LaM-SLidE: Latent Space Modeling of Spatial Dynamical Systems via Linked Entities

Florian Sestak[1], **Artur P. Toshev**[2], **Andreas Fürst**[1], **Günter Klambauer**[*,1,5], **Andreas Mayr**[*,1], **Johannes Brandstetter**[*,1,4]
[1] ELLIS Unit Linz and LIT AI Lab, Institute for Machine Learning, Johannes Kepler University Linz, Austria
[2] Department of Engineering Physics and Computation, TUM, Germany
[3] Munich Institute of Integrated Materials, Energy and Process Engineering, TUM, Germany
[4] Emmi AI GmbH, Linz, Austria
[5] NXAI GmbH, Linz, Austria
{klambauer,mayr,brandstetter}@ml.jku.at
* Equal contribution

Abstract

Generative models are spearheading recent progress in deep learning, showing strong promise for trajectory sampling in dynamical systems as well. However, while latent space modeling paradigms have transformed image and video generation, similar approaches are more difficult for most dynamical systems. Such systems – from chemical molecule structures to collective human behavior – are described by interactions of entities, making them inherently linked to connectivity patterns and the traceability of entities over time. Our approach, LaM-SLidE (Latent Space Modeling of Spatial Dynamical Systems via Linked Entities), combines the advantages of graph neural networks, i.e., the traceability of entities across time-steps, with the efficiency and scalability of recent advances in image and video generation, where pre-trained encoder and decoder are frozen to enable generative modeling in the latent space. The core idea of LaM-SLidE is to introduce identifier representations (IDs) to allow for retrieval of entity properties, e.g., entity coordinates, from latent system representations and thus enables traceability. Experimentally, across different domains, we show that LaM-SLidE performs favorably in terms of speed, accuracy, and generalizability. Code is available at https://github.com/ml-jku/LaM-SLidE.

1 Introduction

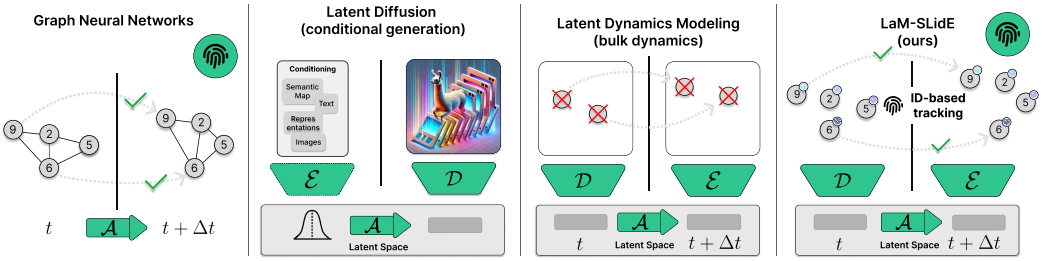

Figure 1: LaM-SLidE modeling paradigm. Right: LaM-SLidE combines the advantages of GNNs, i.e., the traceability of particles across time-steps, with the efficiency and scalability of generative latent space modeling approaches.

Understanding the dynamics of spatial systems is a fundamental challenge in many scientific and engineering domains (Karplus & Petsko, 1990; Jumper et al., 2021; Price et al., 2025). In this paper, we focus on *spatial* dynamical systems, where *scenes* are composed of distinguishable entities

at defined spatial locations. Modeling temporal trajectories of such entities quickly becomes challenging, especially when (i) stochasticity is involved, and (ii) when entities should be *traceable*. A prime example is molecular dynamics (Karplus & Petsko, 1990), where trajectories of individual atoms are modeled via Langevin dynamics, which accounts for omitted degrees of freedom by using of stochastic differential equations. Consequently, the trajectories of the atoms themselves become non-deterministic, but the atoms remain traceable over time.

A conventional approach to predict spatial trajectories of entities is to represent scenes as neighborhood graphs and to subsequently process these graphs with graph neural networks (GNNs). When using GNNs (Scarselli et al., 2009; Micheli, 2009; Gilmer et al., 2017; Battaglia et al., 2018), each entity is usually represented by a node, and the spatial entities nearby are connected by an edge in the neighborhood graph. Neighborhood graphs have extensively been used for trajectory prediction tasks (Kipf et al., 2018), especially for problems with a large number of indistinguishible entities, (e.g., Sanchez-Gonzalez et al., 2020; Mayr et al., 2023). Recently, GNNs have been integrated into generative modeling frameworks to effectively capture the behavior of stochastic systems (Yu et al., 2024; Costa et al., 2024).

Despite their widespread use in modeling spatial trajectories, GNNs hardly follow recent trends in latent space modeling, where unified representations together with universality and scalability of transformer blocks (Vaswani, 2017) offer simple application across datasets and tasks, a behavior commonly observed in computer vision and language processing (Devlin, 2018; Dosovitskiy, 2020). Notably, recent breakthroughs in image and video generation can be accounted to latent space conditioned generative modeling (Ho et al., 2022; Blattmann et al., 2023). In such paradigms, pre-trained encoders and decoders are employed to map data into a latent space, where subsequent modeling is performed, leveraging the efficiency and expressiveness of this representation. This poses the question: what does it take to leverage recent techniques from generative latent space modeling to boost the modeling of stochastic trajectories of entities? Recently, it has been shown (Alkin et al., 2024b) that it is possible to model the bulk behavior of large particle systems purely in the latent space, at the cost of sacrificing the traceability of individual particles, which is acceptable or even favorable for systems where particles are indistinguishable, but challenging for, e.g., molecular modeling where atom assignments are essential.

In order to combine the advantages of GNNs, i.e., the traceability of entities across time-steps, with the efficiency and scalability of latent space approaches, we introduce LAM-SLIDE (see Figure 1). The core idea of LAM-SLIDE is the introduction of identifier representations (IDs) that allow for retrieval of entity properties, e.g., entity coordinates, from latent system representations. Consequently, we can train generative models, such as stochastic interpolants (Albergo et al., 2023), purely in the latent space, where pre-trained decoder blocks map the generated states back to the physics domain. Qualitatively, LAM-SLIDE demonstrates flexibility and favorable performance across a variety of different modeling tasks.

In summary, our contributions are the following:

- We propose LAM-SLIDE for generative modeling of stochastic trajectories, which combines the advantages of GNNs, concretely traceable entities, with the scaling properties of latent space models.
- We introduce entity structure preservation to recover the entity structure of the inputs from latent space.
- We perform experiments in different domains with varying degrees of difficulty, focusing on molecular dynamics. LAM-SLIDE performs favorably with respect to all other architectures, showcasing scalability with model size.

2 BACKGROUND & RELATED WORK

Dynamical systems. Formally, we consider a dynamical system to be defined by a state space $\mathcal{S}$, representing all possible configurations of the system, and an evolution rule $\Phi : \mathbb{R} \times \mathcal{S} \mapsto \mathcal{S}$ that determines how a state $\mathbf{s} \in \mathcal{S}$ evolves over time, and which exhibits the following properties for the

time differences $0, \hat{t}_1$, and, $\hat{t}_2$:

$$\Phi(0, \mathbf{s}) = \mathbf{s} \tag{1}$$

$$\Phi(\hat{t}_2, \Phi(\hat{t}_1, \mathbf{s})) = \Phi(\hat{t}_1 + \hat{t}_2, \mathbf{s}) \tag{2}$$

We note, that Φ does not necessarily need to be defined on the whole space $\mathbb{R} \times \mathcal{S}$, but assume this is the case for notational simplicity. The exact formal definition of random dynamical systems is more involved and consists of a base flow (noise) and a cocycle dynamical system defined on a physical phase space (Arnold, 1998). We skip the details, but assume to deal with random dynamical systems for the remainder of the paper, where the stochasticity of such random dynamical systems is inherently connected with generative modeling.

Flow Matching. Flow Matching (Lipman et al., 2022; Liu et al., 2022; Albergo et al., 2023), has emerged as a powerful alternative to diffusion models (Sohl-Dickstein et al., 2015), in generative modeling. This framework has been successfully applied across different domains like images (Esser et al., 2024), audio (Vyas et al., 2023), videos (Polyak et al., 2024), protein design (Huguet et al., 2024) and robotics (Black et al., 2024).

Latent space modeling. Latent space modeling has achieved remarkable success at image and video generation (Blattmann et al., 2023; Esser et al., 2024), where pre-trained encoders and decoders map data into a latent space, and back into the physics space. The latent space aims to preserve the essential structure and features of the original data, often following a compositional structure $\mathcal{D} \cdot \mathcal{A} \cdot \mathcal{E}$ (Seidman et al., 2022; Alkin et al., 2024a;b), where the encoder $\mathcal{E}$ maps the input signal into the latent space, the approximator $\mathcal{A}$ models a process, and the decoder maps back to the original space. Examples of approximators are conditional generative modeling techniques, e.g., generating an image given a text prompt (condition). This framework was, e.g., recently used for 3D shape generation, where 3D shapes are generated in latent space, the final shape in the spatial domain is then constructed by querying the latent representations over a fixed spatial grid (Zhang et al., 2023; Zhang & Wonka, 2024).

3 Latent Space Modeling of Spatial Dynamical Systems via Linked Entities

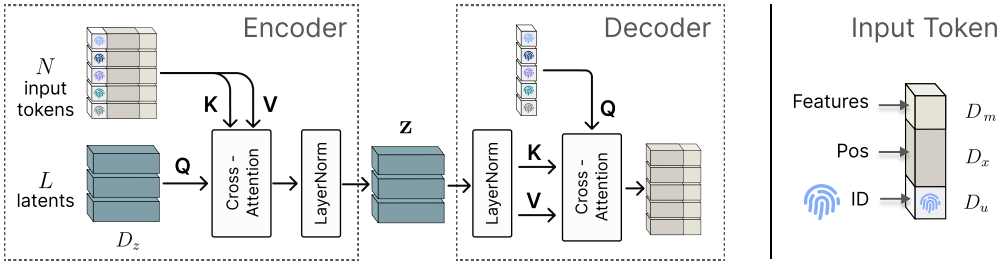

Figure 2: **Overview of our entity-preserving encoder-decoder architecture. Left:** LAM-SLIDE 's encoder maps N input tokens to a fixed size latent representation via cross-attention. The decoder reconstructs the input data from the latent space using the assigned IDs. **Right:** Structure of the input token, consisting of an ID, spatial information and features.

LAM-SLIDE introduces an identifier (ID) pool and an identifier assignment function which allow us to effectively map and retrieve latent system representations. The ID components preserve the relationships between entities, making them traceable across time-steps. LAM-SLIDE follows an encoder $\mathcal{E}$ - approximator $\mathcal{A}$ - decoder $\mathcal{D}$ paradigm.

3.1 Problem Formulation

State space. We consider spatial dynamics. Our states $\mathbf{s} \in \mathcal{S}$ describe the configuration of entities within the scene together with their individual features. We assume that a scene consists of N

entities e_i with $i \in 1, \ldots, N$. An entity e_i is described by its spatial location $\mathbf{x}_i \in \mathbb{R}^{D_x}$ and some further properties $\mathbf{m}_i \in \mathbb{R}^{D_m}$ (e.g., atom type, etc.). We refer to the state at a time point at index t within the given sequence. Analogously, we use t, $\mathbf{m}_i^t$ to describe coordinates and properties at time point t. We refer to the coordinate concatenation $[\mathbf{x}_1^t, .., \mathbf{x}_N^t]$ of the N entities in $\mathbf{s}^t$ as $\mathbf{X}^t \in \mathbb{R}^{N \times D_x}$. Analogously, we use $\mathbf{M}^t \in \mathbb{R}^{N \times D_m}$ to denote $[\mathbf{m}_1^t, .., \mathbf{m}_N^t]$. When properties are conserved over time, i.e., $\mathbf{M}^t = \mathbf{M}^0$, we just skip the time index and the time-wise repetition of states and use $\mathbf{M} \in \mathbb{R}^{N \times D_m}$. We concatenate sequences of coordinate states $\mathbf{X}^t$ with $t \in 0..T{-}1$ to a tensor $\mathbf{X} \in \mathbb{R}^{T \times N \times D_x}$, which describe a whole sampled coordinate trajectory of a system with T time points and N entities. An example for such trajectories from a dynamical systems are molecular dynamics trajectories (see Fig. 6; details in Sec. 4). A notation table is available, see App. A.

Predictive aim. Our aim is to generate the spatial continuation $\mathbf{X}^{[T_o:\, T-1]} = [\mathbf{X}^{T_o}, \mathbf{X}^t, \ldots, \mathbf{X}^{T-1}] \in \mathbb{R}^{(T-T_o) \times N \times D_x}$ of a system trajectory, given a short (observed) initial spatial trajectory $\mathbf{X}^{[0:\, T_o-1]} = [\mathbf{X}^0, \mathbf{X}^t, \ldots, \mathbf{X}^{T_o-1}] \in \mathbb{R}^{T_o \times N \times D_x}$ together with general (time-invariant) entity properties $\mathbf{M}$, where T_o represents the length of the observed trajectory.

