FUNCTIONALODE: REDUCED-ORDER NEURAL DYNAMICS FOR EXTREME-DEFORMATION LAGRANGIAN SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Lagrangian methods discretize a continuum into particles whose positions and properties are tracked over time. Learned graph-based simulators model these dynamics through repeated message passing over particle interaction graphs, so every particle is updated at every step, and the connectivity becomes unstable under extreme deformation and fragmentation. We propose *FunctionalODE*, a reduced-order alternative that views the system state as a vector-valued function over the fixed reference configuration. *FunctionalODE* learns an orthonormal basis for the low-dimensional subspace occupied by the states, encodes each state by projection onto this basis, and evolves the resulting compact coefficients with a *NeuralODE*, avoiding particle-level message passing; the full state can be evaluated at arbitrary material points. On SPH simulations of hypervelocity plate impact, *FunctionalODE* reaches 98.5% R^2 on the fragmenting projectile, compared with 91.8% for the best baseline, and completes a rollout in under a second, about $100\times$ faster than graph-based simulators and the numerical solver. *FunctionalODE* could serve as a cheap surrogate for expensive hypervelocity-impact simulations and enable the integration of simulation into prediction for physical impact experiments.

1 INTRODUCTION

Learned simulators provide computationally efficient approximations to numerical methods by predicting the evolution of physical systems directly from simulation data. For Lagrangian methods such as Smoothed Particle Hydrodynamics (Gingold & Monaghan, 1977) and the Material Point Method (Sulsky et al., 1994), the state is represented by particles that move with the material, making these methods well suited to fluids, granular media, and solids undergoing large deformation.

Graph-based learned simulators (Sanchez-Gonzalez et al., 2020; Kumar & Vantassel, 2022; Brandstetter et al., 2021) typically retain this particle-level representation and model local interactions through repeated message passing. Although local graphs can scale approximately linearly with the number of particles, temporal integration still requires every particle and its neighborhood to be updated at every step. Computational and memory costs therefore remain tied to the discretization, while rapidly changing spatial neighborhoods present an additional difficulty in problems involving severe deformation, contact, separation, and fragmentation.

Hypervelocity impact provides an especially demanding example. The response involves extreme deformation, material separation, shock propagation, and fragmentation, and high-fidelity simulations can contain hundreds of thousands or millions of particles. We study a moderately sized setting of about $6{,}500$ particles, where graph-based baselines can still be trained for comparison, and design the method so that the cost of its rollout does not grow with the particle count.

We propose a reduced-order model for Lagrangian simulations that decouples the complexity of the learned dynamics from the number of sampling points. This raises two fundamental questions: how should the state of a dynamical system be represented so that its complexity is independent of the sampling points, and how should the resulting reduced dynamics be modeled? In contrast to graph-based methods, we represent the system state as a vector-valued function over the reference configuration. This functional perspective admits a compact representation whose complexity is

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

determined by the intrinsic dimensionality of the dynamics rather than the number of particles. Consequently, the evolution of the system is viewed as a trajectory of functions in a Hilbert space.

Recent works (Chen et al., 2023; Yin et al., 2023) parameterize the space of state functions as a nonlinear manifold. In contrast, *FunctionalODE* restricts the space of state functions to a learned linear subspace spanned by a finite set of basis functions. By learning the orthonormal basis functions, *FunctionalODE* obtains explicit access to both the basis and their corresponding coefficients, allowing the system state to be encoded via direct projection, rather than through specialized encoders (Chen et al., 2023) or optimization-based auto-decoding (Yin et al., 2023). Given this representation, the state of the system is fully characterized by a low-dimensional coefficient matrix, and the system dynamics can be learned entirely in this reduced space.

