in *A Practical Guide to Recent Advances in Multiscale Modeling and Simulation of Biomolecules*, edited by Y. Wang and R. Zhou (AIP Publishing LLC).
46. B. Liu, Y. Qiu, E. C. Goonetilleke and X. Huang, MRS Bulletin **47** (9), 958-966 (2022).
47. M. I. Zimmerman, J. R. Porter, M. D. Ward, S. Singh, N. Vithani, A. Meller, U. L. Mallimadugula, C. E. Kuhn, J. H. Borowsky, R. P. Wiewiora, M. F. D. Hurley, A. M. Harbison, C. A. Fogarty, J. E. Coffland, E. Fadda, V. A. Voelz, J. D. Chodera and G. R. Bowman, Nat Chem **13** (7), 651-659 (2021).
48. V. A. Voelz, G. R. Bowman, K. Beauchamp and V. S. Pande, J Am Chem Soc **132** (5), 1526-1528 (2010).
49. A. J. Dominic, 3rd, S. Cao, A. Montoya-Castillo and X. Huang, J Am Chem Soc **145** (18), 9916-9927 (2023).
50. S. Cao, A. Montoya-Castillo, W. Wang, T. E. Markland and X. Huang, J Chem Phys **153** (1), 014105 (2020).
51. S. Cao, Y. Qiu, M. L. Kalin and X. Huang, J Chem Phys **159** (13), 134106 (2023).
52. A. J. Dominic, 3rd, T. Sayer, S. Cao, T. E. Markland, X. Huang and A. Montoya-Castillo, Proc Natl Acad Sci USA **120** (12), e2221048120 (2023).
53. R. Hegger and G. Stock, J Chem Phys **130** (3), 034106 (2009).
54. C. Ayaz, L. Tepper, F. N. Brunig, J. Kappler, J. O. Daldrop and R. R. Netz, Proc Natl Acad Sci USA **118** (31), e2023856118 (2021).
55. C. Ayaz, L. Scalfi, B. A. Dalton and R. R. Netz, Phys Rev E **105** (5), 054138 (2022).
56. F. Noé, H. Wu, J.-H. Prinz and N. Plattner, J Chem Phys **139** (18), 184114 (2013).
57. L. Zhu, H. Jiang, S. Cao, I. C. Unarta, X. Gao and X. Huang, Commun Biol **4** (1), 1345 (2021).
58. J. Cerrillo and J. S. Cao, Phys Rev Lett **112** (11), 110401 (2014).
59. S. Presse, J. Lee and K. A. Dill, J Phys Chem B **117** (2), 495-502 (2013).
60. F. Noé and S. Fischer, Current opinion in structural biology **18** (2), 154-162 (2008).
61. S. J. Peng, X. W. Wang, L. Zhang, S. S. He, X. S. Zhao, X. H. Huang and C. L. Chen, P Natl Acad Sci USA **117** (36), 21889-21895 (2020).
62. R. Patel, T. A. Goldstein, E. L. Dyer, A. Mirhoseini and R. G. Baraniuk, arXiv:1505.05208 (2015).
63. F. Litzinger, L. Boninsegna, H. Wu, F. Nüske, R. Patel, R. Baraniuk, F. Noé and C. Clementi, J Chem Theory Comput **14** (5), 2771-2783 (2018).
64. W. Stacklies, C. Seifert and F. Graeter, Bmc Bioinformatics **12**, 101 (2011).
65. G. Diez, D. Nagel and G. Stock, J Chem Theory Comput **18** (8), 5079-5088 (2022).
66. A. Amadei, A. B. M. Linssen and H. J. C. Berendsen, Proteins: Structure, Function, and Bioinformatics **17** (4), 412-425 (1993).
67. C. R. Schwantes and V. S. Pande, J Chem Theory Comput **9** (4), 2000-2009 (2013).

68. G. Pérez-Hernández, F. Paul, T. Giorgino, G. De Fabritiis and F. Noé, J Chem Phys **139** (1), 015102 (2013).
69. Y. Naritomi and S. Fuchigami, J Chem Phys **139** (21), 215102 (2013).
70. A. Mardt, L. Pasquali, H. Wu and F. Noé, Nat Commun **9**, 5 (2018).
71. B. Liu, M. Xue, Y. Qiu, K. A. Konovalov, M. S. O'Connor and X. Huang, J Chem Phys **159** (9) (2023).
72. M. Ghorbani, S. Prasad, J. B. Klauda and B. R. Brooks, J Chem Phys **156** (18), 184103 (2022).
73. H. Sidky, W. Chen and A. L. Ferguson, J Phys Chem B **123** (38), 7999-8009 (2019).
74. B. E. Husic and F. Noé, J Chem Phys **151** (5), 054103 (2019).
75. D. Nagel, A. Weber and G. Stock, J Chem Theory Comput **16** (12), 7874-7882 (2020).
76. W. Chen, H. Sidky and A. L. Ferguson, J Chem Phys **150** (21), 214114 (2019).
77. C. R. Schwantes and V. S. Pande, J Chem Theory Comput **11** (2), 600-608 (2015).
78. L. Bonati, G. Piccini and M. Parrinello, Proc Natl Acad Sci USA **118** (44), e2113533118 (2021).
79. C. Wehmeyer and F. Noé, J Chem Phys **148** (24), 241703 (2018).
80. C. X. Hernández, H. K. Wayment-Steele, M. M. Sultan, B. E. Husic and V. S. Pande, Phys Rev E **97** (6), 062412 (2018).
