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

Anonymous Author(s)
Affiliation
Address
email

Abstract

Generative models are spearheading recent progress in deep learning, showcasing strong promise for trajectory sampling in dynamical systems as well. However, whereas 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, entity conservation, and the traceability of entities over time. Our approach, LAM-SLIDE (Latent Space Modeling of Spatial Dynamical Systems via Linked Entities), bridges the gap between: (1) keeping the traceability of individual entities in a latent system representation, and (2) leveraging the efficiency and scalability of recent advances in image and video generation, where pre-trained encoder and decoder enable generative modeling directly in latent space. The core idea of LAM-SLIDE is the introduction of identifier representations (IDs) that enable the retrieval of entity properties and entity composition from latent system representations, thus fostering traceability. Experimentally, across different domains, we show that LAM-SLIDE performs favorably in terms of speed, accuracy, and generalizability. Code is available at `https://anonymous.4open.science/r/lam-slide-B38B`.

1 Introduction

Understanding dynamical systems represents a fundamental challenge across numerous scientific and engineering domains [44, 43, 67]. In this work, we address *spatial* dynamical systems, characterized by scenes of distinguishable entities at defined spatial coordinates. Modeling temporal trajectories of such entities quickly becomes challenging, especially when stochasticity is involved. A prime example is molecular dynamics [44].

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 [75, 59, 32, 11], 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 [48], especially for problems with a large number of indistinguishable entities, [e.g., 73, 58]. Recently, GNNs have been integrated into generative modeling frameworks to effectively capture the behavior of stochastic systems [87, 22].

Despite their widespread use in modeling spatial trajectories, GNNs hardly follow recent trends in latent space modeling, where unified representations [42] together with universality and scalability of transformer blocks [82] offer simple application across datasets and tasks, a behavior commonly observed in computer vision and language processing [24, 25]. Notably, recent breakthroughs in image and video generation can be accounted to latent space generative modeling [36, 15]. In such

Submitted to 39th Conference on Neural Information Processing Systems (NeurIPS 2025). Do not distribute.

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:

> Can we leverage recent techniques from **generative latent space modeling** to boost the **modeling of stochastic trajectories of dynamical systems** with varying number of entities?

Recently, it has been shown [5] 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 dynamics, where understanding the dynamics of individual atoms is essential.

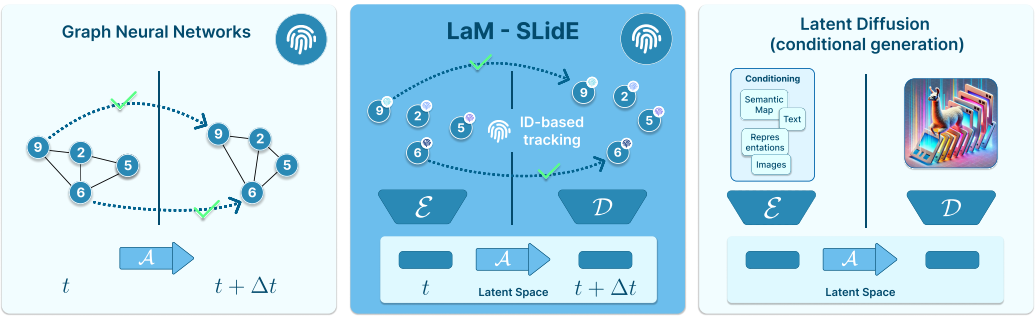

Figure 1: Overview of our approach. **Left:** Conventional graph neural networks (GNNs) model time-evolving systems (e.g., molecular dynamics) by representing entities as nodes and iteratively updating node embeddings and positions to capture system dynamics across timesteps. **Right:** Latent diffusion models employ an encoder-decoder architecture to compress input data into a lower-dimensional latent space where generative modeling is performed. Latent diffusion models, frequently enhanced with conditional information like text, excel at generative tasks, but due to the fixed input/output structure, are not directly adaptable to physical systems with varying number of entities. **Middle:** Our proposed approach LAM-SLIDE bridges these paradigms by: (1) introducing identifiers that allow traceability of individual entities, and (2) leveraging a latent system representation.

In order to leverage a latent systems representation, we need to be able to trace individual entities of the system. The core idea of our approach LAM-SLIDE is the introduction of *identifiers* (IDs) that allow for retrieval of entity properties, e.g. entity coordinates, from the latent system representation. Consequently, we can train generative models, like flow-matching [50, 53, 2], purely in latent space, where pre-trained decoder blocks map the generated representations back to the physics domain. An overview is given in Fig. 1. For more background information, please refer to App. A.

2 LaM - SLidE

We introduce an *identifier pool* and an *identifier assignment function* which allow us to effectively map and retrieve entities to and from a latent system representation. 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.

2.1 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 denote the set of entities as $E = \{e_1, \ldots, e_n\}$. We consider states $\mathbf{s}^t$ at discretized timepoints t. Analogously, we use $\mathbf{x}_i^t$, $\mathbf{m}_i^t$ to describe coordinates and properties at time t, respectively. 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

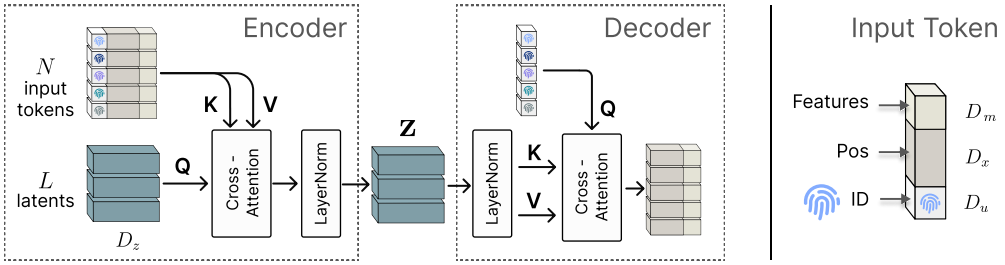

Figure 2: Architecture of our encoder-decoder structure (First Stage): **Left:** The encoder maps N input tokens to a latent system representation by cross-attending to L learned latent query tokens. The decoder reconstructs the input data from the latent representation using the assigned IDs. **Right:** Structure of the input token, consisting of an ID, spatial information and features (see also App. Fig. 3).

properties are conserved over time, i.e., $\mathbf{M}^t = \mathbf{M}^1$, 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 $t \in 1 \,..\, T$ to a tensor $\mathbf{X} \in \mathbb{R}^{T \times N \times D_x}$, which describes a whole sampled coordinate trajectory of a system with T time points and N entities. An example for such trajectories from dynamical systems are molecular dynamics trajectories (e.g. App. Fig. 9). Notation is summarized in App. B.

Prediction task. We predict a trajectory of entity coordinates $\mathbf{X}^{[T_o+1\,:\,T]} = [\mathbf{X}^{T_o+1}, \ldots, \mathbf{X}^t, \ldots, \mathbf{X}^T] \in \mathbb{R}^{(T-T_o) \times N \times D_x}$, given a short (observed) initial trajectory $\mathbf{X}^{[1\,:\,T_o]} = [\mathbf{X}^1, \ldots, \mathbf{X}^t, \ldots, \mathbf{X}^{T_o}] \in \mathbb{R}^{T_o \times N \times D_x}$ together with general (time-invariant) entity properties $\mathbf{M}$. Here, T_o denotes the length of the observed trajectory and $T_f = T - T_o$ the prediction horizon.

2.2 Entity Structure Preservation

We aim to preserve the integrity of individual entities in a latent system representation. More specifically, we aim to preserve both the number of entities as well as their structure. For example, in the case of molecules, we want to preserve both the number of atoms and the atom composition. We therefore assign identifiers from an identifier pool to each entity, which allows us to trace the entities by the assigned identifiers. The two key components are: (i) creating a fixed, finite pool of *identifiers (IDs)* and (ii) defining an unique mapping between entities and identifiers.

Definition 2.1. For a fixed $u \in \mathbb{N}$, let $\mathcal{I} = \{0, 1, \ldots, u-1\}$ be the *identifier pool*. An *identifier* i is an element of the set $\mathcal{I}$.

Definition 2.2. Let E be a finite set of entities and $\mathcal{I}$ an identifier pool. The *identifier assignment pool* is the set of all injective functions from E to $\mathcal{I}$:

$$I = \{\texttt{ida}(\cdot) : E \mapsto \mathcal{I} \mid \forall e_i, e_j \in E : e_i \neq e_j \implies \texttt{ida}(e_i) \neq \texttt{ida}(e_j)\}, \tag{1}$$

An *identifier assignment function* $\texttt{ida}(\cdot)$ is an element of the set I.

Proposition 2.3. *Given an identifier pool $\mathcal{I}$ and a finite set of entities E, an identifier assignment pool I as defined by Definition 2.2 is non-empty if and only if $|E| \leqslant |\mathcal{I}|$.*

Proposition 2.4. *Given an identifier pool $\mathcal{I}$ and a finite set of entities E such that $|E| \leqslant |\mathcal{I}|$, the identifier assignment pool I as defined by Definition 2.2 contains finitely many injective functions.*

Notably, since $\texttt{ida}(\cdot)$ may be selected randomly - the specific choice of the mapping $\texttt{ida}(\cdot)$ can be arbitrary, the only requirement is that an injective mapping between entities and identifiers is established, i.e. each entity is uniquely assigned to an identifier, but not all identifiers need to be assigned to an entity. Further, Proposition 2.3 suggests to use an identifier pool which is large enough, such that a model learned on this identifier pool can generalize across systems with varying numbers of entities. For proofs and an example, we refer to App. C.

2.3 Model Architecture

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. [15] in using a

latent diffusion architecture. We also took inspiration from Jaegle et al. [40] to decompose our model architecture as follows: To map the state of the system composed of N entities to a latent space containing L learned 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 physical information of the individual entities of the system from the latent system representation. To wrap it up, LAM-SLIDE , is built up by an encoder $\mathcal{E}$ - approximator $\mathcal{A}$ - decoder $\mathcal{D}$ architecture, $\mathcal{D} \circ \mathcal{A} \circ \mathcal{E}$. A detailed composition of $\mathcal{E}$ and $\mathcal{D}$ is shown in 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 $\mathbf{Z}^t \in \mathbb{R}^{L \times D_z}$ is constant and should not depend on the different number N of entities. This contrasts with GNNs, where the number of the latent vectors depends on the number of nodes.