3.2 Entity Structure Preservation

As motivated above, our aim is to maintain the integrity of scene entity structures when mapping them to and processing them in a latent space. The main idea of this module is to randomly assign an ID from an ID pool to each entity of the system, which later allows us to learn to retrieve the entity's location from the system state by using an ID-embedding as a query in cross-attention (Vaswani, 2017; Ramsauer et al., 2021). The two key components of this module are: (i) creating a fixed, finite pool of *identifiers (IDs)* and (ii) defining a stochastic function, which assigns, or links, each entity in a scene to of the unique identifiers from our identifier pool.

Definition 3.1 (Identifier pool) *We consider the set $\mathcal{I} = \{i \mid i \in \mathbb{N} \wedge i < u\}$ with $u \in \mathbb{N}$ to be an identifier pool. An identifier i is an element of the set $\mathcal{I}$.*

Definition 3.2 (Identifier assignment function) *Given an entity space, which summarizes the entities of a dynamic system, i.e., $E = \{e_1, \ldots, e_N\}$, we define a stochastic identifier assignment function ID, which maps E to $\mathcal{I}$ as follows:*

$$ID : E \times \Omega \mapsto \mathcal{I}\,, \tag{3}$$

where Ω is a sample space representing randomness, and, $\forall \omega \in \Omega\ \forall e_1, e_2 \in E : e_1 \neq e_2 \Rightarrow ID(e_1, \omega) \neq ID(e_2, \omega)$.

The condition on *ID* in Definition 3.2 is a requirement of injectivity in its first argument given a fixed second argument. Such a function might not always exist.

Proposition 3.3 *Given an identifier pool $\mathcal{I}$ as defined by definition 3.1, then an ID function according to definition 3.2 only exists, if $|E| \leqslant |\mathcal{I}|$.*

The use of a stochastic function allows us to apply our entity structure preservation concept to a wide class of problems, as no exact assignment algorithm between entities and IDs needs to be specified explicitly. Instead, it only matters that an injective assignment is made. Further, Proposition 3.3 suggests to use an identifier pool which is large enough, such that a learned model based on this Identifier Pool can generalize across systems with varying numbers of entities.

3.3 Model Architecture: Latent Space Modeling

Since predicting continuations of system trajectories is a conceptually similar task to generating videos from an initial sequence of images, we took inspiration from Blattmann et al. (2023) in using a latent diffusion architecture. We also took inspiration from Jaegle et al. (2021) to decompose our model architecture as follows: To map the state of the system composed of N entities to a latent space containing L latent tokens ($\in \mathbb{R}^{D_z}$), we use a cross-attention mechanism. In the resulting latent space, we aim to train an approximator to predict future latent states based on the embedded initial states. Inversely to the encoder, we again use a cross-attention mechanism to retrieve the latent information for the entities of the system. To wrap it up, LAM-SLIDE , is built up by an encoder ($\mathcal{E}$) - approximator ($\mathcal{A}$) - decoder (D) architecture, which represents the following function:

$$\mathcal{D} \circ \mathcal{A} \circ \mathcal{E} : \mathbb{R}^{T_o \times N \times D_x} \times \mathbb{R}^{N \times D_m} \mapsto \mathbb{R}^{(T-T_o) \times N \times D_x}$$

A detailed composition of $\mathcal{E}$ and $\mathcal{D}$ is shown in the left part of Fig. 2.

Encoder. The encoder $\mathcal{E}$ aims to encode a state of the system such that the properties of each individual entity e_n can be decoded (retrieved) later. At the same time the structure of the latent state representation ($\in \mathbb{R}^{L \times D_z}$) should not depend on N, i.e., L and D_z are constants and serve as hyperparameters, while individual samples may be composed of different numbers N of entities, as opposed to GNNs, for which the size of the latent depends on the number of nodes throughout each message passing step.

To allow for traceability of the entities, we first embed each identifier i in the space $\mathbb{R}^{D_u}$ by a learned embedding $\texttt{IDEmb}\colon \mathcal{I} \mapsto \mathbb{R}^{D_u}$. Then we draw a random number $\omega \in \Omega$ and map all ($n = 1, \ldots, N$) system entities e_n to $\mathbf{u}_n \in \mathbb{R}^{D_u}$ as follows:

$$\mathbf{u}_n = \texttt{IDEmb}(\texttt{ID}(e_n, \omega)) \qquad \forall n \in 1, \ldots, N \tag{4}$$

The inputs to the encoder comprise the (time-specific) location $\mathbf{x}_n^t \in \mathbb{R}^{D_x}$, properties $\mathbf{m}_n \in \mathbb{R}^{D_x}$, and an identity representation $\mathbf{u}_n \in \mathbb{R}^{D_x}$, as visualized in the right part of Fig. 2. We concatenate the different types of features across the entities of the system: $\mathbf{X}^t = [\mathbf{x}_1^t, .., \mathbf{x}_N^t], \mathbf{M} = [\mathbf{m}_1, .., \mathbf{m}_N], \mathbf{U}_\omega = [\mathbf{u}_1, .., \mathbf{u}_N]$. The encoding function $\mathcal{E}_{\text{Trace}} : \mathbb{R}^{N \times (D_x + D_m + D_u)} \mapsto \mathbb{R}^{L \times D_z}$ maps the input to a fixed-size latent space state representation $\mathbf{Z}^t := \mathcal{E}(\mathbf{X}^t, \mathbf{M}, \mathbf{U}_\omega) \in \mathbb{R}^{L \times D_z}$, realized by cross-attention (Vaswani, 2017) between the input tensor $\in \mathbb{R}^{N \times (D_x + D_m + D_u)}$ of $\mathcal{E}_{\text{Trace}}$, keys and values, and a fixed number of L learned query vectors $\in \mathbb{R}^{D_z}$, see Fig. 2 (left).

Decoder. The aim of the decoder $\mathcal{D}$ is to retrieve the system state information $\mathbf{X}^t$ and $\mathbf{M}$, corresponding to the latent state representation $\mathbf{Z}^t$ and $\mathbf{U}_\omega$, i.e., encoded entity identifier embeddings. This is realized by a decoding function $\mathcal{D}_{\text{Trace}} : \mathbb{R}^{L \times D_z} \times \mathbb{R}^{D_u} \mapsto \mathbb{R}^{(D_x + D_m)}$, which is applied to each $\mathbf{u}_n$ available from $\mathbf{U}_\omega$, i.e., $(\mathbf{x}_n^t, \mathbf{m}_n) = \mathcal{D}_{\text{Trace}}(\mathbf{Z}^t, \mathbf{u}_n)$. As indicated in the left part of Fig. 2, also $\mathcal{D}_{\text{Trace}}$ is realized by cross-attention layers. The latent space information $\mathbf{Z}^t$ is input to both keys and values of the cross-attention mechanism, while the embedded identifier of e_n is input to the queries. The decoder makes use of a cross-attention mechanism using learned ID embeddings as query (Widrich et al., 2020; Locatello et al., 2020; Ramsauer et al., 2021), which could be seen as content-based retrieval system and an associative memory (Amari, 1972; Hopfield, 1982; Ramsauer et al., 2021).

Approximator. Finally, the approximator models the system's time evolution in latent space, i.e., predicts a series of future latent system states $\mathbf{Z}^{[T_o:\, T-1]} = [\mathbf{Z}^{T_o}, \mathbf{Z}^t, \ldots, \mathbf{Z}^{T-1}]$, given a series of initial system states, which are already embedded in latent space $\mathbf{Z}^{[0:\, T_o-1]} = [\mathbf{Z}^0, \mathbf{Z}^t, \ldots, \mathbf{Z}^{T_o-1}]$. Hence, the approximator is a function $\mathcal{A}\colon \mathbb{R}^{T_o \times L \times D_z} \mapsto \mathbb{R}^{(T - T_o) \times L \times D_z}$.

Given the analogy of predicting the time evolution of a dynamic system to the task of synthesizing videos, we realized $\mathcal{A}$ by a flow-based model. Specifically, we constructed it based on the stochastic interpolants framework (Albergo et al., 2023; Ma et al., 2024), see App. C.

We are interested in time-dependent processes, which interpolate between data $\mathbf{o}_1 \sim p_1$ from a target data distribution p_1 and noise $\epsilon \sim p_0 := \mathcal{N}(\mathbf{0}, \boldsymbol{I})$:

$$\mathbf{o}_\tau = \alpha_\tau \mathbf{o}_1 + \sigma_\tau \epsilon, \tag{5}$$

where $\tau \in [0, 1]$ is a diffusion time (to be distinguished from dynamic system times t). α_τ and σ_τ are differentiable functions in τ, which have to fulfill $\alpha_\tau^2 + \sigma_\tau^2 > 0 \; \forall \tau \in [0, 1]$, and, further $\alpha_1 = \sigma_0 = 0$, and, $\alpha_0 = \sigma_1 = 1$. The goal is to learn a parametric model $\boldsymbol{v}_\theta(\mathbf{o}, \tau)$, s.t., $\int_0^1 ||\mathbb{E}[\boldsymbol{v}_\theta(\mathbf{o}_\tau, \tau) - \dot{\alpha}_\tau \mathbf{o}_1 - \dot{\sigma}_\tau \epsilon||^2] \, d\tau$ is minimized. Within the stochastic interpolants framework, we identify $\mathbf{o}_1$ with a whole trajectory $\mathbf{Z} = \mathbf{Z}^{[0:\, T-1]} = [\mathbf{Z}^{[0:\, T_o-1]}, \mathbf{Z}^{[T_o:\, T-1]}] \in \mathbb{R}^{T \times L \times D_z}$. Since the generated trajectories should be conditioned on latent representations of initial time frames $\mathbf{Z}^{[0:\, T_o-1]}$, we extend $\boldsymbol{v}_\theta$ with a conditioning argument $\mathbf{C} \in \mathbb{R}^{T \times L \times D_z}$, making it effectively a conditional vector field $\boldsymbol{v}_\theta\colon \mathbb{R}^{T \times L \times D_z} \times [0, 1] \times \mathbb{R}^{T \times L \times D_z} \mapsto \mathbb{R}^{T \times L \times D_z}$. The tensor structure of $\mathbf{C}$ is the same as the one for $\mathbf{Z}$. For the first time steps, both tensors have equal values, i.e., $\mathbf{C}^{[0:\, T_o-1]} = \mathbf{Z}^{[0:\, T_o-1]}$. The remaining tensor entries $\mathbf{C}^{[T_o:\, T-1]}$ are filled up with mask tokens. (see. Fig. 3).

From a practical point of view, we did not make direct use of implementing $\boldsymbol{v}_\theta$, but instead reparameterize a data prediction model $\boldsymbol{o}_\theta\colon \mathbb{R}^{T \times L \times D_z} \times [0, 1] \times \mathbb{R}^{T \times L \times D_z} \mapsto \mathbb{R}^{T \times L \times D_z}$ with the aim to have small differences $\|\boldsymbol{o}_\theta(\mathbf{o}_\tau, \tau, \mathbf{C}) - \mathbf{o}_1\|^2$, i.e., $\boldsymbol{o}_\theta$ should directly learn to predict $\mathbf{Z}$ instead of

learning a velocity vector field $\boldsymbol{v}_\theta$ directly. To be able to use $\boldsymbol{o}_\theta$, we employ a reparametrization according to Kingma & Gao (2024) (details in App. C.2). The backbone of $\boldsymbol{o}_\theta$ consists of alternating attention blocks that operate across the latent dimension L and the time dimension T (see. App. B).

3.4 Training procedure

The training process is structured into two stages as in latent diffusion models (Rombach et al., 2022): **First stage:** Training the encoder $\mathcal{E}$ and decoder $\mathcal{D}$, to reconstruct entities from latent space using the assigned identifiers (see. Fig. 2). **Second stage:** Train latent approximator $\mathcal{A}$ by mapping system states to latent space using the frozen encoder $\mathcal{E}$ (details in App. E.2).

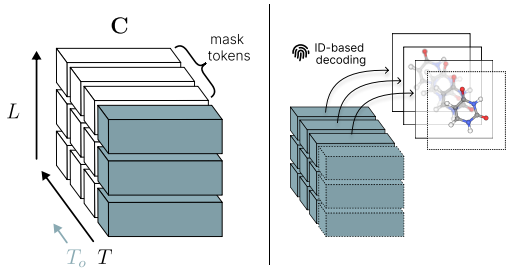

Figure 3: **Left:** The latent model receives conditioning information, through a configuration of known tokens (observed timesteps) and mask tokens (the tokens to be predicted). The example illustrates conditioning on a single timeframe to predict three future timeframes. **Right:** ID-based decoding, the predicted latent vectors are decoded using assigned IDs. For example, when IDs are assigned to individual atoms, we can track the atom's position across the predicted timesteps.