81. Y. H. Wang, J. M. L. Ribeiro and P. Tiwary, Nat Commun **10**, 3573 (2019).
82. D. D. Wang and P. Tiwary, J Chem Phys **154** (13), 134111 (2021).
83. A. L. Ferguson, A. Z. Panagiotopoulos, I. G. Kevrekidis and P. G. Debenedetti, Chem Phys Lett **509** (1-3), 1-11 (2011).
84. H. Wu and F. Noé, arXiv:2309.05878 (2023).
85. A. Mitsutake, H. Iijima and H. Takano, J Chem Phys **135** (16), 164102 (2011).
86. J.-h. Peng, W. Wang, Y.-q. Yu, H.-l. Gu and X. Huang, Chin J Chem Phys **31** (4), 404-420 (2018).
87. S. P. Lloyd, Ieee T Inform Theory **28** (2), 129-137 (1982).
88. D. S. Hochbaum and D. B. Shmoys, Math Oper Res **10** (2), 180-184 (1985).
89. Y. Zhao, F. K. Sheong, J. Sun, P. Sander and X. Huang, J Comput Chem **34** (2), 95-104 (2013).
90. H. S. Park and C. H. Jun, Expert Syst Appl **36** (2), 3336-3341 (2009).
91. F. K. Sheong, D. A. Silva, L. M. Meng, Y. T. Zhao and X. H. Huang, J Chem Theory Comput **11** (1), 17-27 (2015).
92. H. Klem, G. M. Hocky and M. McCullagh, J Chem Theory Comput **18** (5), 3218-3230 (2022).
93. S. Liu, L. Z. Zhu, F. K. Sheong, W. Wang and X. H. Huang, J Comput Chem **38** (3), 152-160 (2017).
94. P. Deuflhard, W. Huisinga, A. Fischer and C. Schütte, Linear Algebra Appl **315** (1-3), 39-59 (2000).
95. P. Deuflhard and M. Weber, Linear Algebra Appl **398**, 161-184 (2005).
96. S. Röblitz and M. Weber, Adv Data Anal Classi **7** (2), 147-179 (2013).
97. Y. Yao, R. Z. Cui, G. R. Bowman, D. A. Silva, J. Sun and X. H. Huang, J Chem Phys **138** (17), 174106 (2013).
98. W. Wang, T. Liang, F. K. Sheong, X. Fan and X. Huang, J Chem Phys **149** (7), 072337 (2018).
99. H. B. Wan and V. A. Voelz, J Chem Phys **152** (2), 024103 (2020).
100. R. T. McGibbon and V. S. Pande, J Chem Phys **142** (12), 124105 (2015).
101. M. P. Harrigan, M. M. Sultan, C. X. Hernandez, B. E. Husic, P. Eastman, C. R. Schwantes, K. A. Beauchamp, R. T. McGibbon and V. S. Pande, Biophys J **112** (1), 10-15 (2017).
102. G. R. Bowman, X. Huang and V. S. Pande, Methods **49** (2), 197-201 (2009).
103. M. K. Scherer, B. Trendelkamp-Schroer, F. Paul, G. Perez-Hernandez, M. Hoffmann, N. Plattner, C. Wehmeyer, J. H. Prinz and F. Noe, J Chem Theory Comput **11** (11), 5525-5542 (2015).
104. V. Hornak, R. Abel, A. Okur, B. Strockbine, A. Roitberg and C. Simmerling, Proteins **65** (3), 712-725 (2006).
105. W. L. Jorgensen, J. Chandrasekhar, J. D. Madura, R. W. Impey and M. L. Klein, J Chem Phys **79** (2), 926-935 (1983).
106. K. Lindorff-Larsen, S. Piana, R. O. Dror and D. E. Shaw, Science **334** (6055), 517-520 (2011).
107. J. Kubelka, E. R. Henry, T. Cellmer, J. Hofrichter and W. A. Eaton, Proc Natl Acad Sci USA **105** (48), 18655-18662 (2008).

108. D. Nagel, S. Sartore and G. Stock, J Phys Chem Lett **14** (31), 6956-6967 (2023).
109. P. V. Banushkina and S. V. Krivov, J Chem Theory Comput **9** (12), 5257-5266 (2013).
110. D. Nagel, S. Sartore and G. Stock, J Chem Theory Comput **19** (11), 3391-3405 (2023).
111. S. Piana, K. Lindorff-Larsen and D. E. Shaw, Proc Natl Acad Sci USA **109** (44), 17845-17850 (2012).