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

$$\texttt{ida}(\cdot) \qquad \text{(arbitrary identifier assignment)} \tag{2}$$

$$\mathbf{u}_n = \texttt{IDEmb}(\texttt{ida}(e_n)) \qquad \forall n \in 1, \ldots, N \tag{3}$$

The inputs to the encoder comprise the time dependent location $\mathbf{x}_n^t \in \mathbb{R}^{D_x}$, properties $\mathbf{m}_n \in \mathbb{R}^{D_x}$, and identity representation $\mathbf{u}_n \in \mathbb{R}^{D_x}$ of each entity e_n, as visualized in the right part of Fig. 2. We concatenate the different types of features across all entities of the system: $\mathbf{X}^t = [\mathbf{x}_1^t, .., \mathbf{x}_N^t]$, $\mathbf{M} = [\mathbf{m}_1, .., \mathbf{m}_N]$ and $\mathbf{U} = [\mathbf{u}_1, .., \mathbf{u}_N]$.

The encoder $\mathcal{E}$ maps the input to a latent system representation via

$$\mathcal{E} : [\mathbf{X}^t, \mathbf{M}, \mathbf{U}] \in \mathbb{R}^{N \times (D_x + D_m + D_u)} \mapsto \mathbf{Z}^t \in \mathbb{R}^{L \times D_z},$$

realized by cross-attention [82, 69] between the input tensor $\in \mathbb{R}^{N \times (D_x + D_m + D_u)}$, which serves as keys and values, and a fixed number of L learned latent vectors $\in \mathbb{R}^{D_z}$ [40], which serve as the queries. The encoding process is depicted on the left side of Fig. 2.

Decoder. The aim of the decoder $\mathcal{D}$ is to retrieve the system state information $\mathbf{X}^t$ and $\mathbf{M}$ from the latent state representation $\mathbf{Z}^t$ using the encoded entity identifier embeddings $\mathbf{U}$. The decoder $\mathcal{D}$ maps the latent system representation back into the coordinates and properties of each entity via

$$\mathcal{D} : \mathbf{Z}^t \in \mathbb{R}^{L \times D_z} \times \mathbf{U} \in \mathbb{R}^{L \times D_u} \mapsto [\mathbf{X}^t, \mathbf{M}] \in \mathbb{R}^{N \times (D_x + D_m)}.$$

As shown in the middle part of Fig. 2, $\mathcal{D}$ is realized by cross-attention layers. The latent space representation $\mathbf{Z}^t$ serves as the keys and values in the cross-attention mechanism, while the embedded identifier $\mathbf{u}_n$ acts as the query. Applied to to all ($n = 1, \ldots, N$) system entities e_n, this results in the retrieved system state information $\mathbf{X}^t$ and $\mathbf{M}$. Using the learned identifier embeddings as queries can be interpreted as a form of content-based retrieval and associative memory [6, 38, 69].

Approximator. Finally, the approximator models the system's time evolution in latent space, i.e., it predicts a series of future latent system states $\mathbf{Z}^{[T_o+1:\, T]} = [\mathbf{Z}^{T_o+1}, \ldots, \mathbf{Z}^t, \ldots, \mathbf{Z}^T]$, given a series of initial latent system states $\mathbf{Z}^{[1:\, T_o]} = [\mathbf{Z}^1, \ldots, \mathbf{Z}^t, \ldots, \mathbf{Z}^{T_o}]$,

$$\mathcal{A} : \mathbf{Z}^{[1:\, T_o]} \in \mathbb{R}^{T_o \times L \times D_z} \mapsto \mathbf{Z}^{[T_o+1:\, T]} \in \mathbb{R}^{T_f \times L \times D_z}.$$

Given the analogy of predicting the time evolution of a dynamical system to the task of synthesizing videos, we realized $\mathcal{A}$ by a flow-based model (specifically it's based on the stochastic interpolants framework [2, 56]).

Further architecture details and general training details can be found in App. D, concrete experimental implementation details and details to datasets related to Sect. 3 are in App. E.

3 Experiments

We evaluate LAM-SLIDE on two molecular dynamics datasets: the well established MD17 [21] dataset and on a tetrapeptides dataset (4AA) [41] to investigate its long prediction horizons. We further provide results on n-body system dynamics in App. F.1.

The **MD17** dataset contains simulated molecular dynamics trajectories of 8 small molecules. The size of those molecules ranges from 9 atoms (Ethanol and Malonaldehyde) to 21 atoms (Aspirin). We use 10 frames as conditioning, 20 frames for prediction and report ADE/FDE averaged over $K = 5$ runs. The **4AA** dataset contains explicit-solvent molecular dynamics trajectories simulated using OpenMM [26]. The dataset comprises 3,109 training, 100 validation and 100 test peptides. We use 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.

Tab. 1 shows the performances of LAM-SLIDE and the performances of compared methods on the MD17 benchmark. LAM-SLIDE achieves the lowest ADE/FDE of all methods and for all molecules. These results are particularly remarkable considering that: (1) our model operates without explicit definition molecular bond information, and (2) it surpasses the performance of all equivariant baselines, an inductive bias we intentionally omitted in LAM-SLIDE .
Notably, we train a single model on all molecules – a feat that is structurally encouraged by the design of LAM-SLIDE . For ablation, we also train GeoTDM [33] on all molecules and evaluate the performance on each one of them ("all→each" in the App. Tab. 10). Interestingly, we also observe consistent improvements in the GeoTDM performance; however, GeoTDM's performance does not reach the one of LAM-SLIDE . We also note that our latent model is trained for 2000 epochs, while GeoTDM was trained for 5000 epochs. Trajectories are shown in App. Fig. 9.

Table 1: **Results on the MD17 dataset**. Compared methods have to predict atom positions of 20 frames, conditioned on 10 input frames. Results in terms of ADE/FDE, averaged over 5 runs.

	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 [49][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 [80][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. [30][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 [74][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 [84][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 [85][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 [33] [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	**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. [33].

Tab. 2 compares performances of MDGen [41] and LAM-SLIDE (for details on used metrics see App. E.6). App. Fig. 10 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. Sampled trajectories are shown in App. Fig. 11.

Table 2: **Results on the Tetrapeptide dataset:** Columns denote the JSD between distributions of *torsion angles* (backbone (BB), side-chain (SC), all angles), the TICA, and the MSM metric.

	Torsions			TICA		MSM	Time
	BB	SC	All	0	0,1 joint		
100ns [a]	.103	.055	.076	.201	.268	.208	$\sim$ 3h
MDGen[41][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. [41].

We assess the computational efficiency and scalability of LAM-SLIDE in App. F.2. LAM-SLIDE requires up to 10x-100x fewer function evaluations.

4 Conclusion

LAM-SLIDE is a novel approach for modeling spatial dynamical systems consisting of a variable number of entities within a fixed-size latent system representation, where assignable identifiers allow traceability of individual entities. LAM-SLIDE matches or exceeds specialized methods and offers promising scalability properties. It's minimal reliance on prior knowledge makes it suitable for many tasks, suggesting its potential as a foundational architecture for dynamical systems. We refer to additional experiments in App. F for further concluding insights, for limitations, see App. G.

References

[1] Abramson, J., Adler, J., Dunger, J., Evans, R., Green, T., Pritzel, A., Ronneberger, O., Willmore, L., Ballard, A. J., Bambrick, J., et al. Accurate structure prediction of biomolecular interactions with alphafold 3. *Nature*, pp. 1–3, 2024.

[2] Albergo, M., Boffi, N., and Vanden-Eijnden, E. Stochastic interpolants: A unifying framework for flows and diffusions, 2023. *ArXiv preprint ArXiv230308797*, 2023.

[3] Albergo, M. S. and Vanden-Eijnden, E. Building normalizing flows with stochastic interpolants. *arXiv preprint arXiv:2209.15571*, 2022.

[4] Alkin, B., Fürst, A., Schmid, S. L., Gruber, L., Holzleitner, M., and Brandstetter, J. Universal physics transformers: A framework for efficiently scaling neural operators. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[5] Alkin, B., Kronlachner, T., Papa, S., Pirker, S., Lichtenegger, T., and Brandstetter, J. Neuraldem-real-time simulation of industrial particulate flows. *arXiv preprint arXiv:2411.09678*, 2024.

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

[7] Ansel, J., Yang, E., He, H., Gimelshein, N., Jain, A., Voznesensky, M., Bao, B., Bell, P., Berard, D., Burovski, E., Chauhan, G., Chourdia, A., Constable, W., Desmaison, A., DeVito, Z., Ellison, E., Feng, W., Gong, J., Gschwind, M., Hirsh, B., Huang, S., Kalambarkar, K., Kirsch, L., Lazos, M., Lezcano, M., Liang, Y., Liang, J., Lu, Y., Luk, C., Maher, B., Pan, Y., Puhrsch, C., Reso, M., Saroufim, M., Siraichi, M. Y., Suk, H., Suo, M., Tillet, P., Wang, E., Wang, X., Wen, W., Zhang, S., Zhao, X., Zhou, K., Zou, R., Mathews, A., Chanan, G., Wu, P., and Chintala, S. PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation. In *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. ACM, April 2024. doi: 10.1145/3620665.3640366. URL `https://docs.pytorch.org/assets/pytorch2-2.pdf`.

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

[9] Arriola, M., Gokaslan, A., Chiu, J. T., Yang, Z., Qi, Z., Han, J., Sahoo, S. S., and Kuleshov, V. Block diffusion: Interpolating between autoregressive and diffusion language models. *arXiv preprint arXiv:2503.09573*, 2025.

[10] Ba, J. L. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[11] Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[12] Biewald, L. Experiment tracking with weights and biases, 2020. URL `https://www.wandb.com/`. Software available from wandb.com.

[13] Black, K., Brown, N., Driess, D., Esmail, A., Equi, M., Finn, C., Fusai, N., Groom, L., Hausman, K., Ichter, B., et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

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

[15] Blattmann, A., Rombach, R., Ling, H., Dockhorn, T., Kim, S. W., Fidler, S., and Kreis, K. 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.

[16] Bommasani, R., Hudson, D. A., Adeli, E., Altman, R., Arora, S., von Arx, S., Bernstein, M. S., Bohg, J., Bosselut, A., Brunskill, E., et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

[17] Brandstetter, J., Hesselink, R., van der Pol, E., Bekkers, E. J., and Welling, M. Geometric and physical quantities improve e (3) equivariant message passing. *arXiv preprint arXiv:2110.02905*, 2021.

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

[19] Chen, R. T. and Lipman, Y. Riemannian flow matching on general geometries. *arXiv e-prints*, pp. arXiv–2302, 2023.

[20] Chen, R. T. Q. torchdiffeq, 2018. URL `https://github.com/rtqichen/torchdiffeq`.