Our contributions are as follows:

- We cast learned Lagrangian simulation as a reduced-order problem in a Hilbert space of functions on the fixed reference configuration Ω_0, so that deformation, separation, and fragmentation do not change the domain on which the state is defined. The resulting error separates into truncation of the basis expansion and error in the learned dynamics (Section 3).
- We learn an orthonormal, affine-augmented basis with a Function Encoder and evolve its coefficients with a FiLM-conditioned NeuralODE. Encoding is a least-squares projection over a fixed number M of sampled points at a cost of $\mathcal{O}(MK^2)$; each integration step depends on K rather than on the particle count N; no neighborhood graph is built during rollout, and decoding to all N particles is needed only where the full state is queried (Sections 3.2 3.3).
- On hypervelocity plate-impact simulations, *FunctionalODE* reaches 98.48% R^2 on the fragmenting projectile, compared to 91.82% for the best baseline, and completes a rollout in under a second: about $100\times$ faster than graph-based simulators and the numerical solver, and comparable to transformer-based operators (Section 4, Table 2).

2 Related Work

Learned simulators for particle dynamics Graph neural networks have become the standard approach for learning Lagrangian particle dynamics: GNS (Sanchez-Gonzalez et al., 2020) and MeshGraphNets (Pfaff et al., 2021) learn inter-particle interactions via message passing, equivariant variants such as EGNN (Satorras et al., 2022) and SEGNN (Brandstetter et al., 2021) additionally build in symmetry constraints, and Neural SPH (Toshev et al., 2024b) stabilizes learned rollouts with solver-based corrections. These methods rebuild the interaction graph from *current* particle positions and propagate messages at every step, the connectivity itself becomes unstable under extreme deformation and fragmentation.

Reduced-order modeling Classical projection-based ROM POD-Galerkin and its descendants (Berkooz et al., 1993; Benner et al., 2015) evolve reduced coordinates in a linear subspace spanned by basis *vectors* in $\mathbb{R}^N$, tying the representation to a fixed discretization and limiting accuracy on transport-dominated problems with slowly decaying Kolmogorov n-widths (Peherstorfer, 2022). Nonlinear-manifold ROMs (Lee & Carlberg, 2020) replace the subspace with an autoencoder, and CROM (Chen et al., 2023) makes the representation continuous: a neural-field decoder, nonlinear in the latent code, trained with a discretization-dependent encoder, whose online stage advances the latent state by sampling the governing equations at a reduced set of points. Our approach shares CROM's continuous, discretization-independent perspective but differs in structure: our decoder is *linear* in the coefficients over a learned orthonormal basis, so encoding is a closed-form least-squares solve rather than an encoder network; the representation inherits Hilbert-space structure (isometry, computable truncation error); and the dynamics are learned outright rather than obtained by evaluating the governing equations online. While we retain a linear-in-coefficients expansion, the basis is learned and adapted to the solution manifold rather than fixed a priori.

Operator learning and neural fields DeepONet (Lu et al., 2021) and Fourier neural operators (Li et al., 2021) learn maps between function spaces, establishing the discretization-invariance program

our work builds on; implicit neural representations such as SIREN (Sitzmann et al., 2020) provide the continuous substrate. Closest to our setting, Neural Implicit Flow (Pan et al., 2023) learns mesh-agnostic neural-field reductions of spatio-temporal data, DINo (Yin et al., 2023) combines neural fields with data-driven latent dynamics for Eulerian PDE forecasting, and CNF-ROM (Kim et al., 2024) couples conditional neural fields with a parametric NeuralODE under a physics-informed objective. These methods encode into the latent space of a nonlinear decoder; our representation is instead linear in the coefficients of an orthonormal basis, with closed-form encoding, and we target Lagrangian SPH data rather than Eulerian grids.

Latent dynamics and function encoders Neural ODEs (Chen et al., 2019) and latent ODE models (Rubanova et al., 2019) learn continuous-time dynamics in a latent space, typically an autoencoder bottleneck, as in recent latent PDE solvers (Li et al., 2025) and latent dynamics frameworks (Farenga et al., 2024). Our latent space is instead the coefficient space of an orthonormal expansion: encoding is closed-form, latent error equals field error per channel by Parseval's identity, and the latent dimension is a truncation order with a computable error. We build directly on Function Encoders (Ingebrand et al., 2025), which learn orthonormal neural bases for function classes and have recently been extended to operator learning between Hilbert spaces (Ingebrand et al., 2024). Whereas basis-to-basis methods learn static maps between two function spaces, we learn continuous-time *dynamics* within the coefficient space of a single learned basis, applied to Lagrangian fields with an affine component for homogeneous deformation and FiLM-conditioned (Perez et al., 2017) coefficient dynamics.