4 Experiments

Our evaluation of LAM-SLIDE , focused on three key aspects: (i) **Robust generalization in diverse domains**. We examine LAM-SLIDE 's generalization in different data domains in relation to other methods, for which we utilized tracking data from human motion behavior and data from *molecular dynamics* (MD) simulations. (ii) **Temporal adaptability**. We evaluated temporal adaptability through various conditioning/prediction horizons, considering single/multi-frame conditioning and short/long-term forecasts; (iii) **Computational efficiency and scalability**. Finally, we assessed LAM-SLIDE 's inference time, and performance in relation model size. The subsequent sections detail our key findings, while comprehensive implementation details, additional results are in App. E.

Metrics. We utilized the Average Discrepancy Error (ADE) and the Final Discrepancy Error (FDE), defined as $\text{ADE}(\mathbf{X}, \hat{\mathbf{X}}) = \frac{1}{(T-T_o)N} \sum_{t=T_o}^{T-1} \sum_{i=1}^{N} \|\mathbf{X}_i^t - \hat{\mathbf{X}}_i^t\|_2$, $\text{FDE}(\mathbf{X}, \hat{\mathbf{X}}) = \frac{1}{N} \sum_{i=1}^{N} \|\mathbf{X}_i^{T-1} - \hat{\mathbf{X}}_i^{T-1}\|_2$, capturing model performance across predicted future time steps and the model performance specifically for the last predicted frame, respectively. These metrics represent well-established evaluation criteria in forecasting (Xu et al., 2023; 2022). For the MD experiments including proteins (tetrapeptides), we used *Jensen-Shannon divergence* (JSD), evaluating the distribution of torsion angles, considering both, backbone (BB) and side chain (SC) angles. In order to capture long temporal behavior, we used *Time-lagged Independent Component Analysis* (TICA) (Pérez-Hernández et al., 2013), focusing on the slowest components TIC 0 and TIC 1. To investigate metastable state transitions we make use of *Markov State Models* (MSMs) (Prinz et al., 2011; Noé et al., 2013). Finally, for inference time and scalability we assessed the *number of function evaluations* (NFEs), and report performance of our method for different parameter sizes.

4.1 Pedestrian movement

Experimental setup. For human motion behavior, we first considered the ETH-UCY dataset (Pellegrini et al., 2009; Lerner et al., 2007), which provides pedestrian movement behavior, over five different scenes: ETH, Hotel, Univ, Zara1 and Zara2. We used the same setup as Han et al. (2024); Xu et al. (2023; 2022), in which the methods obtain the first 8 frames as input and have to predict the next 12 frames. We report the minADE/ minFDE, computed across 20 sampled trajectories and compare LAM-SLIDE to eight state-of-the-art generative methods covering different model categories, including: GANs : SGAN, SoPhie; VAEs: PECNet, Traj+ +, BiTrap, SVAE; diffusion models: MID and GeoTDM; and a Linear baseline. The baseline methods predominantly target pedestrian trajectory prediction, with GeoTDM and Linear being the exceptions.

Results. As shown in Tab. 1, our model performs competitively across all five scenes, achieving lower minFDE for Zara1 and Hotel scene, and lower minADE on the ETH scene. Notably, in contrast to compared baselines, we did not create additional features like velocity and acceleration or imply any kind of connectivity between entities. In terms of computational efficiency, LAM-SLIDE required only 10 NFE using Euler integration, compared to 100 NFE in GeoTDM.

Table 1: **Method comparison at pedestrian motion forecasting.** Methods have to predict locations of pedestrian given 8 time frames as input. The first column provides the method name and the further columns different scenes. In the cells the metrics minADE/minFDE on pedestrian movement forecasting across 20 frames are reported.

	ETH	Hotel	Univ	Zara1	Zara2	Average
Linear[a]	1.07/2.28	0.31/0.61	0.52/1.16	0.42/0.95	0.32/0.72	0.53/1.14
SGAN Gupta et al. (2018)[a]	0.64/1.09	0.46/0.98	0.56/1.18	0.33/0.67	0.31/0.64	0.46/0.91
SoPhie Sadeghian et al. (2019)[a]	0.70/1.43	0.76/1.67	0.54/1.24	0.30/0.63	0.38/0.78	0.54/1.15
PECNet Mangalam et al. (2020)[a]	0.54/0.87	0.18/0.24	0.35/0.60	0.22/0.39	0.17/0.30	0.29/0.48
Traj++ Salzmann et al. (2020)[a]	0.54/0.94	0.16/0.28	0.28/0.55	0.21/0.42	0.16/0.32	0.27/0.50
BiTraP Yao et al. (2021)[a]	0.56/0.98	0.17/0.28	0.25/0.47	0.23/0.45	0.16/0.33	0.27/0.50
MID Gu et al. (2022)[a]	0.50/0.76	0.16/0.24	0.28/0.49	0.25/0.41	0.19/0.35	0.27/0.45
SVAE Xu et al. (2022)[a]	0.47/0.76	0.14/0.22	0.25/0.47	**0.20**/0.37	**0.14/0.28**	**0.24**/0.42
GeoTDM (Han et al., 2024)[a]	0.46/**0.64**	**0.13**/0.21	**0.24/0.45**	0.21/0.39	0.16/0.30	**0.24/0.40**
LAM-SLIDE (ours)	**0.45**/0.75	**0.13/0.19**	0.26/0.47	0.21/**0.35**	0.17 / 0.30	**0.24**/ 0.41

[a] Results from Han et al. (2024).

4.2 PLAYER MOVEMENT IN BASKETBALL

Experimental setup. We evaluate basketball player movement using the SportVU NBA dataset (Yue et al., 2014), which captures player positions during the 2015-2016 season. Each recorded frame included ten player positions (5 for each team) and the ball position, and two different scenarios are considered. The evaluation procedure by, which we used, provides 8 frames as input conditioning and the consecutive 12 frames for prediction. We report minADE/minFDE metrics, computed across 20 sampled trajectories, and consider only the player position in the metric calculation (Xu et al., 2022). We compared against VAE-based methods (Traj++, BiTrap, SGNet-ED, SVAE) and a linear baseline.

Table 2: **Method comparison at forecasting player positions in basketball games.** Compared methods have to predict player positions for 12 frames and are given the initial 8 frames as input. The first column provides the method name, the consecutive columns the performance at Rebounding and Scoring scenes in terms of the metrics minADE/minFDE.

	Rebounding	Scoring
Linear[a]	2.14/5.09	2.07/4.81
Traj++ (Salzmann et al., 2020)[a]	0.98/1.93	0.73/1.46
BiTraP (Yao et al., 2021)[a]	0.83/1.72	0.74/1.49
SGNet-ED (Wang et al., 2022)[a]	0.78/1.55	0.68/1.30
SVAE (Xu et al., 2022)[a]	**0.72/1.37**	**0.64**/1.17
LAM-SLIDE (ours)	0.79/1.42	**0.64/1.09**

[a] Results from Xu et al. (2022).

Results. As illustrated in Tab. 2, our model shows robust performance across both scenarios, Rebounding and Scoring. For the Scoring scenario, LAM-SLIDE achieves parity with SocialVAE (Gupta et al., 2018) in terms of minADE and surpassing the performance in terms of minFDE. In the Rebounding scenario, we observed comparable but slightly lower performance of LAM-SLIDE compared to SocialVAE.

4.3 SMALL MOLECULES

Experimental setup. To assess the performance of LAM-SLIDE against several state-of-the-art methods, we also used the MD17 Chmiela et al. (2017) dataset, which contains the simulated molecular dynamics trajectories of 8 small molecules. The size of those molecules ranges from 9 atomes (Ethanol and Malonaldehyde) to 21 atoms(Aspirin). In line with Han et al. (2024), we used 10 conditioning frames and 20 frames for prediction and report ADE/FDE averaged over $K = 5$ runs. We compared against six g-equivariant GNN based methods: TFN, RF, SE(3)-Tr., EGNN, EqMotion, GeoTDM, and a non-equivariant method: SVAE.

Results. The results in Tab. 3 show the performance on the MD17 benchmark. LAM-SLIDE achieves the lowest ADE/FDE of all methods and for all molecules. These results are particu-

Table 3: **Method comparison at forecasting MD trajectories of small molecules**. Compared methods have to predict atom positions of 20 frames, conditioned on 10 input frames. Results in terms of ADE/FDE, averaged over 5 sampled trajectories.

	Aspirin		Benzene		Ethanol		Malonaldehyde		Naphthalene		Salicylic		Toluene		Uracil	
	ADE	FDE	ADE	FDE	ADE	FDE	ADE	FDE	ADE	FDE	ADE	FDE	ADE	FDE	ADE	FDE
RF (Köhler et al., 2019)[a]	0.303	0.442	0.120	0.194	0.374	0.515	0.297	0.454	0.168	0.185	0.261	0.343	0.199	0.249	0.239	0.272
TFN (Thomas et al., 2018)[a]	0.133	0.268	0.024	0.049	0.201	0.414	0.184	0.386	0.072	0.098	0.115	0.223	0.090	0.150	0.090	0.159
SE(3)-Tr. (Fuchs et al., 2020)[a]	0.294	0.556	0.027	0.056	0.188	0.359	0.214	0.456	0.069	0.103	0.189	0.312	0.108	0.184	0.107	0.196
EGNN (Satorras et al., 2021)[a]	0.267	0.564	0.024	0.042	0.268	0.401	0.393	0.958	0.095	0.133	0.159	0.348	0.207	0.294	0.154	0.282
EqMotion (Xu et al., 2023)[a]	0.185	0.246	0.029	0.043	0.152	0.247	0.155	0.249	0.073	0.092	0.110	0.151	0.097	0.129	0.088	0.116
SVAE (Xu et al., 2022)[a]	0.301	0.428	0.114	0.133	0.387	0.505	0.287	0.430	0.124	0.135	0.122	0.142	0.145	0.171	0.145	0.156
GeoTDM (Han et al., 2024) [a]	0.107	0.193	0.023	0.039	0.115	0.209	0.107	0.176	0.064	0.087	0.083	0.120	0.083	0.121	0.074	0.099
LAM-SLIDE (ours)	**0.059**	**0.098**	**0.021**	**0.032**	**0.087**	**0.167**	**0.073**	**0.124**	**0.037**	**0.058**	**0.047**	**0.074**	**0.045**	**0.075**	**0.050**	**0.074**

[a] Results from Han et al. (2024).

larly remarkable considering that: (1) our model operates without incorporating molecular bond information, and (2) it surpasses the performance of all equivariant baselines, an inductive bias we intentionally omitted in LAM-SLIDE . For additional information see App. E.2.

4.4 TETRAPEPTIDES (4AA)

Experimental setup. For long prediction horizons, we utilized a tetrapeptide dataset Jing et al. (2024), which contains explicit-solvent MD trajectories . We used a single conditioning frame to predict 10,000 consecutive frames. The predictions are structured as a sequence of ten cascading 1,000-step rollouts, where each subsequent rollout is conditioned on the final frame of the previous. Note that, in contrast to the MD17 dataset, the methods predict trajectories of *unseen* molecules. We compare LAM-SLIDE to the recently proposed method MDGen (Jing et al., 2024) which is geared towards protein MD simulation, and to a replicate of the ground truth MD simulation as a baseline.

Table 4: **Method comparison for predicting MD trajectories of tetrapeptides**. The first column denotes the method. The following columns denote the JSD between distributions of *torsion angles* (backbone (BB), side-chain (SC), and all), the TICA, and the MSM metric. LAM-SLIDE performs best with respect to 4 out of 6 metrics.

	Torsions			TICA		MSM	Time
	BB	SC	All	0	0,1 joint		
100ns[a]	.103	.055	.076	.201	.268	.208	$\sim$ 3h
MDGen[a]	.130	**.093**	**.109**	.230	.316	.235	$\sim$ 60s
LAM-SLIDE	**.128**	.122	.125	**.227**	**.315**	**.224**	$\sim$ 53s

[a] Results from Jing et al. (2024).

Results. Tab. 4 shows performance metrics of the methods (see above; for details on those metrics, Sec. E.5). Fig. 5 shows the distribution of backbone torsions angles, and the free energy surfaces of the first two TICA components, for ground truth vs simulated trajectories. LAM-SLIDE performs competitively with the current state-of-the- art method MDGen with respect to torsion angles, which is a notable achievement given that MDGen operates in torsion space only. With respect to the TICA and MSM metrics, LAM-SLIDE even outperforms MDGen.

4.5 ANALYSIS OF SCALING BEHAVIOR

We performed experiments in which we analyzed the dependence of the performance of LAM-SLIDE on the number of parameters. The results indicate that the performance consistently increases with parameter count (see App. F).

