Transformers on Manifolds

Editors: List of editors' names

Abstract

Many prediction tasks require outputs to lie on structured constraint sets (e.g., spheres, rotation groups, Stiefel/PSD manifolds), yet standard transformers provide no mechanism to enforce feasibility. We study *geometry-aware transformers*, derived from projected dynamical variants of neural ordinary differential equations (NODEs), that respect constraints by construction. First, we introduce projected transformer layers that apply analytical projections (e.g., $\mathbb{S}^{d-1}$, Stiefel, PSD-trace) between blocks. Second, for Lie-group targets we propose an *exponential transformer* that learns Lie-algebra coefficients and updates intrinsically via the matrix exponential. Third, when no closed-form projection exists, we learn a differentiable projection with *flow matching* that acts as a data-driven retraction onto the (unknown) manifold. On synthetic dynamical systems over $\mathbb{S}^2$ and $SO(3)$, we evaluate Euclidean accuracy (MSE) and feasibility (norm/determinant statistics). Explicit projections and intrinsic exponential layers achieve *perfect* feasibility, with exponential updates particularly strong on $SO(3)$ and show that our models have the best performance.

Keywords: Geometric deep learning, transformers, manifolds, Lie groups, projected dynamical systems, flow matching, constrained learning, neural ODEs

1. Introduction

Many learning problems impose *hard* geometric or structural constraints on outputs: protein backbones live on (products of) spheres and rotation groups (AlQuraishi, 2019), drone-vision pose estimates are elements of $SE(3)$ or $SO(3)$ (Geiger et al., 2012; Burri et al., 2016), and covariance operators must be positive semidefinite with fixed trace (George et al., 2023). In such settings, minimizing a Euclidean loss is secondary to *feasibility*: predictions must lie on a target set $K \subset \mathbb{R}^d$ (often a manifold $\mathcal{M}$) to be physically or semantically meaningful. Standard transformers lack mechanisms to guarantee feasibility; even small violations can accumulate across layers and corrupt downstream computations (e.g., non-orthogonal "rotations").

There have been a number of works in recent years on manifold-valued neural networks. In Falorsi et al. (2018) and Davidson et al. (2018) developed Variational Autoencoders (VAEs) equipped with manifold latent spaces, including hyperspheres, Lie groups, and toroidal structures, demonstrating the utility of manifold latent representations. In Falorsi and Forré (2020), the authors construct a manifold variant of neural ODEs. The goal of this paper was to extend the adjoint method used to train neural ODEs Chen et al. (2018) to these manifold variants. In Lou et al. (2020), the authors similarly construct a manifold version of neural ODEs and develop an adjoint sensitivity method based on local coordinate charts. The work in Katsman et al. (2023) considers a Riemannian version of neural networks by replacing the addition operation used for defining residual connections with the Riemannian exponential map. In Elamvazhuthi et al. (2023) consider a class of manifold valued neural ODEs with limited width and characterize their universal approximation capabilities. In Kratsios et al.

(2022), the authors consider a variation of transformers that preserves the data geometry and prove a universal approximation result for functions that satisfy constraints.

We study *geometry-aware transformers* that remain on K by construction. Our guiding principle is the following: perform standard transformer computations in ambient space and interleave them with *constraint-preserving* updates that map activations back to K. We instantiate this principle in three complementary ways: **(1) Explicit projections.** For common sets (spheres $\mathbb{S}^{d-1}$, Stiefel manifolds, PSD cones with trace) we apply analytical projections between blocks, yielding deterministic, exactly feasible outputs. **(2) Intrinsic Lie-group updates.** When K is a Lie group (e.g., $SO(3)$), we predict Lie–algebra elements and update via the exponential map, staying on K without post-hoc projection. **(3) Learned projections via flow matching.** If no projection is known, we learn a differentiable map that retracts ambient points to K by training a time-conditioned vector field with flow matching (Lipman et al., 2022) and integrating it backward as a projection operator.