3 METHODOLOGY

In this section, we formulate the problem in the language of continuum mechanics and construct the function space $\mathcal{U}$ containing the states of the system. Classical results identify $\mathcal{U}$ with the space of square-summable sequences, so that each state is represented exactly by its expansion coefficients. Our central assumption is that the trajectory of the system is well approximated within a low-dimensional subspace, so that truncating to K coefficients per channel compresses the state from $N \times d$ to $K \times d$ degrees of freedom at small error; we verify this empirically in Figure 2. The dynamics then reduce to an ordinary differential equation on the coefficients. The remainder of the section presents the two learned components: a Function Encoder that recovers the basis, and a NeuralODE that evolves the coefficients.

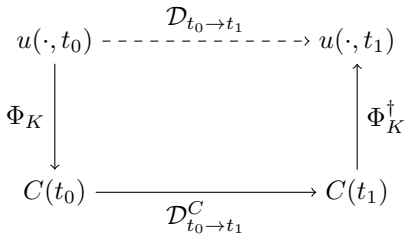


Figure 1: Correspondence between the function space $\mathcal{U}$ and the coefficient space $\mathbb{R}^{K \times d}$. The dynamics in function space (dashed) are never modeled directly; we learn the bases that defines Φ_K and the coefficient dynamics $\mathcal{D}^C_{t_0 \to t_1}$.

3.1 FORMULATION

Continuum mechanics describes a body as a continuous distribution of material particles. The *reference configuration* $\Omega_0 \subset \mathbb{R}^3$ is the bounded region the body occupies at $t = 0$; each particle is labeled by its position $X \in \Omega_0$. The *current configuration* Ω_t is the region occupied at time t: it deforms and fragments as the body evolves, while Ω_0 remains fixed for all $t \in [0, T]$. The *Lagrangian* description expresses fields as functions of the particle label on the fixed domain Ω_0; the *Eulerian* description expresses them as functions of spatial position on the changing domain Ω_t. We adopt the Lagrangian view and denote the state of the continuum by $u : \Omega_0 \times [0, T] \to V \subset \mathbb{R}^d$. For

example, the deformation $x(X,t)$ and density $\rho(X,t)$ give the position and density of the material particle initially at $X \in \Omega_0$.

Function space Consider a snapshot of the system, $u(\cdot,t)$, which is a spatial function defined on Ω_0 and contains information about position, velocity, and other physical quantities of interest. We identify a function space $\mathcal{U}$ that contains all snapshots as a direct sum of d copies of the space of square-integrable functions on Ω_0, by treating each channel of u as a separate scalar field, $u(\cdot,t) = \big(u^{(1)},\dots,u^{(d)}\big)(\cdot,t)$

$$\mathcal{U} := \bigoplus_{i=1}^{d} L^2(\Omega_0;\mathbb{R}), \tag{1}$$

where $u^{(i)}(\cdot,t) \in L^2(\Omega_0,\mathbb{R})$; each copy is a separable Hilbert space and carries a scalar inner product $\langle f,g\rangle$ [1]. It therefore admits a complete, countable orthonormal basis. For each channel i we denote the orthonormal basis by $\{\phi_k^{(i)}\}_{k\geq 1}$, which satisfies $\langle \phi_j^{(i)}, \phi_k^{(i)}\rangle = \delta_{jk}$. Each channel $u^{(i)}$ can then be expanded

$$u^{(i)}(X,t) = \sum_{k=1}^{\infty} c_k^{(i)}(t)\phi_k^{(i)}(X), \tag{2}$$

The trajectory $\{u(\cdot,t)\}_{t\in[0,T]}$ is thus encoded entirely in the coefficients $(c_k^{(i)}(t))_{k=1,2,\dots}$; the basis functions are fixed spatial modes. For each channel, define map $\Phi^{(i)} : L^2(\Omega_0;\mathbb{R}) \to \ell^2$ by

$$\Phi^{(i)}u^{(i)} := (c_1^{(i)}, c_2^{(i)}, \dots), \tag{3}$$

where $c_k^{(i)}(t) := \langle \phi_k^{(i)}, u^{(i)}(\cdot,t)\rangle$. Its inverse map $(\Phi^{(i)})^{-1} : \ell^2 \to L^2(\Omega_0;\mathbb{R})$ is

$$(\Phi^{(i)})^{-1}(c_1^{(i)}, c_2^{(i)}, \dots) := \sum_{k=1}^{\infty} c_k^{(i)}\phi_k^{(i)} \tag{4}$$