[21] Chmiela, S., Tkatchenko, A., Sauceda, H. E., Poltavsky, I., Schütt, K. T., and Müller, K.-R. Machine learning of accurate energy-conserving molecular force fields. *Science advances*, 3(5): e1603015, 2017.

[22] Costa, A. d. S., Mitnikov, I., Pellegrini, F., Daigavane, A., Geiger, M., Cao, Z., Kreis, K., Smidt, T., Kucukbenli, E., and Jacobson, J. Equijump: Protein dynamics simulation via so (3)-equivariant stochastic interpolants. *arXiv preprint arXiv:2410.09667*, 2024.

[23] Dehghani, M., Djolonga, J., Mustafa, B., Padlewski, P., Heek, J., Gilmer, J., Steiner, A. P., Caron, M., Geirhos, R., Alabdulmohsin, I., et al. Scaling vision transformers to 22 billion parameters. In *International Conference on Machine Learning*, pp. 7480–7512. PMLR, 2023.

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

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

[26] Eastman, P., Swails, J., Chodera, J. D., McGibbon, R. T., Zhao, Y., Beauchamp, K. A., Wang, L.-P., Simmonett, A. C., Harrigan, M. P., Stern, C. D., et al. Openmm 7: Rapid development of high performance algorithms for molecular dynamics. *PLoS computational biology*, 13(7): e1005659, 2017.

[27] Esser, P., Kulal, S., Blattmann, A., Entezari, R., Müller, J., Saini, H., Levi, Y., Lorenz, D., Sauer, A., Boesel, F., et al. Scaling rectified flow transformers for high-resolution image synthesis, 2024. *URL https://arxiv. org/abs/2403.03206*, 2, 2024.

[28] Falcon, W. and The PyTorch Lightning team. PyTorch Lightning, March 2019. URL `https://github.com/Lightning-AI/lightning`.

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

[30] Fuchs, F. B., Worrall, D. E., Fischer, V., and Welling, M. Se (3)-transformers: 3d roto-translation equivariant attention networks. *arXiv preprint arXiv:2006.10503*, 2020.

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

[32] Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *International conference on machine learning*, pp. 1263–1272. PMLR, 2017.

[33] Han, J., Xu, M., Lou, A., Ye, H., and Ermon, S. Geometric trajectory diffusion models. *arXiv preprint arXiv:2410.13027*, 2024.

[34] Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

[35] Ho, J., Jain, A., and Abbeel, P. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

[36] Ho, J., Chan, W., Saharia, C., Whang, J., Gao, R., Gritsenko, A., Kingma, D. P., Poole, B., Norouzi, M., Fleet, D. J., et al. Imagen video: High definition video generation with diffusion models. *arXiv preprint arXiv:2210.02303*, 2022.

[37] Hoel, H. and Szepessy, A. Classical langevin dynamics derived from quantum mechanics. *arXiv preprint arXiv:1906.09858*, 2019.

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

[39] Huguet, G., Vuckovic, J., Fatras, K., Thibodeau-Laufer, E., Lemos, P., Islam, R., Liu, C.-H., Rector-Brooks, J., Akhound-Sadegh, T., Bronstein, M., et al. Sequence-augmented se (3)-flow matching for conditional protein backbone generation. *arXiv preprint arXiv:2405.20313*, 2024.

[40] Jaegle, A., Borgeaud, S., Alayrac, J.-B., Doersch, C., Ionescu, C., Ding, D., Koppula, S., Zoran, D., Brock, A., Shelhamer, E., et al. Perceiver io: A general architecture for structured inputs & outputs. *arXiv preprint arXiv:2107.14795*, 2021.

[41] Jing, B., Stärk, H., Jaakkola, T., and Berger, B. Generative modeling of molecular dynamics trajectories. *arXiv preprint arXiv:2409.17808*, 2024.

[42] Joshi, C. K., Fu, X., Liao, Y.-L., Gharakhanyan, V., Miller, B. K., Sriram, A., and Ulissi, Z. W. All-atom diffusion transformers: Unified generative modelling of molecules and materials. *arXiv preprint arXiv:2503.03965*, 2025.

[43] Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., et al. Highly accurate protein structure prediction with alphafold. *nature*, 596(7873):583–589, 2021.

[44] Karplus, M. and Petsko, G. A. Molecular dynamics simulations in biology. *Nature*, 347(6294): 631–639, 1990.

[45] Kingma, D. and Gao, R. Understanding diffusion objectives as the elbo with simple data augmentation. *Advances in Neural Information Processing Systems*, 36, 2024.

[46] Kingma, D. P. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[47] Kingma, D. P. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[48] Kipf, T., Fetaya, E., Wang, K.-C., Welling, M., and Zemel, R. Neural relational inference for interacting systems. In *International conference on machine learning*, pp. 2688–2697. PMLR, 2018.

[49] Köhler, J., Klein, L., and Noé, F. Equivariant flows: sampling configurations for multi-body systems with symmetric energies. *arXiv preprint arXiv:1910.00753*, 2019.

[50] Lipman, Y., Chen, R. T., Ben-Hamu, H., Nickel, M., and Le, M. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.

[51] Lipman, Y., Havasi, M., Holderrieth, P., Shaul, N., Le, M., Karrer, B., Chen, R. T., Lopez-Paz, D., Ben-Hamu, H., and Gat, I. Flow matching guide and code. *arXiv preprint arXiv:2412.06264*, 2024.

[52] Liu, J., Yang, C., Lu, Z., Chen, J., Li, Y., Zhang, M., Bai, T., Fang, Y., Sun, L., Yu, P. S., et al. Towards graph foundation models: A survey and beyond. *arXiv preprint arXiv:2310.11829*, 2023.

[53] Liu, X., Gong, C., and Liu, Q. Flow straight and fast: Learning to generate and transfer data with rectified flow. *arXiv preprint arXiv:2209.03003*, 2022.

[54] Loshchilov, I., Hutter, F., et al. Fixing weight decay regularization in adam. *arXiv preprint arXiv:1711.05101*, 5, 2017.

[55] Lou, A., Meng, C., and Ermon, S. Discrete diffusion modeling by estimating the ratios of the data distribution. *URL https://arxiv. org/abs/2310.16834*, 2024.

[56] Ma, N., Goldstein, M., Albergo, M. S., Boffi, N. M., Vanden-Eijnden, E., and Xie, S. Sit: Exploring flow and diffusion-based generative models with scalable interpolant transformers. *arXiv preprint arXiv:2401.08740*, 2024.

[57] Mao, H., Chen, Z., Tang, W., Zhao, J., Ma, Y., Zhao, T., Shah, N., Galkin, M., and Tang, J. Position: Graph foundation models are already here. In *Forty-first International Conference on Machine Learning*, 2024.

[58] Mayr, A., Lehner, S., Mayrhofer, A., Kloss, C., Hochreiter, S., and Brandstetter, J. 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.

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

[60] Noé, F., Wu, H., Prinz, J.-H., and Plattner, N. Projected and hidden markov models for calculating kinetics and metastable states of complex molecules. *The Journal of chemical physics*, 139, 2013.

[61] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

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

[63] Perez, E., Strub, F., De Vries, H., Dumoulin, V., and Courville, A. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[64] Pérez-Hernández, G., Paul, F., Giorgino, T., De Fabritiis, G., and Noé, F. Identification of slow molecular order parameters for markov model construction. *The Journal of chemical physics*, 139(1), 2013.

[65] Polyak, A., Zohar, A., Brown, A., Tjandra, A., Sinha, A., Lee, A., Vyas, A., Shi, B., Ma, C.-Y., Chuang, C.-Y., et al. Movie gen: A cast of media foundation models. *arXiv preprint arXiv:2410.13720*, 2024.

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

[67] Price, I., Sanchez-Gonzalez, A., Alet, F., Andersson, T. R., El-Kadi, A., Masters, D., Ewalds, T., Stott, J., Mohamed, S., Battaglia, P., et al. Probabilistic weather forecasting with machine learning. *Nature*, 637(8044):84–90, 2025.

[68] Prinz, J.-H., Wu, H., Sarich, M., Keller, B., Senne, M., Held, M., Chodera, J. D., Schütte, C., and Noé, F. Markov models of molecular kinetics: Generation and validation. *The Journal of chemical physics*, 134(17), 2011.

[69] Ramsauer, H., Schäfl, B., Lehner, J., Seidl, P., Widrich, M., Gruber, L., Holzleitner, M., Adler, T., Kreil, D., Kopp, M. K., G, K., and Hochreiter, S. Hopfield networks is all you need. *International Conference on Learning Representations*, 2021.

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

[71] Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. 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.

[72] Sahoo, S., Arriola, M., Schiff, Y., Gokaslan, A., Marroquin, E., Chiu, J., Rush, A., and Kuleshov, V. Simple and effective masked diffusion language models. *Advances in Neural Information Processing Systems*, 37:130136–130184, 2024.

[73] Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J., and Battaglia, P. Learning to simulate complex physics with graph networks. In *International conference on machine learning*, pp. 8459–8468. PMLR, 2020.

[74] Satorras, V. G., Hoogeboom, E., and Welling, M. E(n) equivariant graph neural networks. In Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 9323–9332. PMLR, 18–24 Jul 2021.

[75] Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009. doi: 10.1109/TNN.2008.2005605.

[76] Scherer, M. K., Trendelkamp-Schroer, B., Paul, F., Pérez-Hernández, G., Hoffmann, M., Plattner, N., Wehmeyer, C., Prinz, J.-H., and Noé, F. Pyemma 2: A software package for estimation, validation, and analysis of markov models. *Journal of chemical theory and computation*, 11 (11):5525–5542, 2015.

[77] Seidman, J., Kissas, G., Perdikaris, P., and Pappas, G. J. Nomad: Nonlinear manifold decoders for operator learning. *Advances in Neural Information Processing Systems*, 35:5601–5613, 2022.

[78] Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., and Ganguli, S. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pp. 2256–2265. PMLR, 2015.

[79] Song, Y., Sohl-Dickstein, J., Kingma, D. P., Kumar, A., Ermon, S., and Poole, B. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2021.

[80] Thomas, N., Smidt, T., Kearnes, S., Yang, L., Li, L., Kohlhoff, K., and Riley, P. Tensor field networks: Rotation-and translation-equivariant neural networks for 3d point clouds. *arXiv preprint arXiv:1802.08219*, 2018.

[81] Van Den Oord, A., Vinyals, O., et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

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

[83] Vyas, A., Shi, B., Le, M., Tjandra, A., Wu, Y.-C., Guo, B., Zhang, J., Zhang, X., Adkins, R., Ngan, W., et al. Audiobox: Unified audio generation with natural language prompts. *arXiv preprint arXiv:2312.15821*, 2023.