Contributions. We formulate transformer layers that guarantee feasibility on common manifolds via explicit projections and intrinsic Lie–group updates. We introduce a data-driven, differentiable projection trained by flow matching that acts as a learned retraction when K is unknown. On synthetic dynamics over $\mathbb{S}^2$ and $SO(3)$, we show: (i) explicit and intrinsic methods achieve *exact* feasibility; (ii) learned projections achieve competitive accuracy without prior knowledge of the manifold and approach feasibility; and (iii) frequent interlayer projection helps only when the projection is exact and inexpensive.

1.1. Problem Formulation

To construct geometry preserving transformers we take the point of view of Neural Ordinary Differential equations (NODEs). To generalize NODEs to the geometry preserving setting, we introduce the notion of a projected dynamical system.

Let $\Omega \subseteq \mathbb{R}^d$ be a compact subset representing a smooth manifold with C^2 boundary. Given a vector field $F : \Omega \to \mathbb{R}^d$, we define a projected dynamical system as:

$$\dot{x} = P_{T_x\Omega} F(x) \quad x(0) = x_0 \in \Omega \tag{1}$$

where $P_{T_x\Omega} : \mathbb{R}^d \to T_x\Omega$ denotes the orthogonal projection onto the tangent cone $T_x\Omega$ at point $x \in \Omega$. Let $x_0 \mapsto \Phi_F(x_0)$ be the flow map associated with the Equation 1. Our goal is to approximate the flow map $\Phi_F : \Omega \to \Omega$ by approximating the vector field F using neural network $f_\theta : \Omega \to \Omega$, using the projected NODE:

$$\dot{x} = P_{T_x\Omega} f_\theta(x) \quad x(0) = x_0 \in \Omega \tag{2}$$

The key challenge lies in ensuring that the learned approximation respects the manifold geometry.

2. Methods

Projected Transformers by Euclidean Forward Euler Discretization. The first method, we consider, adds a projection back onto the manifolds (see Algorithm 1). Suppose, the network has L layers with parameters $\Theta = \{\theta^\ell\}_{\ell=1}^L$. For $\ell = 1, \ldots, L$ we compute a pre-projection activation $u^\ell = \sigma^\ell(h^{\ell-1};\theta^\ell)$ with $h^0 = x$, where σ^ℓ is the activation function. This is followed by a projection $h^\ell = \Pi_{\mathcal{M}_\ell}(u^\ell)$ onto a target manifold $\mathcal{M}_\ell \subset \mathbb{R}^{d_\ell}$. The map $\Pi_{\mathcal{M}_\ell}$ is a pointwise projection that acts independently on each sample in the minibatch. We use Euclidean projections when closed forms exist.

1. Sphere $\mathbb{S}^{d-1}$: $\Pi(u) = u/\|u\|$.
2. Stiefel $\mathrm{St}(p,d)$: thin SVD $u = U\Sigma V^\top$ and $\Pi(u) = UV^\top$.
3. PSD with trace τ: eigendecomposition $u = Q\Lambda Q^\top$, set $\Lambda_+ = \max(\Lambda, 0)$ entrywise, rescale to $\mathrm{tr}(\Lambda_+) = \tau$, and return $Q\Lambda_+ Q^\top$.

Projected Transformers (learned projection via flow matching). When a closed-form projection $\Pi_{\mathcal{M}_\ell}$ is unavailable, we replace it with a *learned*, differentiable map $P_{\phi_\ell} : \mathbb{R}^{d_\ell} \to \mathcal{M}_\ell$ trained by flow matching (see Algorithm 2). The layer update becomes

$$u^\ell = \sigma^\ell(h^{\ell-1};\theta^\ell), \qquad h^\ell = P_{\phi_\ell}(u^\ell), \qquad \ell = 1,\dots,L,$$

so that each block returns to $\mathcal{M}_\ell$ by applying P_{ϕ_ℓ} pointwise on the minibatch.