Since $\{\phi_k^{(i)}\}_{k\geq 1}$ is complete, $\Phi^{(i)}$ is an isometric isomorphism (Riesz–Fischer) by Parseval's identity [2], $\|u^{(i)}\|^2 = \sum_{k=1}^{\infty}(c_k^{(i)})^2$, so $L^2(\Omega_0;\mathbb{R}) \cong \ell^2$ per channel.

Truncated coefficient representation We assume that the trajectory $\{u(\cdot,t)\}_{t\in[0,T]}$ is well approximated within a K-dimensional subspace per channel for small K: the expansion coefficients decay rapidly, so truncation at K terms incurs negligible error. We learn the basis for which this approximation holds and verify its accuracy empirically. Linear map $\Phi^{(i)}$ equation 3 is approximated by a truncated map of $\Phi_K^{(i)}$ by collecting the leading coefficients into a column vector $C^{(i)} \in \mathbb{R}^K$. Similarly, reconstruction map $(\Phi_K^{(i)})^\dagger$ denotes approximation of inverse map $(\Phi^{(i)})^{-1}$ equation 4. Unlike $\Phi^{(i)}$, the truncated map is not invertible: the composition $(\Phi_K^{(i)})^\dagger\Phi_K^{(i)}$ is the orthogonal projection onto $\mathrm{span}\{\phi_1^{(i)},\dots,\phi_K^{(i)}\}$, with reconstruction error

$$\big\|u^{(i)} - (\Phi_K^{(i)})^\dagger\Phi_K^{(i)}u^{(i)}\big\|^2 = \sum_{k>K}\big(c_k^{(i)}\big)^2, \tag{5}$$

which vanishes as $K\to\infty$ and is small at modest K under our assumption. Finally, we construct the coefficient map across all channels, $\Phi_K : \mathcal{U} \to \mathbb{R}^{K\times d}$ by gathering per channel coefficient vector $C^{(i)}$ into a coefficient matrix $C \in \mathbb{R}^{K\times d}$, and its corresponding reconstruction map $\Phi_K^\dagger : \mathbb{R}^{K\times d} \mapsto \mathcal{U}$ in the same fashion. A trajectory in the infinite-dimensional space $\mathcal{U}$ is thus represented by a trajectory of the finite-dimensional coefficient matrix $C(t)$, and the dynamics reduce to an ordinary differential equation in t.

[1] $\langle f,g\rangle := \int_{X\in\Omega_0} f\,g, \quad f,g \in L^2(\Omega_0,\mathbb{R})$

[2] $\|u\|$ denotes the norm of function u, $\|u\|^2 = \int u^2$

Dynamics. Assuming the governing equations are well-posed, the solution operator $\mathcal{D}_{t_0 \to t_1} : \mathcal{U} \to \mathcal{U}$ evolves the state $u(\cdot, t_0)$ to $u(\cdot, t_1)$. Under the coefficient representation, this induces an operator on coefficient space, $\mathcal{D}^C_{t_0 \to t_1} : \mathbb{R}^{K \times d} \to \mathbb{R}^{K \times d}$, such that the diagram in Figure 1 commutes up to truncation and dynamic error:

$$\mathcal{D}_{t_0 \to t_1} \;\approx\; \Phi_K^\dagger \circ \mathcal{D}^C_{t_0 \to t_1} \circ \Phi_K. \tag{6}$$

Our method is summarized by equation 6: rather than modelling $\mathcal{D}_{t_0 \to t_1}$ on the full state, which requires resolving all N particles, we learn (i) the set of stacked basis functions $\{\phi_k\}_{k=1,\dots K}$, and hence the maps Φ_K and $\Phi_K^\dagger$, and (ii) the dynamics $\mathcal{D}^C_{t_0 \to t_1}$ in the compact coefficient space. In the subsequent subsections, we present the use of Function Encoders and NeuralODE to recover the coefficient map Φ_K and coefficient dynamics $\mathcal{D}^C_{t_0 \to t_1}$.