[84] Xu, C., Tan, R. T., Tan, Y., Chen, S., Wang, Y. G., Wang, X., and Wang, Y. 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.

[85] Xu, P., Hayet, J.-B., and Karamouzas, I. Socialvae: Human trajectory prediction using timewise latents. In *European Conference on Computer Vision*, pp. 511–528. Springer, 2022.

[86] Yadan, O. Hydra - a framework for elegantly configuring complex applications. Github, 2019. URL `https://github.com/facebookresearch/hydra`.

[87] Yu, Z., Huang, W., and Liu, Y. Force-guided bridge matching for full-atom time-coarsened dynamics of peptides. *arXiv preprint arXiv:2408.15126*, 2024.

[88] Zhang, B. and Wonka, P. Lagem: A large geometry model for 3d representation learning and diffusion. *arXiv preprint arXiv:2410.01295*, 2024.

[89] Zhang, B., Tang, J., Niessner, M., and Wonka, P. 3dshape2vecset: A 3d shape representation for neural fields and generative diffusion models. *ACM Transactions on Graphics (TOG)*, 42(4): 1–16, 2023.

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

Appendix

Table of Contents

A Background & Related Work

A.1 Dynamical systems

Formally, we consider a random 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, $\hat{t}_2$:

$$\Phi(0, \mathbf{s}) = \mathbf{s} \tag{4}$$

$$\Phi(\hat{t}_2, \Phi(\hat{t}_1, \mathbf{s})) = \Phi(\hat{t}_1 + \hat{t}_2, \mathbf{s}) \tag{5}$$

We note that Φ does not necessarily need to be defined on the whole space $\mathbb{R} \times \mathcal{S}$, but we assume this 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 [8]. We skip the details, but assume to deal with random dynamical systems for the remainder of the paper. The non-deterministic behavior of such dynamical systems suggests generative modeling approaches.

A.2 Generative modeling

Recent developments in generative modeling have captured widespread interest. The breakthroughs of the last years were mainly driven by diffusion models [78, 79, 35], a paradigm that transforms a simple distribution into a target data distribution via iterative refinement steps. Flow Matching [50, 53, 2] has emerged as a powerful alternative to diffusion models, enable simulation-free training between arbitrary start and target distributions [51] and were also extended to data manifolds [19]. This approach comes with straighter paths, offering faster integration, and has been successfully applied across different domains like images [27], audio [83], videos [65], protein design [39] and robotics [13].

A.3 Latent space modeling

Latent space modeling has achieved remarkable success at image and video generation [15, 27], 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 $\mathcal{D} \circ \mathcal{A} \circ \mathcal{E}$ [77, 4, 5], 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) [71]. This framework was recently used for 3D shape generation, which 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 [89, 88].

A.4 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: $i\hbar\dfrac{\partial \psi}{\partial t} = \hat{H}(t)\psi$. 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 [29, 90]. 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 [37], 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 [66, 18]:

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

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.

A.5 Relationship to Graph Foundation Models

From our perspective, LAM-SLIDE bears a relationship to graph foundation models [GFMs; 52, 57]. Bommasani et al. [16] 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. [57] 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 [52].

A.6 Relationship to Video and Language Diffusion Models

We want to elaborate our perspective on the relationship between LAM-SLIDE and recent advances in video [15] and language diffusion models [72, 55]. At their core, these approaches share a fundamental similarity: they can be conceptualized as a form of unmasking.

In video diffusion models, the model unmasks future frames; in language diffusion models, the model unmasks unknown tokens. Both paradigms learn to recover information that is initially obscured in the sequence, and importantly, both methods do that in parallel over the whole input sequence [9], compared to autoregressive models which predict a single frame or a single token at a time.

Similarly, LAM-SLIDE represents each timestep as a set of latent tokens (or alternatively, as a single token when concatenated). This perspective allows us to seamlessly incorporate recent advances from both video and language diffusion research into our modeling paradigm.

B Notation

Table 3: 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	$\mathbb{N}$
number of future time steps (prediction horizon)	$T_f = T - T_o$	$\mathbb{N}$
time index for sequences of time steps	t	$\mathbb{N}$
system state space	$\mathcal{S}$	application-dependent set, to be further defined
system state	$\mathbf{s}$	$\mathcal{S}$
entity	e	symbolic
number of entities	N	$\mathbb{N}$
entity index	n	$1\,..\,N$
set of entities	E	$\{e_1, \ldots, e_n\}$
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	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	$\mathbf{U}$	$\mathbb{R}^{N \times D_u}$
projection matrices	$\mathbf{Q}, \mathbf{K}, \mathbf{V}$	not specified; depends on number of heads etc.
identifier assignment function	$\texttt{ida}(\cdot)$	$E \mapsto \mathcal{I}$
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.

C Proofs and Example for Terminology

C.1 Proof of Proposition 2.3

The proof is rather simple, but we include it for completeness.

Proposition 2.3. *Given an identifier pool $\mathcal{I}$ and a finite set of entities E, an identifier assignment pool I as defined by Definition 2.2 is non-empty if and only if $|E| \leqslant |\mathcal{I}|$.*

Proof. Assume $|E| > |\mathcal{I}|$. By pigeonhole principle, any function $f : E \mapsto \mathcal{I}$ must map at least two distinct elements of E to the same element in I, Therefore f cannot be injective.

Conversely, if $|E| \leqslant |\mathcal{I}|$, we can construct an injective function from E to $\mathcal{I}$ by assigning each element in E a unique element in $\mathcal{I}$, which is possible because $\mathcal{I}$ has at least as many elements as E.

Therefore, an injective identifier assignment function $\texttt{ida}(\cdot) \in I$ only exists if $|E| \leqslant |\mathcal{I}|$. Hence the set I is non-empty in this case and empty otherwise.

C.2 Proof of Proposition 2.4

Proposition 2.4. *Given an identifier pool $\mathcal{I}$ and a finite set of entities E such that $|E| \leqslant |\mathcal{I}|$, the identifier assignment pool I as defined by Definition 2.2 contains finitely many injective functions.*

Let $n = |E|$ and $m = |\mathcal{I}|$, then the set of infective functions I is bounded and finite:

$$|I| = (m-1)\dots(m-n+1) = \frac{m!}{(m-n)!} = (m)_n \leqslant \inf \tag{7}$$

Where $(m)_n$ is commonly referred as *falling factorials*, the number of injective functions from a set of size n to a set of size m.

C.3 Example: Aspirin

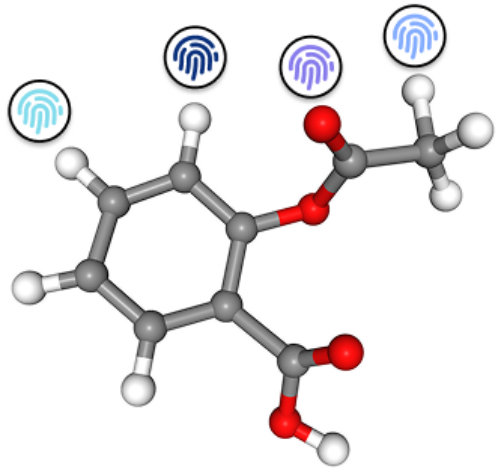

Figure 3: **Example aspirin**: IDs are assigned to the atoms of the molecule.

Aspirin $C_9H_8O_4$ consists of 21 atoms, thus the identifier pool $\mathcal{I}$ needs to have at least 21 unique identifiers. We select an assignment function $\texttt{ida}(\cdot)$, arbitrary, and use it to assign each atom an unique identifier. Notably, e.g. for molecules, we do not explicitly model molecular bond information, as the spatial relationship between atoms (interatomic distances) implicitly capture this information. App. Fig. 3 shows an arbitrary but fixed identifier assignment for aspirin, we illustrate different IDs by colored fingerprint symbols.

D Architecture and Training Details

D.1 Architecture Overview in Detail

App. Fig. 4 shows an expanded view of our architecture, and how the different components of LAM-SLIDE interact. For architectural details on the identifier assignment, the encoder and decoder, and, the approximator we refer to App. D.3, App. D.4, and, App. D.5.

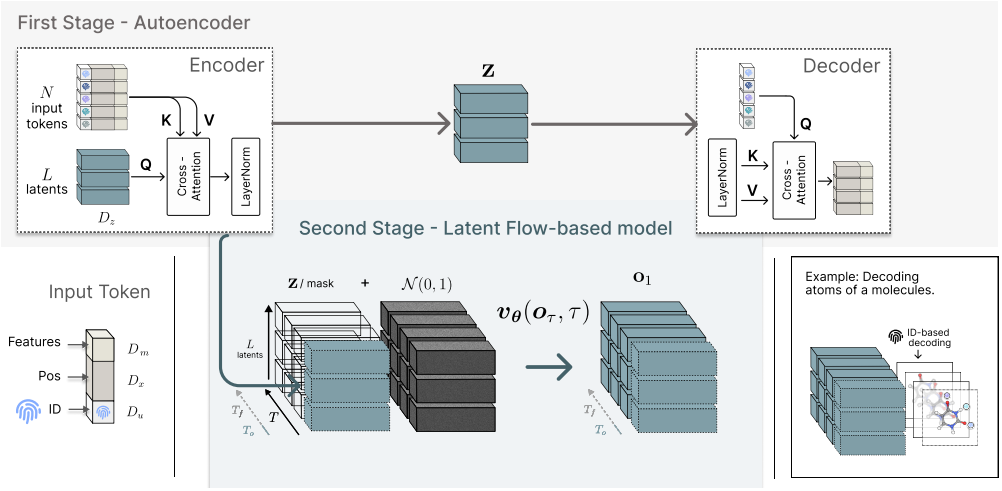

Figure 4: Expanded architectural overview. **First Stage**: The model is trained to reconstruct the encoded system, by querying the latent system representation by IDs. **Second Stage**: Latent flow-based model is trained to predict multiple masked future timesteps. The predicted system states are decoded by the frozen decoder.

D.2 Training Procedure

The training process follows a two-stage approach similar to latent diffusion models [71]. **First Stage:** We train the encoder $\mathcal{E}$ and decoder $\mathcal{D}$, to reconstruct entities from latent space using assigned IDs, see Fig. 2). **Second Stage:** We train the approximator $\mathcal{A}$ on the latent system representations produced by the frozen encoder $\mathcal{E}$ (details in App. E.4).