Let μ be a training distribution supported on $\mathcal{M} \subset \mathbb{R}^d$. We parameterize a time-conditioned vector field $v_\phi : \mathbb{R}^d \times [0,T] \to \mathbb{R}^d$ and train it to reproduce *outward* velocities along straight "rays" emanating from the manifold. Concretely, draw $X \sim \mu$, a perturbation velocity $V \sim \nu$ (e.g., $\mathcal{N}(0,\sigma^2 I)$), and a time $t \sim \mathrm{Unif}[0,T]$, and set $Y_t = X + tV$. The flow-matching loss is

$$\mathcal{L}(\phi) \;=\; \mathbb{E}_{X,V,t}\big\| \, v_\phi(Y_t,t) - V \, \big\|^2, \tag{3}$$

which enforces $\dot{Y}_t = V$ along these rays. After training, the *approximate projection* P_ϕ is defined as the terminal value of the backward flow driven by $-v_\phi$:

$$\frac{d}{ds}x(s) \;=\; -\,v_\phi\big(x(s),s\big), \qquad s \in [0,T], \qquad x(T) = x_0, \qquad P_\phi(x_0) := x(0). \tag{4}$$

Intuitively, (3) learns how points depart from $\mathcal{M}$; integrating (4) reverses that departure to bring x_0 back toward $\mathcal{M}$. In the idealized setting (perfect training and sufficiently small T), P_ϕ is a smooth retraction onto $\mathcal{M}$ on a tubular neighborhood. That this map approximates the projection can be justified using asymptotic properties of the heat kernel. From the perspective of diffusion models, we are using the reverse process to approximate the projection map.

Exponential Transformers by Geometric Forward Euler Discretization on Lie Groups. When the target manifold is a Lie group $G \subset \mathbb{R}^d$ of dimension m, we can approximate smooth maps $G \to G$ intrinsically by learning *Lie–algebra coefficients* and then mapping back to the group with the exponential (see Algorithm 3). Let $\mathfrak{g} = T_eG$ and fix a basis $\{E_1,\dots,E_m\}$ of $\mathfrak{g}$. The *right translation* by $g \in G$, $R_g(h) = h\,g$, gives the right-trivialization of the tangent bundle. Any smooth vector field X on G is written uniquely as

$$X(g) \;=\; \sum_{i=1}^{m} v_i(g)\, E_i^R(g) \;=\; \sum_{i=1}^{m} v_i(g)\,(dR_g)_e E_i,$$

for smooth coefficient functions $v_i : G \to \mathbb{R}$. A neural network $f_\theta : G \to \mathbb{R}^m$ learns these coefficients. We then use the pre-projection activation u^ℓ as the coefficients and update with the exponential map.

3. Experimental Results

We evaluate the proposed layers on two synthetic dynamical systems: **Sphere ($\mathbb{S}^2$) dynamics** and **SO(3) dynamics**. We report mean squared error (MSE) and constraint satisfaction (norm for $\mathbb{S}^2$, determinant for $SO(3)$). We compare probabilistic baseline, and a vanilla transformer. We note that the probabilistic transformer has new version depending on if the manifold is known or not. Additionally, we do a small ablation study to understand the role of the projection. In particular, we consider two different models. One where apply the projection/exponential map after each layer, and one where we do it only at the end.[1]

Table 1: Validation results for $\mathbb{S}^2$ and SO(3).

	$\mathbb{S}^2$ (Sphere)		SO(3)	
Model	**Val Loss ± Std**	**Val Norm ± Std**	**Val Loss ± Std**	**Val Det ± Std**
Regular	0.0063 ± 0.0039	1.0018 ± 0.0050	0.1060 ± 0.0041	0.5710 ± 0.0101
Projected	$\mathbf{0.0062 \pm 0.0027}$	1.0000 ± 0.0000	0.4233 ± 0.0722	1.0000 ± 0.0000
Projected (no int)	0.0078 ± 0.0046	1.0000 ± 0.0000	0.2211 ± 0.0154	1.0000 ± 0.0000
Exponential Map	0.0136 ± 0.0377	1.0000 ± 0.0000	$\mathbf{0.2090 \pm 0.0137}$	1.0000 ± 0.0000
Exponential Map (no int)	0.0106 ± 0.0200	1.0000 ± 0.0000	0.2098 ± 0.0125	1.0000 ± 0.0000
Flow Match	0.0139 ± 0.0045	1.0023 ± 0.0012	0.3102 ± 0.0170	0.9683 ± 0.0015
Flow Match (no int)	0.0059 ± 0.0027	1.0032 ± 0.0051	0.3103 ± 0.0176	0.9684 ± 0.0016
Probabilistic (Unknown)	0.0172 ± 0.0124	1.0000 ± 0.0000	0.2451 ± 0.0093	1.0000 ± 0.0000
Probabilistic (Known)	0.0766 ± 0.0560	1.0000 ± 0.0000	0.2906 ± 0.0099	1.0000 ± 0.0000

Among models that know the manifold, we see that projected transformer does the best for the sphere, while the exponential model does the best for $SO(3)$. This is expected as the sphere is not a Lie-Algebra, hence we expect the exponential model to run into issues. On the other hand, there is not a closed form expression for the the projection onto $SO(3)$. Hence we expect that the projected model to do worse here.