3.2 Learning The Orthonormal Basis Functions

We employ Function Encoders (Ingebrand et al., 2025) to recover the orthonormal bases from simulation data. For each channel i, the Function Encoders parameterizes the basis $\{\phi_k^{(i)}\}_{k=1}^K$ by a set of neural networks $\{\phi_k^{(i)}(\cdot\,;\Theta)\}_{k=1}^K$, where Θ collects the parameters. We parameterize each basis function by an SIREN (Sitzmann et al., 2020) for its capacity to represent high-frequency functions as the field of interest develops sharp features under fragmentation and extreme deformation.

For each snapshot $u(\cdot, t)$, the simulation provides a set of N discretization points $D := \{(X_j, u_j)\}_{j=1}^N$, where $X_j \in \Omega_0$ and $u_j := u(X_j, t) \in \mathbb{R}^d$. We suppress the time argument where it is clear from context. The Function Encoder samples from D two disjoint sets, namely an example set D_e of size M and a query set D_q.

Consider the i-th channel of the example set. By the expansion equation 2, truncated at K and evaluated at each point in the example set,

$$U_e^{(i)} := \begin{bmatrix} u_1^{(i)} \\ \vdots \\ u_M^{(i)} \end{bmatrix} \approx \begin{bmatrix} \sum_{k=1}^K c_k^{(i)} \phi_k^{(i)}(X_1) \\ \vdots \\ \sum_{k=1}^K c_k^{(i)} \phi_k^{(i)}(X_M) \end{bmatrix} = \underbrace{\begin{bmatrix} \phi_1^{(i)}(X_1) & \cdots & \phi_K^{(i)}(X_1) \\ \vdots & \ddots & \vdots \\ \phi_1^{(i)}(X_M) & \cdots & \phi_K^{(i)}(X_M) \end{bmatrix}}_{\mathbf{B}_e^{(i)} \in \mathbb{R}^{M \times K}} \underbrace{\begin{bmatrix} c_1^{(i)} \\ \vdots \\ c_K^{(i)} \end{bmatrix}}_{C^{(i)} \in \mathbb{R}^K}, \tag{7}$$

where the approximation error is the truncation error equation 5 evaluated at the sample points. Equation 7 is an overdetermined linear system ($M \geq K$); the Function Encoder solves for $C^{(i)}$ in the least-squares sense via the normal equations,

$$\hat{C}^{(i)} = \left((\mathbf{B}_e^{(i)})^\top \mathbf{B}_e^{(i)}\right)^{-1} (\mathbf{B}_e^{(i)})^\top U_e^{(i)}. \tag{8}$$

Let $\mathbf{G}^{(i)} := \frac{1}{M}(\mathbf{B}_e^{(i)})^\top \mathbf{B}_e^{(i)}$ denote the empirical Gram matrix. When the example points are sampled uniformly from Ω_0 (normalized to unit volume), $\mathbf{G}^{(i)}$ is a Monte Carlo estimate of the Gram matrix $\left[\langle \phi_j^{(i)}, \phi_k^{(i)} \rangle\right]_{jk}$, which is the identity for an orthonormal basis. Hence $\mathbf{G}^{(i)} \approx \mathbf{I}_K$, and equation 8 reduces to $\hat{C}^{(i)} \approx \frac{1}{M}(\mathbf{B}_e^{(i)})^\top U_e^{(i)}$, the empirical inner product of $u^{(i)}$ with each basis function. The approximation is exact only in the limit $M \to \infty$; in practice we solve equation 8 directly, which remains valid when $\mathbf{G}^{(i)}$ deviates from the identity. The Function Encoder then reconstructs the i-th channel on the query set,

$$\hat{U}_q^{(i)} = \mathbf{B}_q^{(i)} \hat{C}^{(i)}, \tag{9}$$

where $\mathbf{B}_e^{(i)}, \mathbf{B}_q^{(i)}$ are the design matrices of basis evaluations at the example and query points, respectively, constructed as in equation 7. The parameters are obtained by jointly minimizing the reconstruction and orthonormal loss summed over channels,