5 CONCLUSION

We have introduced LAM-SLIDE , which combines the advantages of GNNs and latent space models. Its novel *entity structure preservation module* uses ID embeddings to retrieve entity positions in latent space. Across diverse domains, LAM-SLIDE matches or exceeds specialized methods, and offers efficiency, cross-task information-sharing, and promising scalability. Its minimal reliance on prior knowledge makes LAM-SLIDE suitable for many tasks, e.g., pedestrian or molecule trajectory prediction, suggesting its potential as a foundational system dynamics architecture (see App. D).

ACKNOWLEDGMENTS

The ELLIS Unit Linz, the LIT AI Lab, the Institute for Machine Learning, are supported by the Federal State Upper Austria. We thank the projects Medical Cognitive Computing Center (MC3), INCONTROL-RL (FFG-881064), PRIMAL (FFG-873979), S3AI (FFG-872172), EPILEPSIA (FFG-892171), AIRI FG 9-N (FWF-36284, FWF-36235), AI4GreenHeatingGrids (FFG- 899943), INTEGRATE (FFG-892418), ELISE (H2020-ICT-2019-3 ID: 951847), Stars4Waters (HORIZON-CL6-2021-CLIMATE-01-01). We thank Audi.JKU Deep Learning Center, TGW LOGISTICS GROUP GMBH, Silicon Austria Labs (SAL), FILL Gesellschaft mbH, Google, ZF Friedrichshafen AG, Robert Bosch GmbH, UCB Biopharma SRL, Merck Healthcare KGaA, Verbund AG, GLS (Univ. Waterloo), Software Competence Center Hagenberg GmbH, Borealis AG, TÜV Austria, dSPACE, TRUMPF and the NVIDIA Corporation.

REFERENCES

Josh Abramson, Jonas Adler, Jack Dunger, Richard Evans, Tim Green, Alexander Pritzel, Olaf Ronneberger, Lindsay Willmore, Andrew J Ballard, Joshua Bambrick, et al. Accurate structure prediction of biomolecular interactions with alphafold 3. *Nature*, pp. 1–3, 2024.

Michael S Albergo and Eric Vanden-Eijnden. Building normalizing flows with stochastic interpolants. *arXiv preprint arXiv:2209.15571*, 2022.

MS Albergo, NM Boffi, and E Vanden-Eijnden. Stochastic interpolants: A unifying framework for flows and diffusions, 2023. *ArXiv preprint ArXiv230308797*, 2023.

Benedikt Alkin, Andreas Fürst, Simon Lucas Schmid, Lukas Gruber, Markus Holzleitner, and Johannes Brandstetter. Universal physics transformers: A framework for efficiently scaling neural operators. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024a.

Benedikt Alkin, Tobias Kronlachner, Samuele Papa, Stefan Pirker, Thomas Lichtenegger, and Johannes Brandstetter. Neuraldem-real-time simulation of industrial particulate flows. *arXiv preprint arXiv:2411.09678*, 2024b.

Shun-Ishi Amari. Learning patterns and pattern sequences by self-organizing nets of threshold elements. *IEEE Transactions on computers*, 100(11):1197–1206, 1972.

L. Arnold. *Random Dynamical Systems*. Monographs in Mathematics. Springer, 1998. ISBN 9783540637585.

Jimmy Lei Ba. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

Black Forest Labs. Flux. `https://github.com/black-forest-labs/flux`, 2023.

Andreas Blattmann, Robin Rombach, Huan Ling, Tim Dockhorn, Seung Wook Kim, Sanja Fidler, and Karsten Kreis. Align your latents: High-resolution video synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 22563–22575, 2023.

Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

D.A. Case, H.M. Aktulga, K. Belfon, I.Y. Ben-Shalom, J.T. Berryman, S.R. Brozell, D.S. Cerutti, T.E. Cheatham, III, G.A. Cisneros, V.W.D. Cruzeiro, T.A. Darden, N. Forouzesh, M. Ghazimirsaeed, G. Giambaşu, T. Giese, M.K. Gilson, H. Gohlke, A.W. Goetz, J. Harris, Z. Huang, S. Izadi, S.A. Izmailov, K. Kasavajhala, M.C. Kaymak, A. Kovalenko, T. Kurtzman, T.S. Lee, P. Li, Z. Li, C. Lin, J. Liu, T. Luchko, R. Luo, M. Machado, M. Manathunga, K.M. Merz, Y. Miao, O. Mikhailovskii, G. Monard, H. Nguyen, K.A. O'Hearn, A. Onufriev, F. Pan, S. Pantano, A. Rahnamoun, D.R. Roe, A. Roitberg, C. Sagui, S. Schott-Verdugo, A. Shajan, J. Shen, C.L. Simmerling, N.R. Skrynnikov, J. Smith, J. Swails, R.C. Walker, J. Wang, J. Wang, X. Wu, Y. Wu, Y. Xiong, Y. Xue, D.M. York, C. Zhao, Q. Zhu, and P.A. Kollman. Amber 2024, 2024.

Ricky T. Q. Chen. torchdiffeq, 2018. URL `https://github.com/rtqichen/torchdiffeq`.

Stefan Chmiela, Alexandre Tkatchenko, Huziel E Sauceda, Igor Poltavsky, Kristof T Schütt, and Klaus-Robert Müller. Machine learning of accurate energy-conserving molecular force fields. *Science advances*, 3(5):e1603015, 2017.

Allan dos Santos Costa, Ilan Mitnikov, Franco Pellegrini, Ameya Daigavane, Mario Geiger, Zhonglin Cao, Karsten Kreis, Tess Smidt, Emine Kucukbenli, and Joseph Jacobson. Equijump: Protein dynamics simulation via so (3)-equivariant stochastic interpolants. *arXiv preprint arXiv:2410.09667*, 2024.

Mostafa Dehghani, Josip Djolonga, Basil Mustafa, Piotr Padlewski, Jonathan Heek, Justin Gilmer, Andreas Peter Steiner, Mathilde Caron, Robert Geirhos, Ibrahim Alabdulmohsin, et al. Scaling vision transformers to 22 billion parameters. In *International Conference on Machine Learning*, pp. 7480–7512. PMLR, 2023.

Jacob Devlin. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Alexey Dosovitskiy. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, et al. Scaling rectified flow transformers for high-resolution image synthesis, 2024. *URL https://arxiv. org/abs/2403.03206*, 2, 2024.

GW Ford, M Kac, and P Mazur. Statistical mechanics of assemblies of coupled oscillators. *Journal of Mathematical Physics*, 6(4):504–515, 1965.

Fabian B Fuchs, Daniel E Worrall, Volker Fischer, and Max Welling. Se (3)-transformers: 3d roto-translation equivariant attention networks. *arXiv preprint arXiv:2006.10503*, 2020.

Everette S Gardner Jr. Exponential smoothing: The state of the art. *Journal of forecasting*, 4(1): 1–28, 1985.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pp. 1263–1272. PMLR, 2017.

Tianpei Gu, Guangyi Chen, Junlong Li, Chunze Lin, Yongming Rao, Jie Zhou, and Jiwen Lu. Stochastic trajectory prediction via motion indeterminacy diffusion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 17113–17122, 2022.

Agrim Gupta, Justin Johnson, Li Fei-Fei, Silvio Savarese, and Alexandre Alahi. Social gan: Socially acceptable trajectories with generative adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2255–2264, 2018.

Jiaqi Han, Minkai Xu, Aaron Lou, Haotian Ye, and Stefano Ermon. Geometric trajectory diffusion models. *arXiv preprint arXiv:2410.13027*, 2024.

Jonathan Ho, William Chan, Chitwan Saharia, Jay Whang, Ruiqi Gao, Alexey Gritsenko, Diederik P Kingma, Ben Poole, Mohammad Norouzi, David J Fleet, et al. Imagen video: High definition video generation with diffusion models. *arXiv preprint arXiv:2210.02303*, 2022.

Håkon Hoel and Anders Szepessy. Classical langevin dynamics derived from quantum mechanics. *arXiv preprint arXiv:1906.09858*, 2019.

John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

Guillaume Huguet, James Vuckovic, Kilian Fatras, Eric Thibodeau-Laufer, Pablo Lemos, Riashat Islam, Cheng-Hao Liu, Jarrid Rector-Brooks, Tara Akhound-Sadegh, Michael Bronstein, et al. Sequence-augmented se (3)-flow matching for conditional protein backbone generation. *arXiv preprint arXiv:2405.20313*, 2024.

Brooke E Husic and Vijay S Pande. Markov state models: From an art to a science. *Journal of the American Chemical Society*, 140(7):2386–2396, 2018.

Andrew Jaegle, Sebastian Borgeaud, Jean-Baptiste Alayrac, Carl Doersch, Catalin Ionescu, David Ding, Skanda Koppula, Daniel Zoran, Andrew Brock, Evan Shelhamer, et al. Perceiver io: A general architecture for structured inputs & outputs. *arXiv preprint arXiv:2107.14795*, 2021.

Bowen Jing, Hannes Stärk, Tommi Jaakkola, and Bonnie Berger. Generative modeling of molecular dynamics trajectories. *arXiv preprint arXiv:2409.17808*, 2024.

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *nature*, 596(7873):583–589, 2021.

Martin Karplus and Gregory A Petsko. Molecular dynamics simulations in biology. *Nature*, 347 (6294):631–639, 1990.

Diederik Kingma and Ruiqi Gao. Understanding diffusion objectives as the elbo with simple data augmentation. *Advances in Neural Information Processing Systems*, 36, 2024.

Diederik P Kingma. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. Neural relational inference for interacting systems. In *International conference on machine learning*, pp. 2688–2697. PMLR, 2018.

Jonas Köhler, Leon Klein, and Frank Noé. Equivariant flows: sampling configurations for multi-body systems with symmetric energies. *arXiv preprint arXiv:1910.00753*, 2019.

Alon Lerner, Yiorgos Chrysanthou, and Dani Lischinski. Crowds by example. In *Computer graphics forum*, volume 26, pp. 655–664. Wiley Online Library, 2007.

Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.

Jiawei Liu, Cheng Yang, Zhiyuan Lu, Junze Chen, Yibo Li, Mengmei Zhang, Ting Bai, Yuan Fang, Lichao Sun, Philip S Yu, et al. Towards graph foundation models: A survey and beyond. *arXiv preprint arXiv:2310.11829*, 2023.

Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. *arXiv preprint arXiv:2209.03003*, 2022.

Francesco Locatello, Dirk Weissenborn, Thomas Unterthiner, Aravindh Mahendran, Georg Heigold, Jakob Uszkoreit, Alexey Dosovitskiy, and Thomas Kipf. Object-centric learning with slot attention. *Advances in neural information processing systems*, 33:11525–11538, 2020.

Ilya Loshchilov, Frank Hutter, et al. Fixing weight decay regularization in adam. *arXiv preprint arXiv:1711.05101*, 5, 2017.

Nanye Ma, Mark Goldstein, Michael S Albergo, Nicholas M Boffi, Eric Vanden-Eijnden, and Saining Xie. Sit: Exploring flow and diffusion-based generative models with scalable interpolant transformers. *arXiv preprint arXiv:2401.08740*, 2024.

Karttikeya Mangalam, Harshayu Girase, Shreyas Agarwal, Kuan-Hui Lee, Ehsan Adeli, Jitendra Malik, and Adrien Gaidon. It is not the journey but the destination: Endpoint conditioned trajectory prediction. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16*, pp. 759–776. Springer, 2020.

Haitao Mao, Zhikai Chen, Wenzhuo Tang, Jianan Zhao, Yao Ma, Tong Zhao, Neil Shah, Mikhail Galkin, and Jiliang Tang. Position: Graph foundation models are already here. In *Forty-first International Conference on Machine Learning*.

Andreas Mayr, Sebastian Lehner, Arno Mayrhofer, Christoph Kloss, Sepp Hochreiter, and Johannes Brandstetter. Boundary graph neural networks for 3d simulations. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(8):9099–9107, Jun. 2023. doi: 10.1609/aaai.v37i8.26092.

Alessio Micheli. Neural network for graphs: A contextual constructive approach. *IEEE Transactions on Neural Networks*, 20(3):498–511, 2009. doi: 10.1109/TNN.2008.2010350.

Frank Noé, Hao Wu, Jan-Hendrik Prinz, and Nuria Plattner. Projected and hidden markov models for calculating kinetics and metastable states of complex molecules. *The Journal of chemical physics*, 139(18), 2013.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 4195–4205, 2023.

Stefano Pellegrini, Andreas Ess, Konrad Schindler, and Luc Van Gool. You'll never walk alone: Modeling social behavior for multi-target tracking. In *2009 IEEE 12th international conference on computer vision*, pp. 261–268. IEEE, 2009.

Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Guillermo Pérez-Hernández, Fabian Paul, Toni Giorgino, Gianni De Fabritiis, and Frank Noé. Identification of slow molecular order parameters for markov model construction. *The Journal of chemical physics*, 139(1), 2013.

Adam Polyak, Amit Zohar, Andrew Brown, Andros Tjandra, Animesh Sinha, Ann Lee, Apoorv Vyas, Bowen Shi, Chih-Yao Ma, Ching-Yao Chuang, et al. Movie gen: A cast of media foundation models. *arXiv preprint arXiv:2410.13720*, 2024.

Jay W. Ponder and David A. Case. Force fields for protein simulations. *Advances in protein chemistry*, 66:27–85, 2003. ISSN 0065-3233. doi: 10.1016/S0065-3233(03)66002-X.

Ilan Price, Alvaro Sanchez-Gonzalez, Ferran Alet, Tom R Andersson, Andrew El-Kadi, Dominic Masters, Timo Ewalds, Jacklynn Stott, Shakir Mohamed, Peter Battaglia, et al. Probabilistic weather forecasting with machine learning. *Nature*, 637(8044):84–90, 2025.

Jan-Hendrik Prinz, Hao Wu, Marco Sarich, Bettina Keller, Martin Senne, Martin Held, John D Chodera, Christof Schütte, and Frank Noé. Markov models of molecular kinetics: Generation and validation. *The Journal of chemical physics*, 134(17), 2011.

Hubert Ramsauer, Bernhard Schäfl, Johannes Lehner, Philipp Seidl, Michael Widrich, Lukas Gruber, Markus Holzleitner, Thomas Adler, David Kreil, Michael K Kopp, Klambauer G, and S Hochreiter. Hopfield networks is all you need. *International Conference on Learning Representations*, 2021.

Alex Rogozhnikov. Einops: Clear and reliable tensor manipulations with einstein-like notation. In *International Conference on Learning Representations*, 2021.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10684–10695, 2022.

Amir Sadeghian, Vineet Kosaraju, Ali Sadeghian, Noriaki Hirose, Hamid Rezatofighi, and Silvio Savarese. Sophie: An attentive gan for predicting paths compliant to social and physical constraints. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 1349–1358, 2019.

Tim Salzmann, Boris Ivanovic, Punarjay Chakravarty, and Marco Pavone. Trajectron++: Dynamically-feasible trajectory forecasting with heterogeneous data. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XVIII 16*, pp. 683–700. Springer, 2020.

Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks. In *International conference on machine learning*, pp. 8459–8468. PMLR, 2020.

Victor Garcia Satorras, Emiel Hoogeboom, and Max Welling. E(n) equivariant graph neural networks. *arXiv preprint arXiv:2102.09844*, 2021.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009. doi: 10.1109/TNN.2008.2005605.

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

Jacob Seidman, Georgios Kissas, Paris Perdikaris, and George J Pappas. Nomad: Nonlinear manifold decoders for operator learning. *Advances in Neural Information Processing Systems*, 35: 5601–5613, 2022.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pp. 2256–2265. PMLR, 2015.

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

A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

Apoorv Vyas, Bowen Shi, Matthew Le, Andros Tjandra, Yi-Chiao Wu, Baishan Guo, Jiemin Zhang, Xinyue Zhang, Robert Adkins, William Ngan, et al. Audiobox: Unified audio generation with natural language prompts. *arXiv preprint arXiv:2312.15821*, 2023.

Chuhua Wang, Yuchen Wang, Mingze Xu, and David J Crandall. Stepwise goal-driven networks for trajectory prediction. *IEEE Robotics and Automation Letters*, 7(2):2716–2723, 2022.

Michael Widrich, Bernhard Schäfl, Milena Pavlović, Hubert Ramsauer, Lukas Gruber, Markus Holzleitner, Johannes Brandstetter, Geir Kjetil Sandve, Victor Greiff, Sepp Hochreiter, et al. Modern hopfield networks and attention for immune repertoire classification. *Advances in neural information processing systems*, 33:18832–18845, 2020.

Chenxin Xu, Robby T Tan, Yuhong Tan, Siheng Chen, Yu Guang Wang, Xinchao Wang, and Yanfeng Wang. Eqmotion: Equivariant multi-agent motion prediction with invariant interaction reasoning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1410–1420, 2023.

Pei Xu, Jean-Bernard Hayet, and Ioannis Karamouzas. Socialvae: Human trajectory prediction using timewise latents. In *European Conference on Computer Vision*, pp. 511–528. Springer, 2022.

Yu Yao, Ella Atkins, Matthew Johnson-Roberson, Ram Vasudevan, and Xiaoxiao Du. Bitrap: Bi-directional pedestrian trajectory prediction with multi-modal goal estimation. *IEEE Robotics and Automation Letters*, 6(2):1463–1470, 2021.

Ziyang Yu, Wenbing Huang, and Yang Liu. Force-guided bridge matching for full-atom time-coarsened dynamics of peptides. *arXiv preprint arXiv:2408.15126*, 2024.

Yisong Yue, Patrick Lucey, Peter Carr, Alina Bialkowski, and Iain Matthews. Learning fine-grained spatial models for dynamic sports play prediction. In *IEEE International Conference on Data Mining*, pp. 670–679, 2014.

Biao Zhang and Peter Wonka. Lagem: A large geometry model for 3d representation learning and diffusion. *arXiv preprint arXiv:2410.01295*, 2024.

Biao Zhang, Jiapeng Tang, Matthias Niessner, and Peter Wonka. 3dshape2vecset: A 3d shape representation for neural fields and generative diffusion models. *ACM Transactions on Graphics (TOG)*, 42(4):1–16, 2023.

Robert Zwanzig. Nonlinear generalized langevin equations. *Journal of Statistical Physics*, 9(3): 215–220, 1973.

Appendix

Table of Contents

A Notation **16**

B Details on the LAM-SLIDE Model Architecture **17**
B.1 Encoder and Decoder Functions 17
B.2 Latent Flow Model Architecture 17

C Additional Information on Stochastic Interpolants **19**
C.1 Interpolants 19
C.2 Parametrizations 19

D Related Work **20**
D.1 Molecular Dynamics (MD) 20
D.2 Relationship of LAM-SLIDE to Graph Foundation Models 21

E Experimental Details **22**
E.1 Employed Loss Functions 22
E.2 Implementation details 22
E.3 Hyperparameters 23
E.4 Employed Datasets 23
E.5 Evaluation Details 23
E.6 Computational Resources 24

F Additional Results and Scaling Behavior **25**

A NOTATION

Table 5: Overview of used symbols and notations

Definition	Symbol/Notation	Type
continuous time	$\hat{t}$	$\mathbb{R}$
overall number of (sampled) time steps	T	$\mathbb{N}$
number of observed time steps (when predicting later ones)	T_o	$0 \,..\, T{-}1$
time index for sequences of time steps	t	$0 \,..\, T{-}1$
system state space	$\mathcal{S}$	application-dependent set, to be further defined
system state	$\boldsymbol{s}$	$\mathcal{S}$
randomness	ω	$\Omega \equiv$ pool of randomness
entity	e	symbolic
number of entities	N	$\mathbb{N}$
entity index	i, n	$1 \,..\, N$
spatial entity dimensionality	D_x	$\mathbb{N}$
entity feature dimensionality	D_m	$\mathbb{N}$
entity location (coordinate)	$\mathbf{x}$	$\mathbb{R}^{D_x}$
entity properties (entity features)	$\mathbf{m}$	$\mathbb{R}^{D_m}$
identifier representation dimensionality	D_u	$\mathbb{N}$
number of latent vectors used	L	$\mathbb{N}$
latent vector dimensionality	D_z	$\mathbb{N}$
trajectory of a system (locations of entities over time)	$\mathbf{X}$	$\mathbb{R}^{T_o \times N \times D_x}$
entity locations at t	$\mathbf{X}^t$	$\mathbb{R}^{N \times D_x}$
entity i of trajectory at t	$\mathbf{X}_i^t$	$\mathbb{R}^{D_x}$
trajectory in latent space	$\mathbf{Z}$	$\mathbb{R}^{T_o \times L \times D_z}$
latent system state at t	$\mathbf{Z}^t$	$\mathbb{R}^{L \times D_z}$
time invariant features of entities	$\mathbf{M}$	$\mathbb{R}^{N \times D_m}$
matrix of identifier embeddings	$\mathbf{U}_\omega$	$\mathbb{R}^{N \times D_u}$
projection matrices	$\mathbf{Q}, \mathbf{K}, \mathbf{V}$	not specified; depends on number of heads etc.
encoder	$\mathcal{E}(.)$	$\mathbb{R}^{N \times (D_u + D_x + D_m)} \mapsto \mathbb{R}^{L \times D_z}$
decoder	$\mathcal{D}(.)$	$\mathbb{R}^{L \times D_z} \times \mathbb{R}^{N \times D_u} \mapsto \mathbb{R}^{N \times (D_x + D_m)}$
approximator (time dynamics model)	$\mathcal{A}(.)$	$\mathbb{R}^{T \times L \times D_z} \mapsto \mathbb{R}^{T \times L \times D_z}$
loss function	$\mathcal{L}(.,.)$	var.
time parameter of the flow-based model	τ	$[0, 1]$
noise distribution	$\mathbf{o}_0$	$\mathbb{R}^{T \times L \times D_z}$
de-noised de-masked trajectory	$\mathbf{o}_1 = \mathbf{Z}$	$\mathbb{R}^{T \times L \times D_z}$
flow-based model "velocity prediction" (neural net)	$\boldsymbol{v}_\theta(\mathbf{o}_\tau, \tau)$	$\mathbb{R}^{T \times L \times D_z} \times \mathbb{R} \mapsto \mathbb{R}^{T \times L \times D_z}$
flow-based model "data prediction" (neural net)	$\boldsymbol{o}_\theta(\mathbf{o}_\tau, \tau)$	$\mathbb{R}^{T \times L \times D_z} \times \mathbb{R} \mapsto \mathbb{R}^{T \times L \times D_z}$
neural network parameters	θ	undef.

B DETAILS ON THE LAM-SLIDE MODEL ARCHITECTURE

B.1 ENCODER AND DECODER FUNCTIONS

We provide pseudocode of the forward passes for encoding ($\mathcal{E}_{\text{Trace}}$) to and decoding ($\mathcal{D}_{\text{Trace}}$) from the latent system space of LAM-SLIDE in algorithm 1 and algorithm 2 respectively. In general, encoder and decoder blocks follow the standard Transformer architecture (Vaswani, 2017) with feedforward and normalization layers. To simplify the explanation, we omitted additional implementation details here and refer readers to our provided source code.

Algorithm 1 (Cross-Attention) Encoder Function $\mathcal{E}_{\text{Trace}}$

Input: input data $\mathbf{XMU} = [\mathbf{X}, \mathbf{M}, \mathbf{U}_\omega] \in \mathbb{R}^{N \times (D_x + D_m + D_u)}$
Output: latent system state $\mathbf{Z} \in \mathbb{R}^{L \times D_z}$
Additional Internal Encoder Weights: learned latent queries $\mathbf{Z}_{\text{init}} \in \mathbb{R}^{L \times D_z}$
$\mathbf{K} = \text{Linear}(\mathbf{XMU})$
$\mathbf{V} = \text{Linear}(\mathbf{XMU})$
$\mathbf{Q} = \text{Linear}(\mathbf{Z}_{\text{init}})$
return $\text{LayerNorm}(\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}))$ // without affine transformation

Algorithm 2 (Cross-Attention) Decoder Function $\mathcal{D}_{\text{Trace}}$

Input: latent system representation $\mathbf{Z} \in \mathbb{R}^{L \times D_z}$, entity representation $\mathbf{u} \in \mathbb{R}^{D_u}$ from $\mathbf{U}_\omega \in \mathbb{R}^{N \times D_u}$
Output: $[\mathbf{x}, \mathbf{m}] \in \mathbb{R}^{D_x + D_m}$
$\mathbf{Z} = \text{LayerNorm}(\mathbf{Z})$ // without affine transformation
$\mathbf{K} = \text{Linear}(\mathbf{Z})$
$\mathbf{V} = \text{Linear}(\mathbf{Z})$
$\mathbf{q} = \text{Linear}(\mathbf{u})$
return $\text{Attention}([\mathbf{q}], \mathbf{K}, \mathbf{V})$

For the decoding functionality presented in Algorithm 2, we made use of multiple specific decoder blocks depending on the actual task (e.g., for the molecules dataset, we used one decoder block for atom positions and one decoder block for atom types).