Amongst models that do not know the manifold structure - Regular, Flow Match, Flow Match (no internal), and Probabilistic (Unknown), we see that Flow Match (no internal) consistently does well. In particular, it has the smallest MSE for the sphere example. While the Regular transformer has the smallest MSE for the $SO(3)$ example, we can see that the returned output signficantly violates the constraint.

Where to apply projections? Ablations indicate that frequent interlayer projections and exponential maps help only when the calculation is exact. Otherwise, operating only at the output is preferable.

Key takeaways. (i) When an exact projection exists, layering it inside the network can deliver SOTA accuracy and perfect feasibility. (ii) For group-valued targets, intrinsic exponential layers are a robust default. (iii) Learned projections are promising on the sphere but require stronger training to reach exact feasibility on $SO(3)$.

1. All code can be found at https://anonymous.4open.science/r/Constrained-Transformer-F64D/

References

Mohammed AlQuraishi. ProteinNet: a standardized data set for machine learning of protein structure. *BMC Bioinformatics*, 20:311, 2019. doi: 10.1186/s12859-019-2932-0.

Michael Burri, Janosch Nikolic, Pascal Gohl, Thomas Schneider, Joern Rehder, Sammy Omari, Markus W Achtelik, and Roland Siegwart. The euroc micro aerial vehicle datasets. *The International Journal of Robotics Research*, 2016. doi: 10.1177/0278364915620033. URL http://ijr.sagepub.com/content/early/2016/01/21/0278364915620033.abstract.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

Tim R. Davidson, Luca Falorsi, Nicola De Cao, Thomas Kipf, and Jakub M. Tomczak. Hyperspherical variational auto-encoders. In *Proceedings of the 34th Conference on Uncertainty in Artificial Intelligence (UAI)*, 2018.

Karthik Elamvazhuthi, Xuechen Zhang, Samet Oymak, and Fabio Pasqualetti. Learning on manifolds: Universal approximations properties using geometric controllability conditions for neural odes. In *Learning for Dynamics and Control Conference*, pages 1–11. PMLR, 2023.

Luca Falorsi and Patrick Forré. Neural ordinary differential equations on manifolds. *arXiv preprint arXiv:2006.06663*, 2020.

Luca Falorsi, Patrick de Haan, Tim R. Davidson, Nicola De Cao, Maurice Weiler, Patrick Forré, and Taco S. Cohen. Explorations in homeomorphic variational auto-encoding. In *ICML Workshop on Theoretical Foundations and Applications of Deep Generative Models*, 2018.

Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

Anand George, Niko Koivumäki, Teemu Hakala, Juha Suomalainen, and Eija Honkavaara. Visual-inertial odometry using high flying altitude drone datasets. *Drones*, 7(1):36, 2023. doi: 10.3390/drones7010036.

Isay Katsman, Eric Chen, Sidhanth Holalkere, Anna Asch, Aaron Lou, Ser Nam Lim, and Christopher M De Sa. Riemannian residual neural networks. *Advances in Neural Information Processing Systems*, 36:63502–63514, 2023.

Anastasis Kratsios, Behnoosh Zamanlooy, Tianlin Liu, and Ivan Dokmanić. Universal approximation under constraints is possible with transformers. In *International Conference on Learning Representations (ICLR)*, 2022.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022. doi: 10.48550/arXiv.2210.02747.

Aaron Lou, Derek Lim, Isay Katsman, Leo Huang, Qingxuan Jiang, Ser Nam Lim, and Christopher M De Sa. Neural manifold ordinary differential equations. *Advances in Neural Information Processing Systems*, 33:17548–17558, 2020.