D.3 Identifier Assignment

We provide pseudocode for the identifier creation in App. Algorithm 1. This algorithm prevents the reuse of already assigned IDs, maintaining unique IDs across all entities. From a practical perspective we sample IDs randomly, so that all entity embeddings receive gradient updates.

Algorithm 1: Identifier Construction

Input : number of entities N; identifier pool size $|\mathcal{I}|$ where $N \leqslant |\mathcal{I}|$; embedding dimension D_u
Output : $\mathbf{U} \in \mathbb{R}^{N \times D_u}$

$\mathbf{U} \leftarrow$ empty matrix of size $N \times D_u$
$S \leftarrow \{\}$ `// track assigned identifiers`
for $i \leftarrow 1$ **to** N **do**
  $r \leftarrow \text{RandomSample}(\mathcal{I} \setminus S)$
  $S \leftarrow S \cup \{r\}$
  $\mathbf{e}_r \leftarrow \text{Embedding}(r)$ `// learnable embeddings`
  $\mathbf{U}[i] \leftarrow \mathbf{e}_r$
return $\mathbf{U}$

D.4 Encoder and Decoder

We provide pseudocode of the forward passes for encoding ($\mathcal{E}$) to and decoding ($\mathcal{D}$) from the latent system space of LAM-SLIDE in App. Algorithm 2 and App. Algorithm 3 respectively. In general, encoder and decoder blocks follow the standard Transformer architecture [82] with feedforward and normalization layers. To simplify the explanation, we omitted additional implementation details here and refer readers to our provided source code.

Algorithm 2: Encoder Function $\mathcal{E}$ (Cross-Attention)

Input : input data $\mathbf{XMU} = [\mathbf{X}, \mathbf{M}, \mathbf{U}] \in \mathbb{R}^{N \times (D_x + D_m + D_u)}$
Output : latent system state $\mathbf{Z} \in \mathbb{R}^{L \times D_z}$
Internal parameters : learned latent queries $\mathbf{Z}_{\text{init}} \in \mathbb{R}^{L \times D_z}$

$\mathbf{K} \leftarrow \text{Linear}(\mathbf{XMU})$
$\mathbf{V} \leftarrow \text{Linear}(\mathbf{XMU})$
$\mathbf{Q} \leftarrow \text{Linear}(\mathbf{Z}_{\text{init}})$
return $\text{LayerNorm}\big(\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V})\big)$ `// without learnable affine parameters`

Algorithm 3: Decoder Function $\mathcal{D}$ (Cross-Attention)

Input : latent system representation $\mathbf{Z} \in \mathbb{R}^{L \times D_z}$; entity representation $\mathbf{u} \in \mathbb{R}^{D_u}$ drawn from $\mathbf{U} \in \mathbb{R}^{N \times D_u}$
Output : $[\mathbf{x}, \mathbf{m}] \in \mathbb{R}^{D_x + D_m}$

$\mathbf{Z} \leftarrow \text{LayerNorm}(\mathbf{Z})$ `// without learnable affine parameters`
$\mathbf{K} \leftarrow \text{Linear}(\mathbf{Z})$
$\mathbf{V} \leftarrow \text{Linear}(\mathbf{Z})$
$\mathbf{q} \leftarrow \text{Linear}(\mathbf{u})$
return $\text{Attention}\big([\mathbf{q}],\, \mathbf{K},\, \mathbf{V}\big)$

For the decoding functionality presented in App. Algorithm 3, we made use of multiple specific decoder blocks depending on the actual task (e.g., for the molecules dataset, we use one decoder block for atom positions and one decoder block for atom types).

D.5 Approximator

To realize the latent approximator, 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 $\epsilon \sim p_0 \coloneqq \mathcal{N}(\mathbf{0}, \boldsymbol{I})$:

$$\mathbf{o}_\tau = \alpha_\tau \mathbf{o}_1 + \sigma_\tau \epsilon, \tag{8}$$

where $\tau \in [0, 1]$ is the time parameter of the flow (to be distinguished from 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 $\alpha_0 = \sigma_1 = 0$, and, $\alpha_1 = \sigma_0 = 1$. 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 $\mathbf{Z} = \mathbf{Z}^{[1:\,T]} = [\mathbf{Z}^{[1:\,T_o]}, \mathbf{Z}^{[T_o+1:\,T]}] \in \mathbb{R}^{T \times L \times D_z}$.

Since the generated trajectories should be conditioned on the latent system representations of initial time frames $\mathbf{Z}^{[1:\,T_o]}$, 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., $\mathbf{C}^{[1:\,T_o]} = \mathbf{Z}^{[1:\,T_o]}$. The remaining tensor entries $\mathbf{C}^{[T_o+1:\,T]}$ are filled up with mask tokens, see App. Fig. 5.

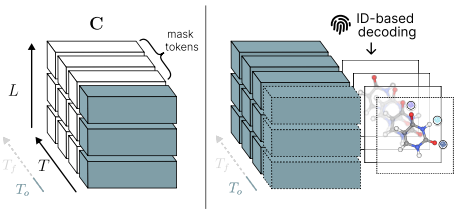

Figure 5: **Left:** The latent model receives conditioning via known tokens (observed timesteps) and mask tokens (for prediction). This example shows conditioning on one timeframe to predict three future ones. **Right:** ID-based decoding, where the predicted atom positions are decoded by the assigned IDs.

The latent model is structured as series transformer blocks [81, 62], alternating between the spatial and the temporal dimension. We parametrized our model via a data prediction objective [51]. Pseudocode of the data prediction network $\boldsymbol{o}_\theta$ forward pass is provided in App. Algorithm 4. The latent layer functionality is given by App. Algorithm 5. The architecture of the latent layers (i.e., our flow model) is based on Dehghani et al. [23], with the additional usage of adaptive layer norm (adaLN) [63] as also used for Diffusion Transformers [62]. The implementation is based on ParallelMLP block codes from Black Forest Labs [14], which was adopted to use it along the latent dimension as well as along the temporal dimension (see App. Fig. 6). The velocity model is obtained via reparameterization as outlined in App. D.6.

Algorithm 4: 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) $\mathbf{o} \in \mathbb{R}^{T \times L \times D_z}$

$\boldsymbol{\tau} \leftarrow \text{Embed}(\tau)$
$\mathbf{o} \leftarrow \text{Linear}(\mathbf{o}_{\text{inter}}) + \text{Linear}(\mathbf{C}) + \text{Embed}(\mathbf{B})$
for $i \leftarrow 1$ **to** *num_layers* **do**
  $\mathbf{o} \leftarrow \text{LatentLayer}(\mathbf{o}, \boldsymbol{\tau})$
$\alpha, \beta, \gamma \leftarrow \text{Linear}(\text{SiLU}(\boldsymbol{\tau}))$
return $\mathbf{o} + \gamma \odot \text{MLP}\big(\alpha \odot \text{LayerNorm}(\mathbf{o}) + \beta\big)$

Algorithm 5: 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 $\mathbf{o}$

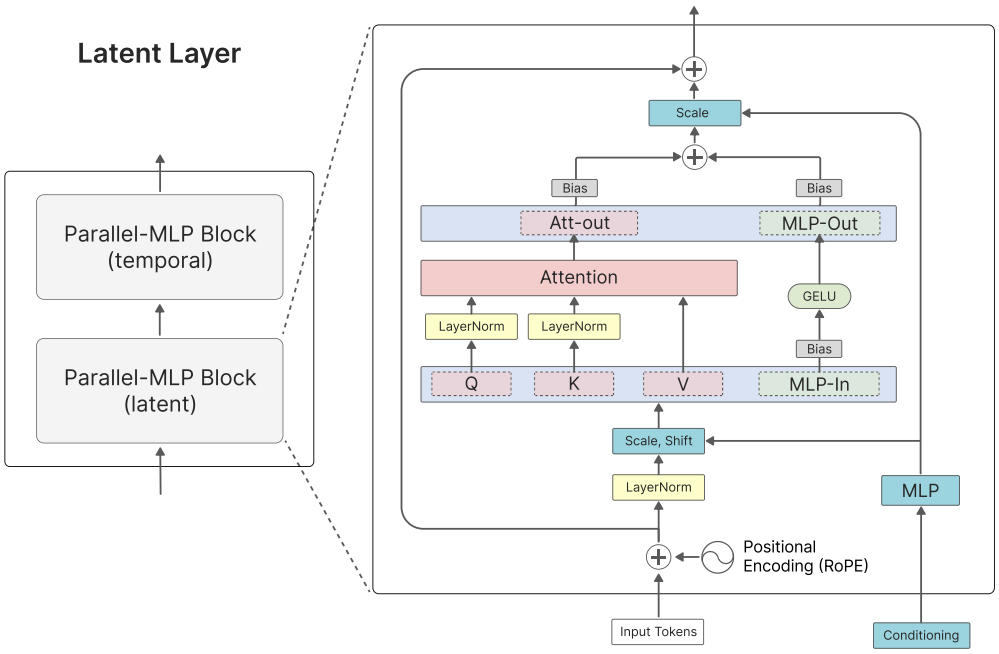

Figure 6: **Left**: LatentLayer of our method, consisting of a latent and a temporal ParallelMLP block. **Right**: Zoomed in view of the ParalellMLP block

Using `einops` [70] notation, the latent layer in App. Fig. 6 can be expressed as:

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

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.

D.6 Additional Information on our Parametrization

General interpolants. The stochastic interpolants framework [2, 3, 56] 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 $\alpha_1 = \sigma_0 = 1$

Interpolants. Two common choices for α_t and σ_t are the Linear and a Generalized Variance-Preserving (GVP) path:

$$\text{Linear:} \quad \alpha_\tau = \tau, \qquad \sigma_\tau = 1 - \tau \tag{9}$$

$$\text{GVP:} \quad \alpha_\tau = \sin\left(\frac{1}{2}\pi\tau\right), \qquad \sigma_\tau = \cos\left(\frac{1}{2}\pi\tau\right) \tag{10}$$

Parametrization. Our latent flow-based model is implemented via data prediction objective [51, 45], with the aim to have small differences:

$$||\hat{\boldsymbol{o}}_\theta(\mathbf{o}; \tau) - \mathbf{o}_1||^2 \, . \tag{11}$$

The velocity model $\hat{\boldsymbol{v}}_\theta$ is obtained by reparameterization according to Lipman et al. [51]:

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

where

$$\hat{\boldsymbol{s}}_\theta(\mathbf{o}; \tau) = -\sigma_\tau^{-2}(\mathbf{o} - \alpha_\tau \hat{\boldsymbol{o}}_\theta(\mathbf{o}; \tau)) \, . \tag{13}$$

For integration, we employed the `torchdiffeq` package [20], which provides solvers for differential equations.