B.2 LATENT FLOW MODEL ARCHITECTURE

We provide pseudocode of the data prediction network $\boldsymbol{o}_\theta$ forward pass in algorithm 3. The latent layer functionality is given by algorithm 4. The architecture of the latent layers (i.e., our flow model) is based on Dehghani et al. (2023), with the additional usage of adaptive layer norm (adaLN) (Perez et al., 2018) as also used for Diffusion Transformers (Peebles & Xie, 2023). The exact implementation is based on ParalellMLP block codes from Black Forest Labs (2023), which are used along the latent dimension as well as along the temporal dimension (see Fig. 4).

Algorithm 3 Latent Flow Model $\boldsymbol{o}_\theta$ (data prediction network)

Input: noise-interpolated data $\mathbf{o}_{\text{inter}} \in \mathbb{R}^{T \times L \times D_z}$, diffusion time τ used for interpolation, conditioning $\mathbf{C} \in \mathbb{R}^{T \times L \times D_z}$, conditioning mask $\mathbf{B} \in \{0, 1\}^{T \times L \times D_z}$
Output: prediction of original data (not interpolated with noise): $\mathbf{o} \in \mathbb{R}^{T \times L \times D_z}$
$\boldsymbol{\tau} \leftarrow \text{Embed}(\tau)$;
$\mathbf{o} = \text{Linear}(\mathbf{o}_{\text{inter}}) + \text{Linear}(\mathbf{C}) + \text{Embed}(\mathbf{B})$
for $i = 1$ **to** *num_layers* **do**
  $\mathbf{o} = \text{LatentLayer}(\mathbf{o}, \boldsymbol{\tau})$
end for
$\alpha, \beta, \gamma = \text{Linear}(\text{SiLU}(\boldsymbol{\tau}))$
return $\mathbf{o} + \gamma \odot \text{MLP}(\alpha \odot \text{LayerNorm}(\mathbf{o}) + \beta)$

Algorithm 4 LatentLayer

Input: $\mathbf{o} \in \mathbb{R}^{T \times L \times C}$, diffusion time embedding $\boldsymbol{\tau}$
Output: updated $\mathbf{o} \in \mathbb{R}^{T \times L \times C}$
$\mathbf{o} \mathrel{+}= \text{ParallelMLPAttentionWithRoPE}(\mathbf{o}, \boldsymbol{\tau}, \text{dim} = 0)$
$\mathbf{o} \mathrel{+}= \text{ParallelMLPAttentionWithRoPE}(\mathbf{o}, \boldsymbol{\tau}, \text{dim} = 1)$
return $\mathbf{o}$

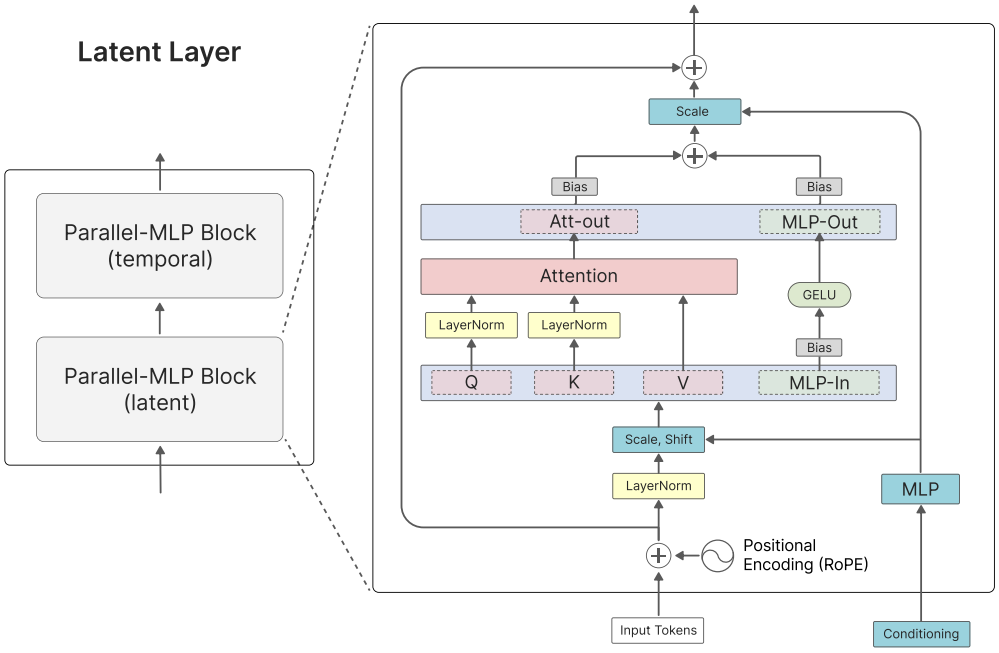

Figure 4: **Left**: LatentLayer of our method, consisting of a latent and a temporal ParallelMLP block. **Right**: Zoomed in view of the ParalellMLP block

Using `einops` Rogozhnikov (2021) notation, the latent layer in Fig. 4 can be expressed as:

$$\mathbf{o}' \leftarrow \texttt{rearrange}(\mathbf{o},\ \texttt{(B L) T D} \rightarrow \texttt{(B T) L D})$$
$$\mathbf{o}' \leftarrow l^i_\psi(\mathbf{o}', t)$$
$$\mathbf{o}' \leftarrow \texttt{rearrange}(\mathbf{o}',\ \texttt{(B T) L D} \rightarrow \texttt{(B L) T D})$$
$$\mathbf{o}' \leftarrow l^i_\phi(\mathbf{o}', t)$$

with parameters sets ψ and ϕ, where for the latent block the time dimension gets absorbed into the batch dimension and for the temporal block the latent dimension gets absorbed into the batch dimension.

C ADDITIONAL INFORMATION ON STOCHASTIC INTERPOLANTS

General Interpolants. The stochastic interpolants framework (Albergo et al., 2023; Albergo & Vanden-Eijnden, 2022; Ma et al., 2024) is defined without reference to an forward SDE, which allows a lot of flexibility, any choice of α_t an σ_t, satisfying the following conditions is possible:

1. $\alpha_\tau^2 + \sigma_\tau^2 > 0$;
2. α_τ and σ_τ are differentiable for all $\tau \in [0, 1]$
3. $\alpha_0 = \sigma_1 = 0$ and $\alpha_1 = \sigma_0 = 1$

According to Ma et al. (2024) the stochastic interpolant framework decouples the formulation of the stochastic process specified by eq. (5) from the forward SDE and therefore allows for more flexibility in the choosing α_t and σ_t. App. C.1 details α_t and σ_t for a linear and a generalized variance-preserving (GVP) interpolant. App. C.2 show equivalences of different parameterizations for a score network, which are according to Ma et al. (2024) and Kingma & Gao (2024).

C.1 INTERPOLANTS

$$\text{Linear:} \quad \alpha_\tau = \tau, \qquad \sigma_\tau = 1 - \tau, \tag{6}$$

$$\text{GVP:} \quad \alpha_\tau = \sin(\frac{1}{2}\pi\tau), \qquad \sigma_\tau = \cos(\frac{1}{2}\pi\tau), \tag{7}$$

C.2 PARAMETRIZATIONS

$$\hat{\boldsymbol{s}}_\theta(\mathbf{o};\tau) = -\nabla_{\mathbf{x}} E_\theta(\mathbf{o},\tau) \qquad \text{(With the gradient of an energy-based model)} \tag{8}$$

$$= -\hat{\boldsymbol{\epsilon}}_\theta(\mathbf{o};\tau)/\sigma_\tau \qquad \text{(With a noise prediction model)} \tag{9}$$

$$= -\sigma_\tau^{-2}(\mathbf{o} - \alpha_\tau \hat{\boldsymbol{o}}_\theta(\mathbf{o};\tau)) \qquad \text{(With a data prediction model)} \tag{10}$$

$$= \sigma_\tau^{-1} \frac{\alpha_\tau \hat{\boldsymbol{v}}_\theta(\mathbf{o},\tau) - \dot{\alpha}_\tau \mathbf{o}}{\dot{\alpha}_\tau \sigma_\tau - \alpha_\tau \dot{\sigma}_\tau} \qquad \text{(With a velocity prediction model)} \tag{11}$$

Note: In our work we learn a data prediction model $\hat{\boldsymbol{o}}_\theta(\mathbf{o};\tau)$, which can be expressed in terms of velocity:

$$\hat{\boldsymbol{v}}_\theta(\mathbf{o},\tau) = \hat{\boldsymbol{s}}_\theta(\mathbf{o};\tau)\left(\frac{\dot{\alpha}_\tau \sigma_\tau^2}{\alpha_\tau} - \sigma_\tau \dot{\sigma}_\tau\right) + \frac{\dot{\alpha}_\tau}{\alpha_\tau}\mathbf{o} \tag{12}$$

D RELATED WORK

D.1 MOLECULAR DYNAMICS (MD)

The most fundamental concepts nowadays to describe the dynamics of molecules are given by the laws of quantum mechanics. The Schrödinger equation is a partial differential equation, that gives the evolution of the complex-valued wave function ψ over time t: $i\hbar\frac{\partial\psi}{\partial t} = \hat{H}(t)\psi$. Here i is the imaginary unit with $i^2 = -1$, $\hbar$ is reduced Planck constant, and, $\hat{H}(t)$ is the Hamiltonian operator at time t, which is applied to a function ψ and maps to another function. It determines how a quantum system evolves with time and its eigenvalues correspond to measurable energy values of the quantum system. The solution to Schrödinger's equation in the many-body case (particles $1, \ldots, N$) is the wave function $\psi(\mathbf{x}_1, \ldots, \mathbf{x}_N, t) : \bigtimes_{i=1}^{N} \mathbb{R}^3 \times \mathbb{R} \to \mathbb{C}$ which we abbreviate as $\psi(\{\mathbf{x}\}, t)$. It's the square modulus $|\psi(\{\mathbf{x}\}, t)|^2 = \psi^*(\{\mathbf{x}\}, t)\psi(\{\mathbf{x}\}, t)$ is usually interpreted as a probability density to measure the positions $\mathbf{x}_1, \ldots, \mathbf{x}_N$ at time t, whereby the normalization condition $\int \ldots \int |\psi(\{\mathbf{x}\}, t)|^2 d\mathbf{x}_1 \ldots d\mathbf{x}_N = 1$ holds for the wave function ψ.

Analytic solutions of ψ for specific operators $\hat{H(t)}$ are hardly known and are only available for simple systems like free particles or hydrogen atoms. In contrast to that are proteins with many thousands of atoms. However, already for much smaller quantum systems approximations are needed. A famous example is the Born–Oppenheimer approximation, where the wave function of the multi-body system is decomposed into parts for heavier atom nuclei and the light-weight electrons, which usually move much faster. In this case, one obtains a Schrödinger equation for electron movement and another Schrödinger equation for nuclei movement. A much faster option than solving a second Schrödinger equation for the motion of the nuclei is to use the laws from classical Newtonian dynamics. The solution of the first Schrödinger equation defines an energy potential, which can be utilized to obtain forces $\mathbf{F}_i$ on the nuclei and to update nuclei positions according to Newton's equation of motion: $\mathbf{F}_i = m_i\, \ddot{\mathbf{q}}_i(t)$ (with m_i being the mass of particle i and $\mathbf{q}_i(t)$ describing the motion trajectory of particle i over time t).

Additional complexity in studying molecule dynamics is introduced by environmental conditions surrounding molecules. Maybe the most important is temperature. For bio-molecules it is often of interest to assume that they are dissolved in water. To model temperature, a usual strategy is to assume a system of coupled harmonic oscillators to model a heat bath, from which Langevin dynamics can be derived (Ford et al., 1965; Zwanzig, 1973). The investigation of the relationship between quantum-mechanical modeling of heat baths and Langevin dynamics still seems to be a current research topic, where there there are different aspects like the coupling of the oscillators or Markovian properties when stochastic forces are introduced. For instance, Hoel & Szepessy (2019), studies how canonical quantum observables are approximated by molecular dynamics. This includes the definition of density operators, which behave according to the quantum Liouville-von Neumann equation.

The forces in molecules are usually given as the negative derivative of the (potential) energy: $\mathbf{F}_i = -\nabla E$. In the context of molecules, E is usually assumed to be defined by a force field, which is a parameterized sum of intra- and intermolecular interaction terms. An example is the Amber force field (Ponder & Case, 2003; Case et al., 2024):

$$E = \sum_{\text{bonds } r} k_b(r - r_0)^2 + \sum_{\text{angles } \theta} k_\theta(\theta - \theta_0)^2 + \sum_{\text{dihedrals } \phi} V_n(1 + cos(n\phi - \gamma)) + \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \left(\frac{A_{ij}}{R_{ij}^{12}} - \frac{B_{ij}}{R_{ij}^{6}} + \frac{q_i q_j}{\epsilon R_{ij}} \right) \tag{13}$$

Here $k_b, r_0, k_\theta, \theta_0, V_n, \gamma, A_{ij}, B_{ij}, \epsilon, q_i, q_j$ serve as force field parameters, which are found either empirically or which might be inspired by theory.