$$\Theta^\star = \arg\min_{\Theta} \sum_{i=1}^{d} \left\| U_q^{(i)} - \hat{U}_q^{(i)} \right\|_2^2 + \lambda \left\| \mathbf{G}^{(i)} - \mathbf{I}_K \right\|_F^2, \tag{10}$$

where λ is the regularization constant, $\mathbf{I}_K$ is the $K \times K$ identity matrix, and $\|\cdot\|_F$ is the Frobenius norm. The regularization term enforces the orthonormal assumption of the learned basis (Ingebrand et al., 2025).

3.3 Learning the Dynamics Operator

It remains to learn the coefficient dynamics $\mathcal{D}^C_{t_0 \to t_1}$ of equation 6. Following the Neural ODE formulation (Chen et al., 2019), we parameterize the time derivative of the latent state,

$$\dot{C} = f(C, t; \Psi) \tag{11}$$

where f is a multilayer perceptron with parameters Ψ. The dynamic operator $\mathcal{D}^C_{t_0 \to t_1}$ is realized by

$$\hat{C}(t_1) = C(t_0) + \int_{t_0}^{t_1} f(C, t; \Psi) dt. \tag{12}$$

Rather than training on one-step transitions, we fit whole rollouts. For each training trajectory n, we sample an increasing sequence of time steps $t_0 < t_1 < \cdots < t_L$, encode the corresponding snapshots into $C_n(t_0), \ldots, C_n(t_L)$ via the coefficient map Φ_K, and integrate equation 12 from $C_n(t_0)$ to every t_ℓ. The parameters are obtained by minimizing the prediction error in coefficient space accumulated along the rollout,

$$\Psi^\star = \arg\min_{\Psi} \sum_{n} \sum_{\ell=1}^{L} \left\| \hat{C}_n(t_\ell) - C_n(t_\ell) \right\|_F^2, \tag{13}$$

where $\hat{C}_n(t_\ell)$ is the result of equation 12 with t_1 replaced by t_ℓ. Because the loss is evaluated on the integrated trajectory rather than on single steps, errors that accumulate during rollout are penalized directly during training.

Dynamics modulation We use a FiLM conditioning (Perez et al., 2017) to condition the vector field on a modulation features $\eta \in \mathbb{R}^m$,

$$\dot{C} = f(C, t, \eta; \Psi) \tag{14}$$

FiLM conditions the vector field by scaling and shifting the hidden output of the dynamics function $h \mapsto \gamma(\eta) \odot h + \beta(\eta)$, where h is the output of a hidden layer in f and $\odot$ is the Hadamard product. The conditioned dynamic operator $\mathcal{D}^C_{t_0 \to t_1}|_\eta$ is realized by

$$\hat{C}(t_1) = C(t_0) + \int_{t_0}^{t_1} f(C, t, \eta; \Psi) dt. \tag{15}$$

4 Experiments

4.1 Datasets

(A) Plate–projectile impact. To compare against graph-based methods, we generate a dataset of moderately sized SPH simulations of a spherical projectile impacting a plate by *ABSTRAO* (Collé et al., 2019). The dataset consists of 192 simulations with varying impact velocity and obliquity: the impact speed is sampled uniformly in $[1.5, 5]\,\mathrm{km/s}$, and the obliquity angle uniformly in $[-\pi/6, \pi/6]\,\mathrm{rad}$. The projectile is a sphere of radius $0.15\,\mathrm{cm}$; the plate is a box of dimensions $2 \times 2 \times 0.2\,\mathrm{cm}$. The spatial resolution is $\Delta x = 5 \times 10^{-2}\,\mathrm{cm}$ and the temporal resolution

is $\Delta t = 2 \times 10^{-7}$ seconds, yielding approximately $6{,}500$ particles (projectile $N_p = 136$ and plate $N_w = 6400$ particles) per simulation over 50 time steps. We split the simulations $64/64/64$ into training, validation, and test sets; all methods, including baselines, are trained on the same 64 training simulations.