E Experimental Details

E.1 Datasets

Small Molecules (MD17). The MD17 dataset is available at `http://www.sgdml.org/#datasets`. Preprocessing and dataset splits follow Han et al. [33] 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. [41] 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.

N-Body. The dataset creation scripts, along with its predefined splits is available at `https://github.com/hanjq17/GeoTDM`.

E.2 Condition and Prediction Horizon

App. Tab. 4 shows the conditioning and prediction horizon for the individual experiments. For the Tetrapeptides experiments we predicted 1000 steps in parallel and reconditioned the model ten times on the last frame for each predicted block, this concept is similar to Arriola et al. [9].

Table 4: Number of conditioning and predicted frames for the different experiments.

Experiment	Conditioning Frames	Predicted Frames	Total Frames
Molecular Dynamics (MD17)	10	20	30
Molecular Dynamics - Tetrapeptides (4AA)	1	9 999	10 000
N-Body	10	20	30

E.3 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

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

Cross-Entropy Loss. Depending on the experiment we have different CE losses depending on the problem see App. E.5.

$$\mathcal{L}_{CE} = \frac{1}{N}\left(-\sum_{k=1}^{K} y_k \log(p_k)\right) \tag{17}$$

Frame and Torsion Loss. For the Tetrapeptide experiments, we employ two additional auxiliary loss functions tailored to better capture unique geometric constraints of proteins, complementing our primary optimization objectives: a frame loss $\mathcal{L}_{frame}$, which is based on representing all atoms withing a local reference frame [1, Algorithm 29], and a torsion loss torsion loss $\mathcal{L}_{\text{tors}}$ inspired by Jumper et al. [43].

E.4 Implementation Details

Training procedure. (i) **First Stage**. In the first stage we train the encoding and decoding functions $\mathcal{E}$ and $\mathcal{D}$ in an auto-encoding fashion, i.e., we optimize for a precise 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.5. 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 system representations. To learn a consistent behavior over time, we pass $\mathbf{U}$ from the encoder $\mathcal{E}$ to the decoder $\mathcal{D}$. To avoid high variance latent spaces we used layer-normalization [10] (see. App. E.4).

Data Augmentation. To compensate for the absence of built-in inductive biases such as equivariance/invariance with respect to spatial transformations, we apply random rotations and translations to the input coordinates.

Identifiers. For the embedding of the identifiers we use a `torch.nn.Embedding` [61] layer, where we assign a random subset of the possible embeddings to the entities in each training step. See also App. Algorithm 1.

Latent space regularization. To avoid high variance latent spaces, Rombach et al. [71] relies on KL-reg., imposing a small KL-penalty towards a standard normal on the latent space, as used in VAE [46]. Recent work [88] has shown that layer normalization [10] 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.5. Where we decode the the predicted latent system representations 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 train GeoTDM [33] on all molecules and evaluate the performance on each one of them ("all→each" in the App. Tab. 10). Interestingly, we also observe consistent improvements in the GeoTDM performance; however, GeoTDM's performance does not reach the one of LAM-SLIDE .

Tetrapeptides. For the experiments on tetrapeptides, we employ the Atom14 representation as used in AlphaFold [1]. 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 computationally more efficient.

E.5 Hyperparameters

App. Tables 5 to 7 show the hyperparameters for the individual tasks, loss functions are as defined in App. E.3. For all trained models we use the AdamW [47, 54] optimizer and use EMA [31] in each update step with a decay parameter of $\beta = 0.999$.

E.6 Evaluation Details

Prediction Performance for MD17 and N-Body. For MD17 and the n-body experiments, we utilized the Average Discrepancy Error (ADE) and the Final Discrepancy Error (FDE), defined as $\text{ADE}(\mathbf{X}, \hat{\mathbf{X}}) = \frac{1}{(T-T_o)N} \sum_{t=T_o+1}^{T} \sum_{i=1}^{N} \|\mathbf{X}_i^t - \hat{\mathbf{X}}_i^t\|_2$, $\text{FDE}(\mathbf{X}, \hat{\mathbf{X}}) = \frac{1}{N} \sum_{i=1}^{N} \|\mathbf{X}_i^T - \hat{\mathbf{X}}_i^T\|_2$, 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 trajectory forecasting [84, 85].

Prediction Performance for Tetrapeptides. Our analysis of the Tetrapeptide trajectories utilized PyEMMA [76] and followed the procedure as Jing et al. [41]. For the MD experiments on peptides (Tetrapeptides), we use 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 use *Time-lagged Independent Component Analysis* (TICA) [64], focusing on the slowest components TIC 0 and TIC 1. To investigate metastable state transitions we make use of *Markov State Models* (MSMs) [68, 60]. For the evaluation of these metrics we relied on the implementations provided by [41].

Time and Scalability. For inference time and scalability, we assess the *number of function evaluations* (NFE) and report performance of our method for different model sizes.

E.7 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.

E.8 Software

We used `PyTorch 2` [7] for the implementation of our models. Our training pipeline was structured with `PyTorch Lightning` [28]. We used `Hydra` [86] to run our experiments with different hyperparameter settings. Our experiments were tracked with `Weights & Biases` [12].

Table 5: Hyperparameter configuration for the small molecule (MD17) experiments.

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 6: Hyperparameter configuration for the Tetrapeptides experiments.

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 [20]
ODE steps	adaptive

Table 7: Hyperparameter configuration for the N-Body experiments (App. F.1).

First Stage	
Network	
Encoder	
Number of latents L	16
Number of entity embeddings	10
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
Training	
Learning rate	1e-3
Batch size	128
Epochs	2K
Precision	32-Full
Second Stage	
Setup	
Condition	10 Frames
Prediction	20 Frames
Network	
Hidden dimension	256
Number of Layers	6
Auxiliary - Loss	**Weight**
$\mathcal{L}_{pos}(\mathbf{X}, \hat{\mathbf{X}})$	0.0
$\mathcal{L}_{int}(\mathbf{X}, \hat{\mathbf{X}})$	0.0
Training	
Learning rate	1e-3
Learning rate scheduler	CosineAnnealing(min_lr=1e-7)
Batch size	64
Epochs	1K
Precision	BF16-Mixed
Inference	
Integrator	Euler
ODE steps	10

F Additional Experiments

We conducted additional experiments to investigate how well our method works on a different dynamic system and to investigate its computational efficiency and its sensitivity to different hyperparameter settings.

F.1 N-Body System Dynamics (Particle Systems)

We evaluate LAM-SLIDE across three distinct N-Body simulation scenarios: a) *Charged Particles*: comprising particles with randomly assigned charges $+1/-1$ interacting via Coulomb forces [48, 74]; b) *Spring Dynamics*: consisting of $N = 5$ particles with randomized masses connected by springs with a probability 0.5 between particle pairs [48]; and c) *Gravitational Systems*: containing $N = 10$ particles with randomized masses and initial velocities governed by gravitational interactions [17]. For all three scenarios, we consider 10 conditioning frames and 20 frames for prediction. In line with Han et al. [33], we use 3000 trajectories for training and 2000 trajectories for validation and testing, we report ADE/FDE averaged over $K = 5$ runs.

LAM-SLIDE achieves the best performance in terms of ADE/FDE for the Charged Particles and Gravity scenarios and competitive second-rank performance in the Spring Dynamics scenario, see App. Tab. 8. Unlike compared methods, LAM-SLIDE achieves these results without computing intermediate physical quantities such as velocities or accelerations. We present sampled trajectories in App. Fig. 12.

Table 8: **Results on generation on N-body dataset**, in terms of ADF/FDE averaged over 5 runs.

	Particle		Spring		Gravity	
	ADE	FDE	ADE	FDE	ADE	FDE
RF [49][a]	0.479	1.050	0.0145	0.0389	0.791	1.630
TFN [80][a]	0.330	0.754	0.1013	0.2364	0.327	0.761
SE(3)-Tr [30][a]	0.395	0.936	0.0865	0.2043	0.338	0.830
EGNN [74][a]	0.186	0.426	0.0101	0.0231	0.310	0.709
EqMotion [84][a]	0.141	0.310	0.0134	0.0358	0.302	0.671
SVAE [85][a]	0.378	0.732	0.0120	0.0209	0.582	1.101
GeoTDM [33][a]	<u>0.110</u>	<u>0.258</u>	**0.0030**	**0.0079**	<u>0.256</u>	<u>0.613</u>
LAM-SLIDE	**0.104**	**0.238**	<u>0.0070</u>	<u>0.0135</u>	**0.157**	**0.406**

[a] Results from Han et al. [33].

F.2 Computational Efficiency and Scaling Behavior

We conduct a comparative analysis on computational efficiency by measuring the number of function evaluations (NFEs) required to achieve the performance results reported in the main section of our publication. As shown in App. Tab. 9, our approach demonstrates remarkable efficiency compared to the previous state-of-the-art method, GeoTDM [33], across MD17 molecular dynamics and N-Body simulations experiments. Our model consistently requires significantly fewer NFEs than GeoTDM to reach comparable or superior performance levels.

It is worth noting that flow-based models generally require fewer NFEs compared to diffusion-based approaches like GeoTDM. However, this efficiency advantage does not come at the expense of performance quality [27]. Indeed, the relationship between NFEs and performance is not strictly monotonic, as demonstrated in other domains. For instance, Esser et al. [27] achieved optimal image generation results in terms of FID [34] with 25 NFEs, showing that computational efficiency and high performance can be simultaneously achieved with properly designed architectures.

Further, in the case of the Tetrapeptides (4AA) experiments shown in the main part of the paper, we use an adaptive step size solver to reach the reported performance, which achieved better results than a Euler solver. We use Dopri5 as implemented in the `torchdiffeq` package [20].

Table 9: Comparison of the number of functions evaluations (NFEs) for LAM-SLIDE and GeoTDM.

	N-Body	MD17
GeoTDM [33][a]	1000	1000
LAM-SLIDE	**10**	**10**

[a] Results from Han et al. [33].

We conducted scaling experiments on both the MD17 and the Tetrapeptides (4AA) datasets to evaluate how LAM-SLIDE 's performance scales with model size. On MD17, we evaluate LAM-SLIDE using model variants with 1.7M, 2.1M, and 2.5M parameters. Our results show that, for nearly all molecules, performance improves with parameter count in terms of ADE/FDE, see App. Tab. 10. Similarly, on the Tetrapeptides dataset, we evaluate using model variants with 4M, 7M, 11M, and 28M parameters. All performance metrics show consistent improvement with increased model capacity, see App. Tab. 11. These findings indicate favorable scaling behavior of our method and suggests that LAM-SLIDE benefits from larger model capacity and could potentially achieve even better results with additional computational resources.