Newton's equations of motions for all particles under consideration form a system of ordinary differential equations (ODEs), to which different numeric integration schemes like Euler, Leapfrog, or, Verlet can be applied to obtain particle position trajectories for given initial positions and initial velocities. In case temperature is included, the resulting Langevin equations form a system of

stochastic differential equations (SDEs), and Langevin integrators can be used. It should be mentioned, that it is often necessary to use very small integration timesteps to avoid large approximation errors. This, however, increases the time needed to find new stable molecular configurations.

D.2 Relationship of LaM-SLidE to Graph Foundation Models

From our perspective, LaM-SLidE bears a relationship to graph foundation models (GFMs; Liu et al., 2023; Mao et al.). Bommasani et al. (2021) consider foundation models to be *trained on broad data at scale* and to be *adaptable to a wide range of downstream tasks*. Mao et al. argue, that graphs are more diverse than natural language or images, and therefore there are quite unique challenges for GFMs. Especially they mention that *none of the current GFM have the capability to transfer across all graph tasks and datasets from all domains*. It is for sure true that LaM-SLidE is not a GFM in this sense. However, it might be debatable whether LaM-SLidE might serve as a domain- or task-specfic GFM. While we mainly focused on a trajectory prediction task and are from that point of view task-specific, we observed that our trained models can generalize across different molecules or differently taken scenes, which might seem quite remarkable given that it is common practice to train specific trajectory prediction models for single molecules or single scenes. Nevertheless, it was not our aim in this research to provide a GFM, since we believe that this would require more investigation into further domains and could also require, for instance, checking whether emergent abilities might arise with larger models and more training data (Liu et al., 2023).

E EXPERIMENTAL DETAILS

E.1 EMPLOYED LOSS FUNCTIONS

This section defines the losses, which we use throughout training:

Position loss.

$$\mathcal{L}_{\text{pos}}(\mathbf{X}^t, \hat{\mathbf{X}}^t) = \frac{1}{N} \sum_{i=1}^{N} ||\mathbf{X}_i^t - \hat{\mathbf{X}}_i^t||_2^2 \tag{14}$$

Inter-distance loss.

$$\mathcal{L}_{\text{int}}(\mathbf{X}^t, \hat{\mathbf{X}}^t) = \frac{1}{N^2} \sum_{i=1}^{N} \sum_{j=1}^{N} (D_{ij}(\mathbf{X}^t) - D_{ij}(\hat{\mathbf{X}}^t))^2 \tag{15}$$

with

$$D_{ij}(\mathbf{X}^t) = ||\mathbf{X}_i^t - \mathbf{X}_j^t||_2 \tag{16}$$

For our experiments on the tetrapeptide dataset, we employ two well-established loss functions tailored to better capture the unique geometric constraints of proteins, which are outlined below.

Frame Loss. The frame loss is based on representing all residue atoms within local reference frames (Abramson et al., 2024, Algorithm 29). This approach ensures the invariance of the loss to the protein's overall orientation. The frame loss is denoted as D_{frame} and is backpropagated through our network.

Torsion Loss. Inspired by Jumper et al. (2021), LAM-SLIDE uses a torsion loss $\mathcal{L}_{\text{tors}}$, which is backpropagated to the coordinates.

E.2 IMPLEMENTATION DETAILS

Training procedure. (i) **First stage**. In the first stage we train the encoding and decoding functions $\mathcal{E}_{\text{Trace}}$ and $\mathcal{D}_{\text{Trace}}$ in an auto-encoding fashion, i.e., we optimize the reconstruction of the original system state representation from its latent representation well. For discrete features (e.g., atom type, residue type) we tend to use a cross-entropy loss, whereas for continuous features we use a regression loss (e.g., position, distance). The loss functions for each individual task are summarized in App. E.3. Since our method is not equivariant w.r.t. translations or rotations, we apply random rotations and translations to the input positions. Notably, also the entity identifier assignment is random. (ii) **Second stage**. In the second stage, we freeze the encoder and train the approximator to model the temporal dynamics via the encoded latent vectors. To learn a consistent behavior over time, we pass $\mathbf{U}_\omega$ from the encoder $\mathcal{E}$ to the decoder $\mathcal{D}$. To avoid high variance latent spaces we used layer-normalization (Ba, 2016) (see. App. E.2).

Identifiers. For the embedding of the identifiers we used a `torch.nn.Embedding` (Paszke et al., 2019) layer, where we assign a random subset of the possible embeddings to the entities in each training step.

Latent space regularization. To avoid high variance latent spaces, Rombach et al. (2022) relies on KL-reg., imposing a small KL-penalty towards a standard normal on the latent space, as used in VAE (Kingma, 2013). Recent work (Zhang & Wonka, 2024) has shown that layer normalization (Ba, 2016) can achieve similar regulatory effects without requiring an additional loss term and simplifying training procedure, we adapt this approach in our method (see left part of Fig. 2).

Latent Model. For the latent Flow Model we additionally apply auxiliary losses for the individual tasks, as shown in App. E.3. Were we decode the the predicted latents and back-propagate through the frozen decoder to the latent model.

MD17. We train a single model on all molecules – a feat that is structurally encouraged by the design of LAM-SLIDE . For ablation, we also trained GeoTDM Han et al. (2024) on all molecules and evaluated the performance on each one of them ("all→each" in the Tab. 10). Interestingly, we also observe consistent improvements in the GeoTDM performance; however, GeoTDM's performance does not reach the one of LAM-SLIDE . Furthermore, Whereas GeoTDM requires 1000 integration steps in their diffusion process, our model achieves the reported performance with 10 Euler integration steps, demonstrating strongly improved computational efficiency.

Tetrapeptides. For the experiments on tetrapeptides in Section 4.4, we employ the Atom14 representation as used in AlphaFold (Abramson et al., 2024). In this representation, each entity corresponds to one amino acid of the tetrapeptide, where multiple atomic positions are encoded into a single vector of dimension $D_x = 3 \times 14$. Masked atomic positions are excluded from gradient computation during model updates. This representation is computational more efficient.

E.3 HYPERPARAMETERS

Tab. 6, Tab. 7, Tab. 9 and Tab. 8 show the hyperparameters for the individual tasks, loss functions are as defined above (App. E.1). For all trained models we used the AdamW (Kingma, 2014; Loshchilov et al., 2017) optimizer and use EMA (Gardner Jr, 1985) in each update step with a decay parameter of $\beta = 0.999$.

E.4 EMPLOYED DATASETS

Pedestrian Movement. The pedestrian movement dataset is accessible at `http://vision.imar.ro/human3.6m/description.php`, with data processing based on `https://github.com/MediaBrain-SJTU/EqMotion`.

Basketball Player Movement. The dataset, along with its predefined splits, is available at `https://github.com/xupei0610/SocialVAE`. Data processing is provided in our source code.

Small Molecules (MD17). The MD17 dataset is available at `http://www.sgdml.org/#datasets`. Preprocessing and dataset splits follow Han et al. (2024) and can be accessed through their GitHub repository at `https://github.com/hanjq17/GeoTDM`. The dataset comprises, 5,000 training, 1000 validation and 1000 test trajectories for each molecule.

Tetrapeptides. The dataset, including the full simulation parameters for ground truth simulations, is sourced from Jing et al. (2024) and is publicly available in their GitHub repository at `https://github.com/bjing2016/mdgen`. The dataset comprises 3,109 training, 100 validation and 100 test peptides.

E.5 EVALUATION DETAILS

Tetrapeptides. Our analysis of the Tetrapeptide trajectories utilized PyEMM (Scherer et al., 2015) and followed the procedure as (Jing et al., 2024), incorporating both Time-lagged Independent Component Analysis (TICA) (Pérez-Hernández et al., 2013) and Markov State Models (MSM) (Husic & Pande, 2018).

E.6 COMPUTATIONAL RESOURCES

Our experiments were conducted using a system with 128 CPU cores and 2048GB of system memory. Model training was performed on 4 NVIDIA H200 GPUs, each equipped with 140GB of VRAM. In total, roughly 5000 GPU hours were used in this work.

F ADDITIONAL RESULTS AND SCALING BEHAVIOR

On the MD17 and the tetrapeptides dataset, we performed experiments in which we increased the number of parameters of LAM-SLIDE . On MD17, LAM-SLIDE uses 1.7M, 2.1M and 2.5M parameters, and for almost all molecules, the performance metrics ADE and FDE increase with parameter count (see Tab. 10). On the tetrapeptides dataset, LAM-SLIDE was trained with configurations of 4M, 7M, 11M, and 28M parameters. Again, all metrics consistently improve with parameter counts (see Tab. 11). Overall, LAM-SLIDE exhibits a favorable scaling behavior.

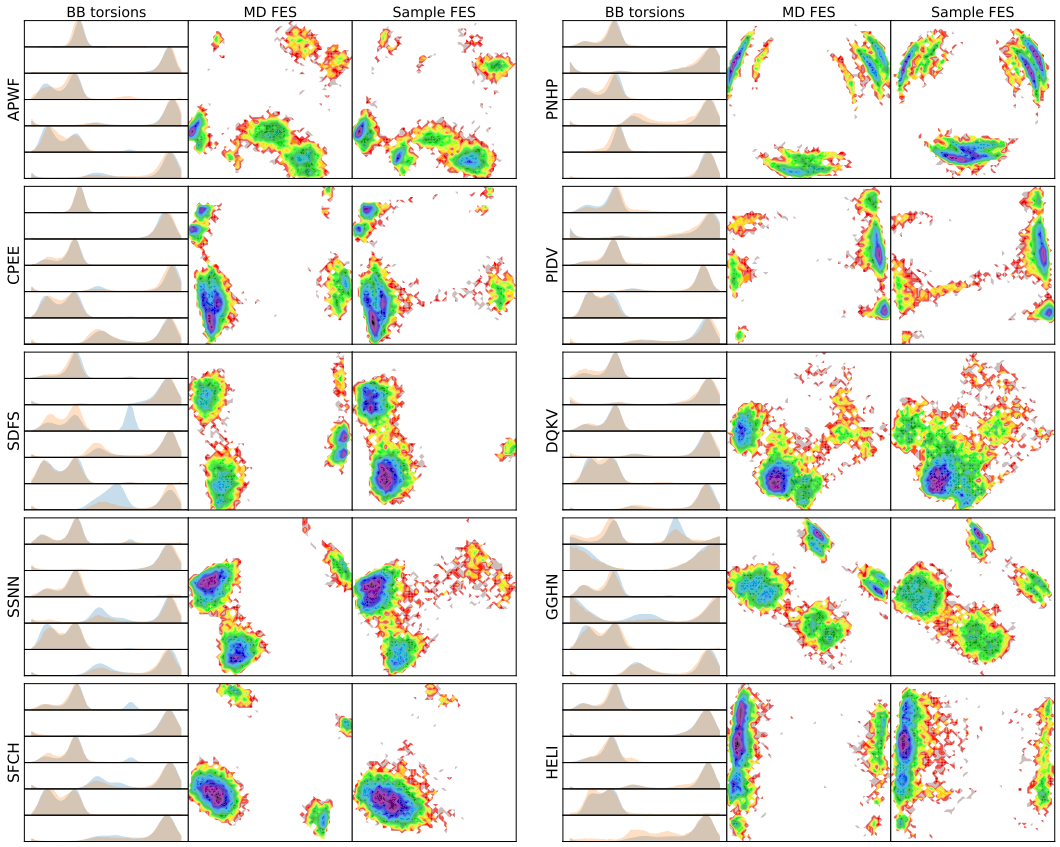

Figure 5: **Torsion angle distributions** of the six backbone torsion angles, comparing molecular dynamics (MD) trajectories (orange) and sampled trajectories (blue); and **Free energy surfaces** projected onto the top two time-lagged independent component analysis (TICA) components, computed from both backbone and sidechain torsion angles.