Appendix A. Algorithms

Algorithm 1: Projected transformer layer (generic projection)

```
Input: Input $h^0 = x$; layers $\{\sigma^\ell, \theta^\ell\}_{\ell=1}^L$; target manifolds $\{\mathcal{M}_\ell \subset \mathbb{R}^{d_\ell}\}_{\ell=1}^L$ with
       projections $\{\Pi_{\mathcal{M}_\ell} : \mathbb{R}^{d_\ell} \to \mathcal{M}_\ell\}$.
Output: Output $h^L \in \mathcal{M}_L$.
for $\ell = 1$ to $L$ do
    $u^\ell \leftarrow \sigma^\ell\big(h^{\ell-1}; \theta^\ell\big)$ ;                // Pre-projection activation
    $h^\ell \leftarrow \Pi_{\mathcal{M}_\ell}\big(u^\ell\big)$ ;            // Pointwise projection back to $\mathcal{M}_\ell$
end
```

Algorithm 2: Learning a differentiable projection P_ϕ onto $\mathcal{M}$ by flow matching

```
Input: Training distribution $\mu$ on $\mathcal{M} \subset \mathbb{R}^d$; velocity prior $\nu$ (e.g., $\mathcal{N}(0, \sigma^2 I)$); horizon
       $T > 0$; step size $\Delta s$; one-step solver ODEStep (e.g., Heun/RK2).
Output: Differentiable projection $P_\phi : \mathbb{R}^d \to \mathcal{M}$.
Train time-conditioned vector field $v_\phi$;
for $iter = 1, 2, \ldots$ do
    Sample $X \sim \mu$, $V \sim \nu$, $t \sim \mathrm{Unif}[0, T]$;
    Set $Y \leftarrow X + tV$;
    Take one optimizer step on $\ell(\phi; X, V, t) = \|v_\phi(Y, t) - V\|^2$;
end
Define the projection $P_\phi$ by backward integration;
Function Project($x_0$):
    $x \leftarrow x_0$;
    for $s \leftarrow T$; $s > 0$; $s \leftarrow s - \Delta s$ do
        $x \leftarrow x + \mathrm{ODEStep}\big(-v_\phi(\cdot, s),\, x,\, \Delta s\big)$;        // Integrate $\dot{x} = -v_\phi(x, s)$
    end
    return $x$;
Use within the $\ell$-th projected transformer layer;
for $\ell = 1, \ldots, L$ do
    $u^\ell \leftarrow \sigma^\ell(h^{\ell-1}; \theta^\ell)$;
    $h^\ell \leftarrow$ Project($u^\ell$) ;                  // Apply pointwise on the minibatch
end
```

Appendix B. Synthetic Data Generation

To evaluate our models in a controlled setting, we generated synthetic trajectory data on two manifolds, the 2-sphere ($\mathbb{S}^2$) and the special orthogonal group ($SO(3)$).

Algorithm 3: Exponential transformer layer on a Lie group G

```
Input: Input $h^0 = x \in G$; layers $\{\theta^\ell\}_{\ell=1}^L$ with coefficient nets $f_{\theta^\ell} : G \to \mathbb{R}^m$; fixed basis
        $\{E_1, \dots, E_m\}$ of $\mathfrak{g} = T_e G$; step size $\Delta t > 0$.
Output: Output $h^L \in G$.
for $\ell = 1$ to $L$ do
    $c^\ell \leftarrow f_{\theta^\ell}(h^{\ell-1}) \in \mathbb{R}^m$ ;                // Learned Lie{algebra coefficients
    $\xi^\ell \leftarrow \sum_{i=1}^m c_i^\ell E_i \in \mathfrak{g}$ ;                     // Assemble algebra element
    $h^\ell \leftarrow$ ExpStep($h^{\ell-1}$, $\xi^\ell$, $\Delta t$) ;             // Intrinsic update stays in $G$
end
Function ExpStep($g \in G$, $\xi \in \mathfrak{g}$, $\Delta t$):
    return $\exp_e(\Delta t\, \xi)\, g$ ;                     // Matrix exponential if $G \subset \mathbb{R}^{d\times d}$
```