Table 10: **Method comparison for forecasting MD trajectories of small molecules**. Compared methods predict atom positions for 20 frames, conditioned on 10 input frames. Results are reported 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 [49][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 [80][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. [30][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 [74][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 [84][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 [85][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	**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	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	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. [33].

Table 11: **Method comparison for predicting MD trajectories of tetrapeptides**. The columns denote the JSD between distributions of *torsion angles* (backbone (BB), side-chain (SC) and all angles), the TICA, the MSM metric, and the number of parameters.

	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. [41].

F.3 Number of Learned Latent Vectors

We conducted experiments to quantify the relationship between model performance and the number of latent vectors L using the MD17 dataset. As shown in App. Tab. 12, performance increases with the number of latent vectors L. Of particular significance is the performance at $L = 21$, which corresponds to the maximum number of entities, allowing us to investigate whether this constitutes an upper bound on model capacity. Notably, performance continues to improve at $L = 32$, indicating that model capacity scales favorably even beyond the number of entities. Still at $L = 16$, representing a compressed latent representation, our model remains competitive with the second best method GeoTDM [33].

We further analyze if the improvement by increasing L is due to the reconstruction performance of the encoder-decoder only. App. Fig. 8 shows the reconstruction error for varying number of latent vectors L. Even with substantially fewer latent vectors than entities, the model achieves good reconstruction performance. This gap suggests, that the performance gains from increasing L are not due to improved reconstruction, but from the ability of the model to leverage the enlarged latent space representation better.

Table 12: Model performance in terms of ADF/FDE with respect to **different number of latent vectors** L.

	Aspirin		Benzene		Ethanol		Malonaldehyde		Naphthalene		Salicylic		Toluene		Uracil	
	ADE	FDE	ADE	FDE	ADE	FDE	ADE	FDE	ADE	FDE	ADE	FDE	ADE	FDE	ADE	FDE
LAM-SLIDE ($L = 4$)	0.354	0.483	0.213	0.264	0.420	0.541	0.366	0.537	0.210	0.223	0.253	0.286	0.316	0.418	0.243	0.275
LAM-SLIDE ($L = 8$)	0.234	0.315	0.114	0.135	0.242	0.361	0.201	0.300	0.157	0.163	0.179	0.201	0.206	0.250	0.158	0.180
LAM-SLIDE ($L = 16$)	0.099	0.146	0.039	0.049	0.108	0.187	0.095	0.149	0.070	0.089	0.075	0.101	0.073	0.103	0.071	0.093
LAM-SLIDE ($L = 21$)	0.078	0.118	0.031	0.041	0.097	0.175	0.082	0.135	0.054	0.074	0.059	0.085	0.057	0.085	0.059	0.083
LAM-SLIDE ($L = 32$)	**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**

F.4 Identifier Pool Size

We evaluate the impact of identifier pool size using the MD17 dataset. This dataset contains at most 21 atoms, so in general an identifier pool of $|\mathcal{I}| = 21$ would be enough. However, for the results shown in the main paper, we used $|\mathcal{I}| = 32$. To investigate the impact of a larger identifier pool $\mathcal{I}$, we conduct additional experiments by training multiple first-stage models with varying identifier pool sizes and report the reconstruction error measured by Euclidean distance. App. Fig. 7 shows that the reconstruction error increases with the size of the identifier pool, since a larger pool results in fewer updates to each entity embedding during training.

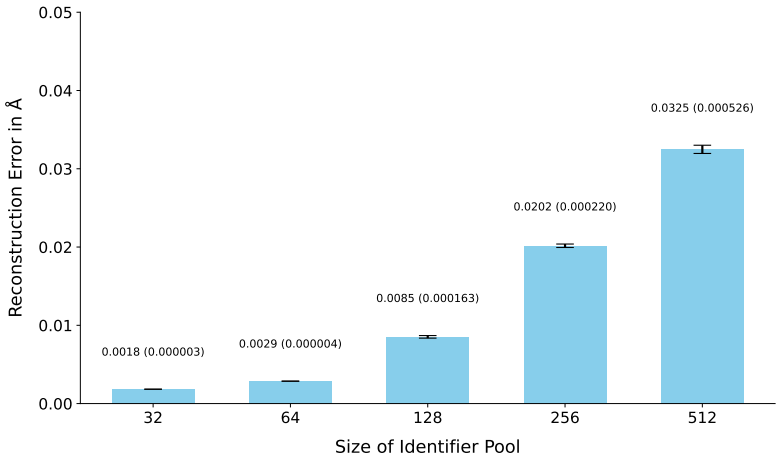

Figure 7: Reconstruction error in Å for the MD17 dataset. We report the reconstruction error for the encoder-decoder model for **different identifier pool sizes**. The error bars show the standard deviation across five runs with different random ID assignments.

F.5 Identifier Assignment

To assess the impact of different ID assignments on reconstruction performance, we run our model five times with different random ID assignments and measure the standard deviation across these assignments in terms of reconstruction error in Å. Results are shown in App. Figures 7 and 8. The low standard deviation across different ID assignments demonstrates the robustness of our model with respect to random ID assignment.

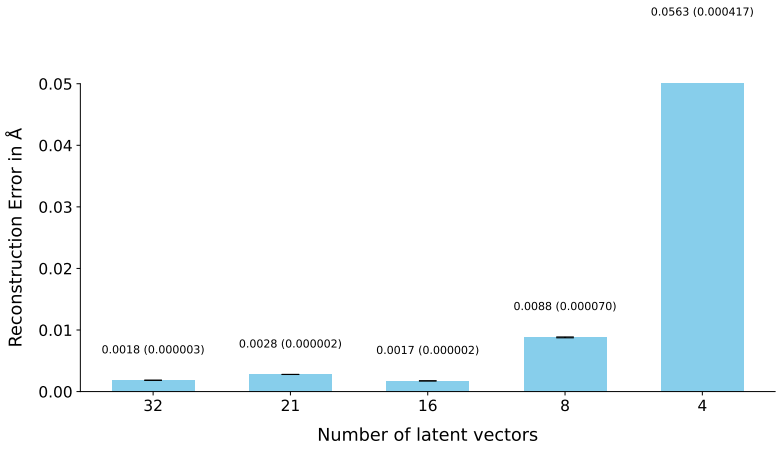

Figure 8: Reconstruction error in Å for the MD17 dataset. We report the reconstruction error for the encoder-decoder model for **different number of latent vectors** L. The error bars show the standard deviation across five runs with different random ID assignments.

G Extended Discussion

Our experiments indicate that our architecture is applicable to a diverse set of problems; however, a few limitations provide opportunities for future improvement. While our current approach successfully allows to compress entities with beneficial reconstruction performance, our experiments indicate a tradeoff between the number of latent space vectors to encode system states and the performance of our latent model, for experimental details on this see App. F.3.

H Visualizations

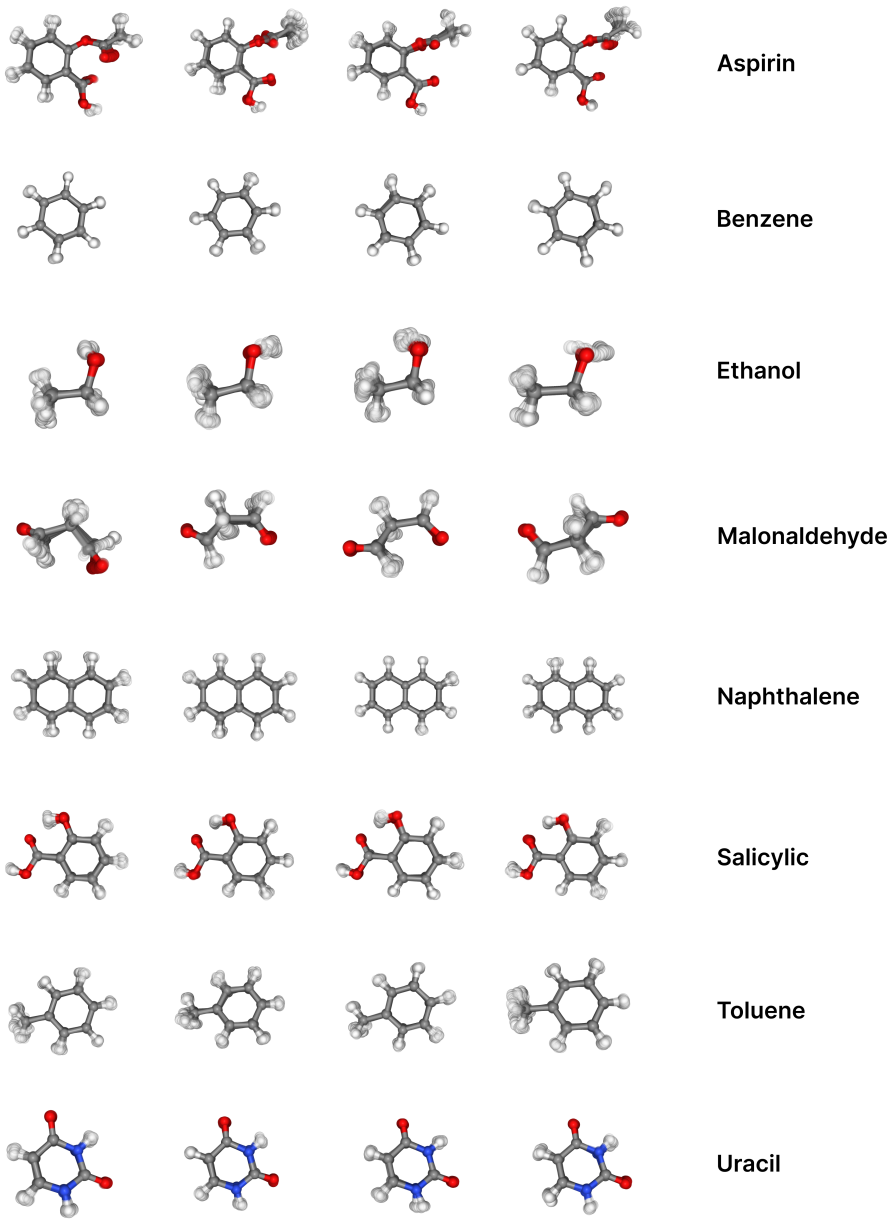

Figure 9: **Molecular dynamics trajectories from the MD17 dataset**, showing time-evolved structural predictions for each molecule. For every compound, we display four distinct trajectory predictions, with each prediction comprising 20 superimposed time frames to illustrate the range of conformational changes.