(B) 2D Navier-Stokes equations. To assess generality to the Eulerian setting, we additionally evaluate on the vorticity form of the 2D incompressible Navier-Stokes equations taken from Yin et al. (2023). The vorticity is observed on a uniform 64×64 grid over $[-1, 1]^2$ every $\Delta t = 1$ time unit. Models are trained on the horizon $[0, 19]$ and evaluated up to 39; In-t and Out-t denote errors within and beyond the training horizon. Following Yin et al. (2023), training and test trajectories share an observation grid obtained by subsampling a fraction $s \in \{5\%, 25\%\}$ of the grid nodes, and errors are reported on the full grid.

4.2 Evaluation Metrics

We evaluate the performance of models with Mean-Absolute Error (MAE), coefficient of determination (R^2), and relative L_2 error on unnormalized coordinates of the entire particle set. See Appendix C.5 for definitions.

4.3 Baselines

Lagrangian setting. We compare our method against graph-based and transformer-based methods. For graph-based methods, we include GNS (Sanchez-Gonzalez et al., 2020) and SEGNN (Brandstetter et al., 2021). GNS is the standard learned simulator for Lagrangian particle dynamics, while SEGNN provides an $E(3)$-equivariance baseline. We use LagrangeBench's implementation (Toshev et al., 2024a) for both methods, For transformer-based methods, we included ONO (Xiao et al., 2024), GNOT (Hao et al., 2023), Transolver-V3 (Zhou et al., 2026), and Galerkin Transformer (Cao, 2021). We use Neural-Solver-Library's implementation (Wu et al., 2024).

Eulerian setting. We compare our method against MP-PDE (Brandstetter et al., 2023), DeepONet (Lu et al., 2021), SIREN (Sitzmann et al., 2020), and DINo (Yin et al., 2023) on *dataset B*. These baselines span message-passing, operator-learning, and implicit-neural-representation approaches to PDE surrogates. Our method follows the evaluation protocol and is compared against baseline results taken from Yin et al. (2023, Table 2, page 7).

4.4 FunctionalODE

Lagrangian setting. We use two variants. In *variant A*, the state variable contains information on both position and velocity $u(\cdot, t) = [x \quad v](\cdot, t)$, which was implemented by sharing the basis between position and velocity fields (Appendix A.1). The position and velocity coefficients are separate channels of C that the NeuralODE predicts directly; the kinematic relation $v = \dot{x}$ is not imposed as a constraint. The coefficient matrix $C \in \mathbb{R}^{K \times 6}$, which contains 3 channels for position and 3 channels for velocity, is pushed forward in time via the learned dynamic operator $\mathcal{D}^C_{t_0 \to t_1}$ equation 12. In contrast, the state variable in *variant B* contains position $u(\cdot, t) = x(\cdot, t)$. The matrix $C \in \mathbb{R}^{K \times 3}$ is pushed forward in time by the conditioned dynamic operator $\mathcal{D}^C_{t_0 \to t_1}|_{v_0}$ equation 15, where v_0 is the initial velocity of the projectile. In both variants, each part, projectile and plate, has its own Function Encoder with K basis functions and its own example set $M = 80$ points for the projectile and $M = 512$ for the plate.

Eulerian setting Despite being formulated in the Lagrangian setting, our method applies directly to the Eulerian setting: the solution is a scalar vorticity field on the fixed domain $[-1, 1]^2$, which serves as the reference configuration. The observed grid nodes are the discretization points from which we sample the example and query sets.

Table 1: Particle's position MAE (in meter), R^2, and relative L^2 error on trajectory rollout. Baselines receive six ground-truth warm-up frames ($t = 0, \ldots, 5$), which are included in the metrics with zero error; FunctionalODE predicts all steps from the encoded state at $t = 0$. Best and second best methods are highlighted by bold font and underline respectively.