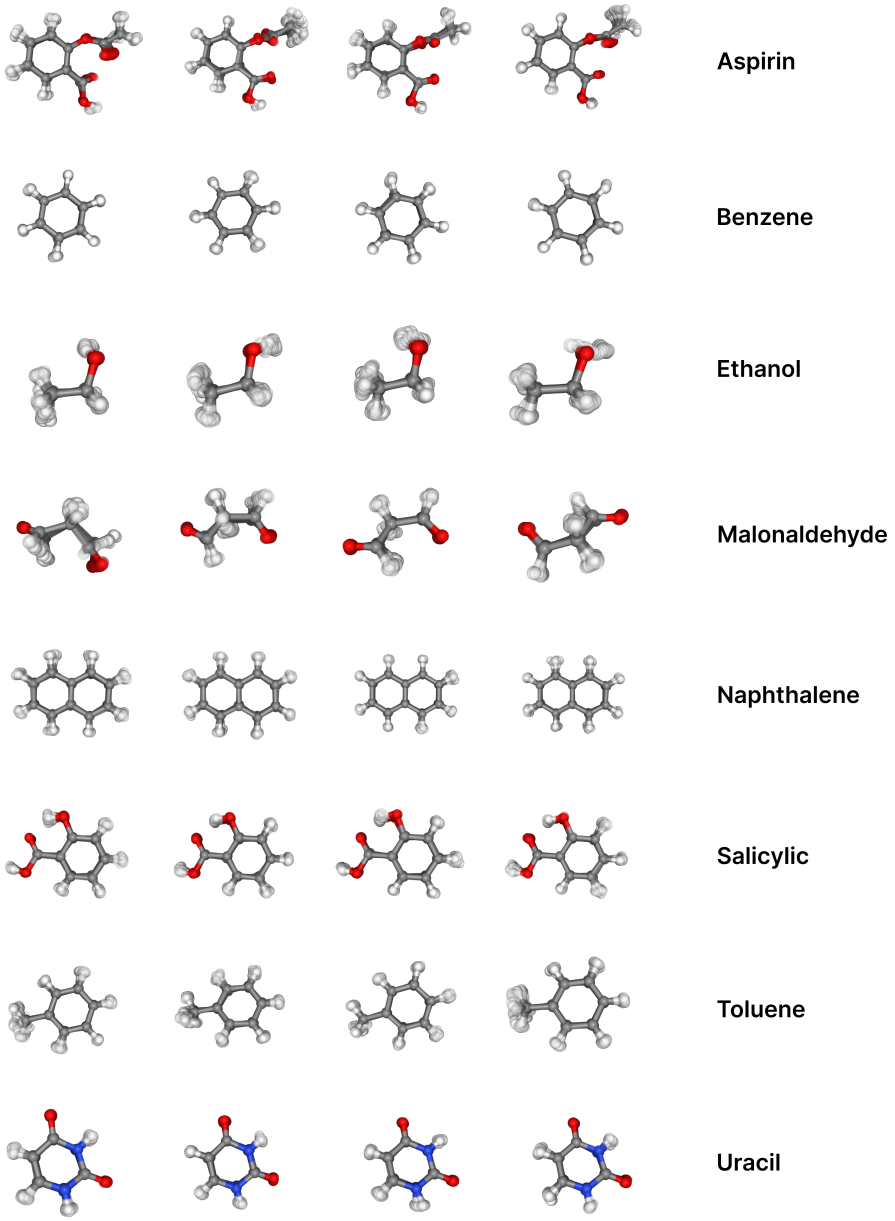

Figure 6: **Molecular dynamics trajectories from the MD17 dataset**, showing time-evolved structural predictions for each molecule. For every compound, we display four distinct trajectory predictions, with each prediction comprising 20 superimposed time frames to illustrate the range of conformational changes.

Table 6: Hyperparameter configuration for the pedestrian movement experiments (Section 4.1).

First Stage	
Network	
Encoder	
Number of latents L	2
Number of entity embeddings	8
Number of attention heads	2
Number of cross attention layers	1
Dimension latents D_z	32
Dimension entity embedding	128
Dimension attention head	16
Decoder	
Number of attention heads	2
Number of cross attention layers	1
Dimension attention head	16
Loss	**Weight**
$\mathcal{L}_{pos}(\mathbf{X}, \hat{\mathbf{X}})$	1
$\mathcal{L}_{int}(\mathbf{X}, \hat{\mathbf{X}})$	1
Training	
Learning rate	1e-4
Learning rate scheduler	CosineAnnealing(min_lr=1e-7)
Batch size	256
Epochs	3K
Precision	32-Full
Batch size	1024
Second Stage	
Setup	
Condition	8 Frames
Prediction	12 Frames
Network	
Hidden dimension	128
Number of Layers	6
Auxiliary - Loss	**Weight**
$\mathcal{L}_{pos}(\mathbf{X}, \hat{\mathbf{X}})$	0.25
$\mathcal{L}_{int}(\mathbf{X}, \hat{\mathbf{X}})$	0.25
Training	
Learning rate	1e-3
Batch size	64
Epochs	1K
Precision	BF16-Mixed
Inference	
Integrator	Euler
ODE steps	10

Table 7: Hyperparameter configuration for the basketball player movement experiments (Section 4.2).

First Stage	
Network	
Encoder	
Number of latents L	32
Number of entity embeddings	11
Number of attention heads	2
Number of cross attention layers	1
Dimension latents D_z	32
Dimension entity embedding	128
Dimension attention head	16
Decoder	
Latent dimension	32
Number of attention heads	8
Number of cross attention layers	1
Loss	**Weight**
$\mathcal{L}_{pos}(\mathbf{X}, \hat{\mathbf{X}})$	1
$\mathcal{L}_{int}(\mathbf{X}, \hat{\mathbf{X}})$	1
$\mathcal{L}_{CE}(\cdot, \cdot)$ − Group	0.01
$\mathcal{L}_{CE}(\cdot, \cdot)$ − Team	0.01
Training	
Learning rate	1e-4
Learning rate scheduler	CosineAnnealing(min_lr=1e-7)
Optimizer	AdamW
Batch size	16
Second Stage	
Setup	
Condition	8 Frames
Prediction	12 Frames
Network	
Hidden dimension H	128
Number of Layers	6
Auxiliary - Loss	**Weight**
$\mathcal{L}_{pos}(\mathbf{X}, \hat{\mathbf{X}})$	0.25
$\mathcal{L}_{int}(\mathbf{X}, \hat{\mathbf{X}})$	0.25
Training	
Learning rate	1e-3
Learning rate scheduler	CosineAnnealing(min_lr=1e-7)
Batch size	64
Epochs	500
Precision	BF16-Mixed
Inference	
Integrator	Euler
ODE steps	10

Table 8: Hyperparameter configuration for the small molecule (MD17) experiments (Section 4.3).

First Stage	
Network	
Encoder	
Number of latents L	32
Number of entity embeddings	8
Number of attention heads	2
Number of cross attention layers	1
Dimension latents D_z	32
Dimension entity embedding	128
Dimension attention head	16
Decoder	
Number of cross attention layers	1
Number of attention heads	2
Number of cross attention layers	16
Loss	**Weight**
$\mathcal{L}_{pos}(\mathbf{X}, \hat{\mathbf{X}})$	1
$\mathcal{L}_{int}(\mathbf{X}, \hat{\mathbf{X}})$	1
$\mathcal{L}_{CE}(\cdot, \cdot)$ – Atom type	1
Training	
Learning rate	1e-4
Batch size	256
Epochs	3K
Precision	32-Full
Second Stage	
Setup	
Condition	10 Frames
Prediction	20 Frames
Network	
Hidden dimension	128
Number of Layers	6
Auxiliary - Loss	**Weight**
$\mathcal{L}_{pos}(\mathbf{X}, \hat{\mathbf{X}})$	0.25
$\mathcal{L}_{int}(\mathbf{X}, \hat{\mathbf{X}})$	0.25
Training	
Learning rate	1e-3
Learning rate scheduler	CosineAnnealing(min_lr=1e-7)
Batch size	64
Epochs	2K
Precision	BF16-Mixed
Inference	
Integrator	Euler
ODE steps	10

Table 9: Hyperparameter configuration for the Tetrapeptides experiments (Section 4.4).

First Stage	
Network	
Encoder	
Number of latents L	5
Number of entity embeddings	8
Number of attention heads	2
Number of cross attention layers	1
Dimension latents D_z	96
Dimension entity embedding	128
Dimension attention head	16
Decoder	
Number of attention heads	2
Number of cross attention layers	1
Dimension attention head	16
Loss	**Weight**
$\mathcal{L}_{pos}(\mathbf{X}, \hat{\mathbf{X}})$	1
$\mathcal{L}_{int}(\mathbf{X}, \hat{\mathbf{X}})$	1
$\mathcal{L}_{frame}(\mathbf{X}, \hat{\mathbf{X}})$	1
$\mathcal{L}_{tors}(\mathbf{X}, \hat{\mathbf{X}})$	0.1
$\mathcal{L}_{CE}(\cdot, \cdot)$ − Residue type	0.001
Training	
Learning rate	1e-4
Batch size	16
Epochs	200K
Precision	32-Full
Second Stage	
Setup	
Condition	1 Frame
Prediction	10,000 Frames (10x rollouts)
Network	
Hidden dimension	384
Number of Layers	6
Auxiliary - Loss	**Weight**
$\mathcal{L}_{pos}(\mathbf{X}, \hat{\mathbf{X}})$	0.25
$\mathcal{L}_{int}(\mathbf{X}, \hat{\mathbf{X}})$	0.25
$\mathcal{L}_{frame}(\mathbf{X}, \hat{\mathbf{X}})$	0.25
Training	
Learning rate	1e-3
Optimizer	AdamW
Batch size	64
Epochs	1.5K
Precision	BF16-Mixed
Inference	
Integrator	Dopri5 (Chen, 2018)
ODE steps	adaptive

Table 10: **Method comparison at forecasting MD trajectories of small molecules**. Compared methods have to predict atom positions of 20 frames, conditioned on 10 input frames. Results in terms of ADE/FDE, averaged over 5 sampled trajectories.

	Aspirin		Benzene		Ethanol		Malonaldehyde		Naphthalene		Salicylic		Toluene		Uracil	
	ADE	FDE	ADE	FDE	ADE	FDE	ADE	FDE	ADE	FDE	ADE	FDE	ADE	FDE	ADE	FDE
RF Köhler et al. (2019)[a]	0.303	0.442	0.120	0.194	0.374	0.515	0.297	0.454	0.168	0.185	0.261	0.343	0.199	0.249	0.239	0.272
TFN Thomas et al. (2018)[a]	0.133	0.268	0.024	0.049	0.201	0.414	0.184	0.386	0.072	0.098	0.115	0.223	0.090	0.150	0.090	0.159
SE(3)-Tr. Fuchs et al. (2020)[a]	0.294	0.556	0.027	0.056	0.188	0.359	0.214	0.456	0.069	0.103	0.189	0.312	0.108	0.184	0.107	0.196
EGNN Satorras et al. (2021)[a]	0.267	0.564	0.024	0.042	0.268	0.401	0.393	0.958	0.095	0.133	0.159	0.348	0.207	0.294	0.154	0.282
EqMotion Xu et al. (2023)[a]	0.185	0.246	0.029	0.043	0.152	0.247	0.155	0.249	0.073	0.092	0.110	0.151	0.097	0.129	0.088	0.116
SVAE Xu et al. (2022)[a]	0.301	0.428	0.114	0.133	0.387	0.505	0.287	0.430	0.124	0.135	0.122	0.142	0.145	0.171	0.145	0.156
GeoTDM 1.9M[a]	0.107	0.193	0.023	0.039	0.115	0.209	0.107	0.176	0.064	0.087	0.083	0.120	0.083	0.121	0.074	0.099
GeoTDM 1.9M (all→each)	0.091	0.164	0.024	0.040	0.104	0.191	0.097	0.164	0.061	0.092	0.074	0.114	0.073	0.112	0.070	0.102
LAM-SLIDE 2.5M (ours)	**0.059**	**0.098**	**0.021**	**0.032**	**0.087**	**0.167**	**0.073**	**0.124**	**0.037**	**0.058**	**0.047**	**0.074**	**0.045**	**0.075**	**0.050**	**0.074**
LAM-SLIDE 2.1M (ours)	0.064	0.104	0.023	0.033	0.097	0.182	0.084	0.141	0.044	0.067	0.053	0.081	0.054	0.086	0.054	0.079
LAM-SLIDE 1.7M (ours)	0.074	0.117	0.025	0.037	0.110	0.195	0.097	0.159	0.053	0.074	0.063	0.091	0.064	0.094	0.064	0.089

[a] Results from Han et al. (2024).

Table 11: **Method comparison for predicting MD trajectories of tetrapeptides**. The first column denotes the method. The following columns denote the JSD between distributions of *torsion angles*, backbone (BB), side-chain (SC), and all, the TICA, and the MSM metric, for different model sizes.

	Torsions			TICA		MSM	Params	Time
	BB	SC	All	0	0,1 joint		(M)	
100 ns[a]	.103	.055	.076	.201	.268	.208		$\sim$ 3h
MDGen[a]	.130	**.093**	**.109**	.230	.316	.235	34	$\sim$ 60s
LAM-SLIDE	**.128**	0.122	0.125	**.227**	**.315**	**.224**	28	$\sim$ 53s
LAM-SLIDE	.152	.151	.152	.239	.331	.226	11	
LAM-SLIDE	.183	.191	.187	.26	.356	.235	7	
LAM-SLIDE	.284	.331	.311	.339	.461	.237	**4**	

[a] Results from Jing et al. (2024).