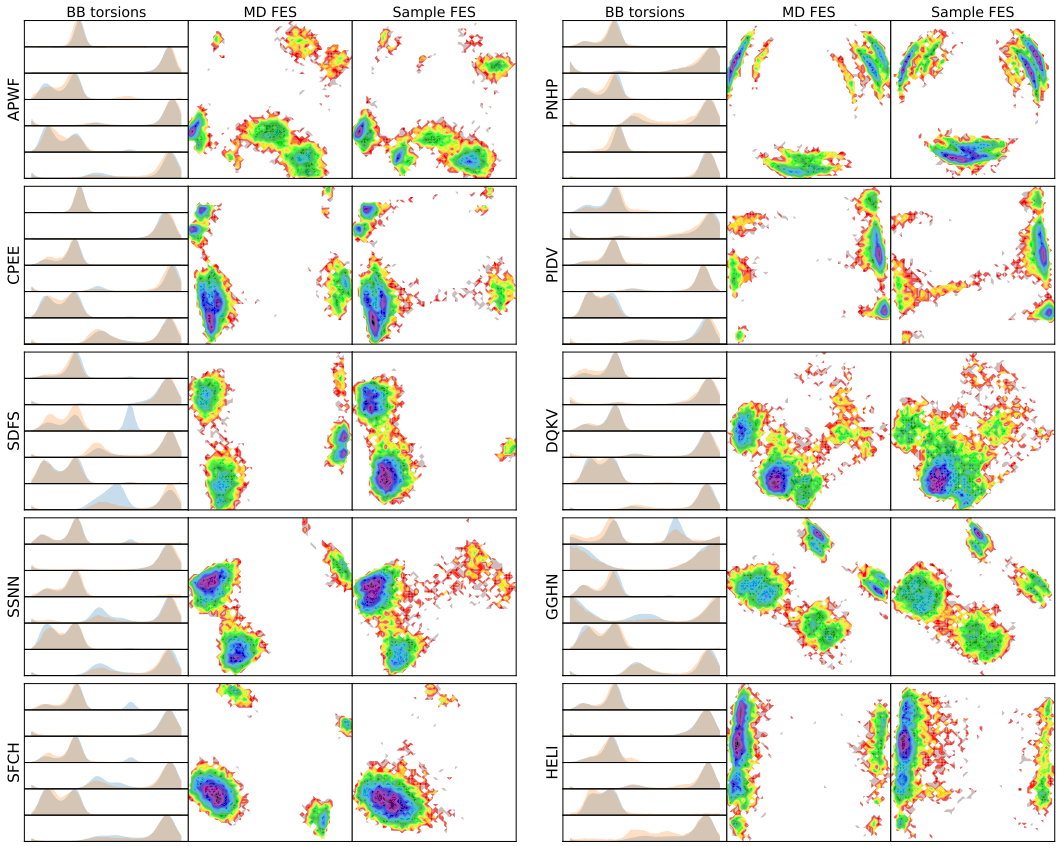

Figure 10: **Torsion angle distributions** of the six backbone torsion angles, comparing molecular dynamics (MD) trajectories (orange) and sampled trajectories (blue); and **Free energy surfaces** projected onto the top two time-lagged independent component analysis (TICA) components, computed from both backbone and sidechain torsion angles.

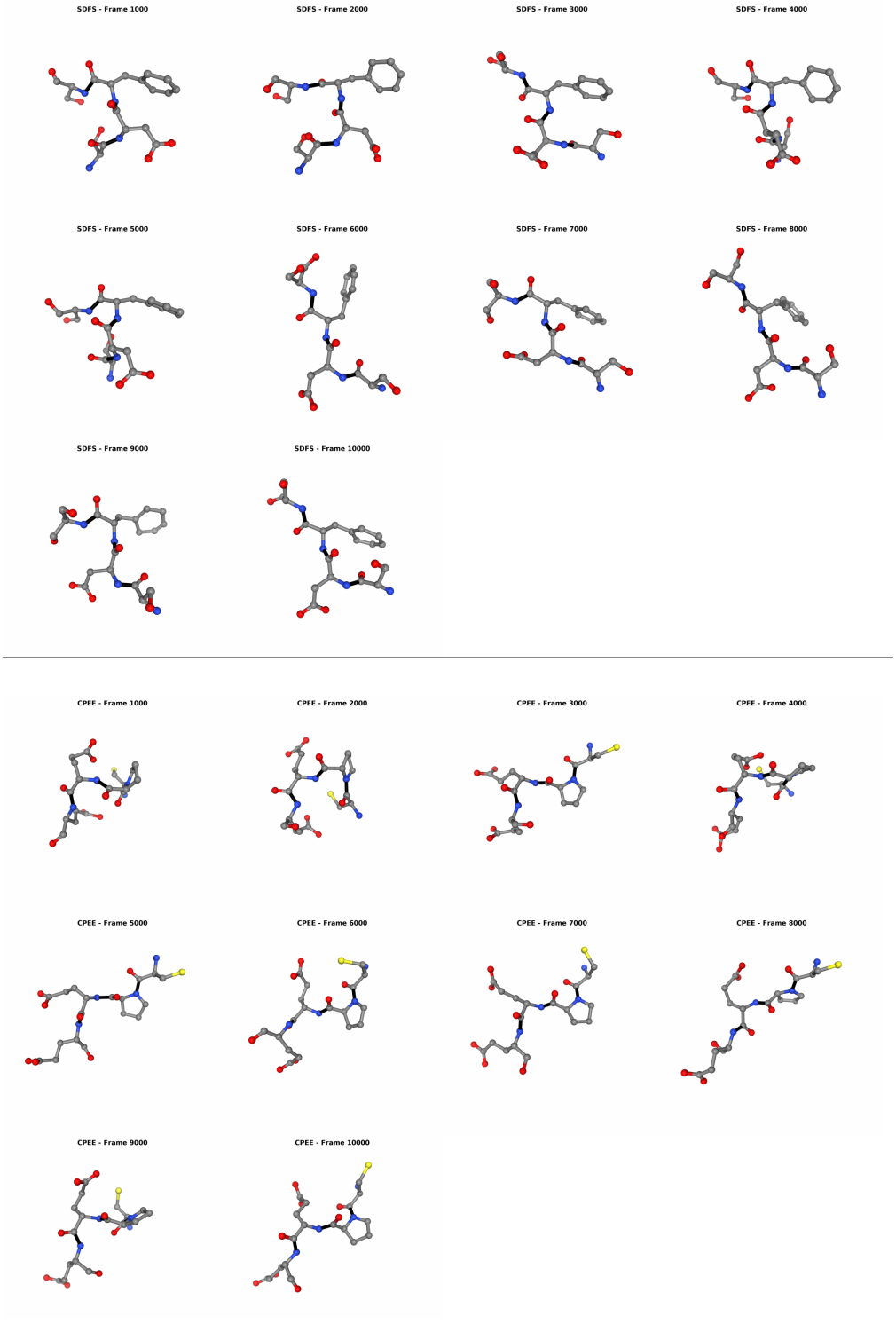

Figure 11: **Molecular Dynamics trajectories from the Tetrapeptides (4AA) dataset**, showing time-evolved structural predictions for ten frames at an interval of 1000 frames. **Top**: SDFS (Serine - Aspartic Acid - Phenylalanine - Serine) peptide. **Bottom**: CPEE peptide (Cysteine - Proline - Glutamic Acid - Glutamic Acid).

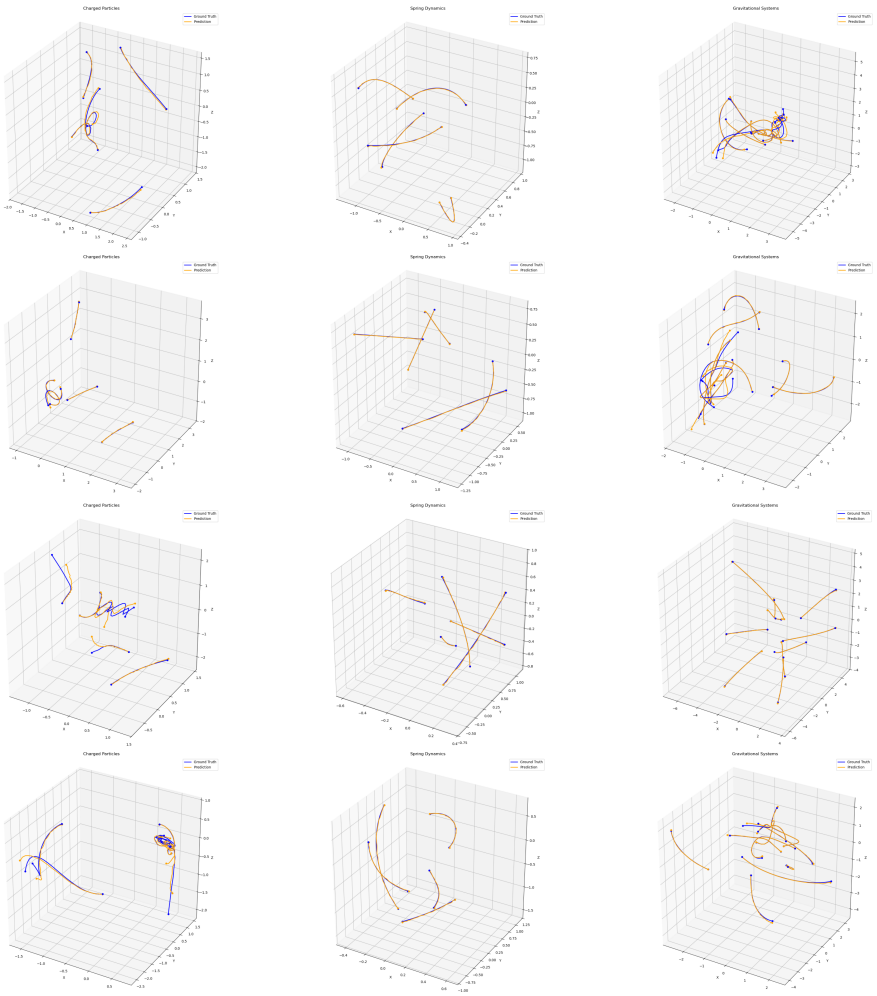

Figure 12: Trajectories from the N-Body dataset, predicted vs ground truth trajectories. **Left**: Charged particles. **Middle**: Spring dynamics. **Right**: Gravitational system.

839