B.1. Sphere ($\mathbb{S}^2$) Dynamics

We generate smooth, random tangent vector fields on the sphere. First, four coefficients are sampled from a uniform distribution:

$$a_0, a_1, b_0, b_1 \sim \mathcal{U}(-1, 1)$$

These coefficients define two smooth functions on the sphere, parameterized by spherical coordinates (θ, ϕ):

$$a(\theta, \phi) = a_0 \sin(\theta) \cos(\phi) + a_1 \cos(2\theta)$$
$$b(\theta, \phi) = b_0 \cos(\theta) \sin(\phi) + b_1 \sin(2\phi)$$

The vector field V is constructed as a linear combination of the standard tangent basis vectors, ∂_θ and ∂_ϕ:

$$V(\theta, \phi) = a(\theta, \phi)\, \partial_\theta + b(\theta, \phi)\, \partial_\phi$$

where

$$\partial_\theta = \begin{bmatrix} \cos\theta\cos\phi \\ \cos\theta\sin\phi \\ -\sin\theta \end{bmatrix} \quad \text{and} \quad \partial_\phi = \begin{bmatrix} -\sin\phi \\ \cos\phi \\ 0 \end{bmatrix}.$$

The final vector field is normalized to have unit magnitude at every point. Trajectories are then generated by advecting initial points on the sphere along this field, using a numerical ODE solver (`solve_ivp`) to integrate the dynamics.

B.2. SO(3) Dynamics

For dynamics on the special orthogonal group $SO(3)$, we define a matrix ODE whose flow preserves the group structure. The dynamics are built using the standard basis for the Lie algebra $\mathfrak{so}(3)$:

$$B_1 = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad B_2 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix}, \quad B_3 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}$$

From these, we define a linear map $g : \mathbb{R}^{3\times3} \to \mathbb{R}^{3\times3}$ as:

$$g(X) = \sum_{i=1}^{3} B_i X$$

The evolution of a rotation matrix $X(t) \in SO(3)$ is governed by the following ODE:

$$\frac{dX}{dt} = \operatorname{tr}(X^2 + I) \cdot g(X)$$

where I is the identity matrix and $\operatorname{tr}(\cdot)$ is the trace operator. Given an initial rotation $X_0 \in SO(3)$, trajectories are generated by numerically integrating this system, either with a standard ODE solver or through discrete forward Euler steps.

Appendix C. Experimental Setup

The experimental procedure used employs a hyperparameter parsing search through a SLURM job array system that explores the training parameter space. The SLURM script implements a two-dimensional search strategy with 900 total jobs to evaluate performance across the 18 model variants for 50 randomized datasets. The job array index mapping follows:

$$\text{TASK\_ID} = \text{SLURM\_ARRAY\_TASK\_ID} - 1 \tag{5}$$

$$\text{MODEL\_INDEX} = \text{TASK\_ID} \bmod 18 \tag{6}$$

$$\text{DATASET\_ID} = \left\lfloor \frac{\text{TASK\_ID}}{18} \right\rfloor \tag{7}$$

This mapping ensures that each model variant is evaluated exactly once on each dataset of the appropriate variety. The SLURM script also standardizes model parameters, training parameters to ensure fair comparison as follows:

Table 2: Fixed architecture parameters shared across all model variants.

Parameter	**Value**
Input size	3 (sphere), 9 (SO3)
Model dimension (`d_model`)	64
Number of attention heads (`nhead`)	8
Hidden dimension (`d_hid`)	128
Number of layers (`nlayers`)	4
Dropout rate	0.1
Geometry	'sph' (sphere), 'so3' (SO3)

This systematic approach ensures that models are evaluated under identical conditions, enabling fair comparisons accross each dataset.

Table 3: Training hyperparameters used across all model variants.

Parameter	**Value**
Optimizer	AdamW
Learning rate	3×10^{-4}
Weight decay	1×10^{-4}
Learning rate scheduler	ReduceLROnPlateau
Scheduler factor	0.5
Scheduler patience	15
Gradient clipping	Maximum norm 1.0
Training epochs	500
Early stopping patience	30 epochs
Velocity network epochs	1000 (flow matching only)
Velocity network patience	100 epochs
Batch size	Dataset-dependent (32-128)
Loss function	Mean squared error (MSE)