	↓ MAE (m)	↑ R^2 (%)	↓ L_2 (%)
		Overall	
GNS	3.62e−04(±6.39e−05)	88.66(±4.28)	15.38(±3.58)
SEGNN	1.82e−04(±3.90e−05)	95.23(±1.73)	10.55(±2.92)
GNOT	3.86e−04(±1.05e−04)	73.72(±4.75)	5.14(±2.54)
Galerkin Transformer	1.14e−04(±3.77e−05)	97.67(±1.05)	0.51(±0.29)
ONO	3.50e−04(±1.21e−04)	74.97(±3.13)	5.48(±3.16)
Transolver-V3	<u>1.11e−04</u>(±3.81e−05)	97.64(±1.16)	0.49(±0.29)
FunctionalODE (A)	1.50e−04(±7.93e−05)	<u>99.37</u>(±0.46)	<u>0.26</u>(±0.34)
FunctionalODE (B)	**1.04e−04**(±5.69e−05)	**99.50**(±0.38)	**0.17**(±0.22)
		Projectile	
GNS	1.83e−03(±4.35e−04)	48.20(±14.61)	52.06(±8.85)
SEGNN	1.21e−03(±3.24e−04)	70.71(±13.07)	38.34(±9.03)
GNOT	3.60e−03(±9.62e−04)	−22.01(±22.77)	79.34(±13.86)
Galerkin Transformer	7.21e−04(±1.94e−04)	91.54(±4.68)	4.55(±2.61)
ONO	3.53e−03(±1.09e−03)	−12.13(±8.83)	72.20(±3.11)
Transolver-V3	7.16e−04(±2.18e−04)	91.82(±4.81)	4.52(±2.64)
FunctionalODE (A)	<u>3.52e−04</u>(±1.72e−04)	<u>98.06</u>(±1.45)	<u>1.03</u>(±0.84)
FunctionalODE (B)	**3.13e−04**(±1.46e−04)	**98.48**(±1.08)	**0.81**(±0.62)
		Plate	
GNS	3.31e−04(±5.68e−05)	89.28(±3.20)	12.94(±3.36)
SEGNN	1.60e−04(±3.48e−05)	95.95(±1.37)	8.48(±2.49)
GNOT	3.18e−04(±8.74e−05)	81.04(±1.77)	2.93(±1.67)
Galerkin Transformer	1.01e−04(±3.49e−05)	97.66(±0.97)	0.41(±0.25)
ONO	2.83e−04(±1.01e−04)	79.95(±4.71)	3.36(±2.17)
Transolver-V3	**9.78e−05**(±3.47e−05)	97.69(±0.99)	0.39(±0.25)
FunctionalODE (A)	1.46e−04(±7.75e−05)	<u>99.34</u>(±0.45)	<u>0.25</u>(±0.33)
FunctionalODE (B)	<u>9.96e−05</u>(±5.52e−05)	**99.46**(±0.37)	**0.15**(±0.20)

5 Results

5.1 Lagrangian setting

Comparison against baselines. Table 1 demonstrates that *FunctionalODE* outperforms baseline methods on hypervelocity impact simulations on all metrics except plate MAE, where it is within 2% of *Transolver-V3*. This protocol favors the baselines, which receive six ground-truth frames scored with zero error, whereas *FunctionalODE* predicts all 50 steps from the encoded initial state. The gap is most pronounced on the projectile, where particles undergo extreme deformation and fragmentation as demonstrated numerically in Table 1 and visually in Figure 3. Particularly on the projectile, *FunctionalODE* achieves 98.48% R^2 compared to 91.82% for the best baseline (*Transolver-V3*), and other methods (*GNOT*, *ONO*) fail entirely with negative R^2 values. We attribute this improvement to a fundamental paradigm difference. Both graph-based and transformer-based methods rely on learning particle-level interaction patterns, either via message passing or attention mechanisms, which become unreliable under extreme fragmentation when interaction topologies change drastically. By contrast, *FunctionalODE* views the system's state as a function defined on Ω_0, avoiding

tracking individual particle interactions altogether, making the method inherently robust to topology changes and extreme deformations.

Coefficient decay To verify our assumption that coefficients decay rapidly, we train basis functions with $K \in \{8, 16, 24, 32, 40, 48\}$ and report reconstruction error. From equation 5, the sum of squares of truncated coefficients $\sum_{k>K} \left(c_k^{(i)}\right)^2$ equals the reconstruction error. Figure 2 demonstrates that the sum of squares of truncated coefficients diminishes rapidly as we increase the number of basis functions K, confirming that coefficient magnitudes decay quickly. The slight increase in reconstruction error for the projectile at $K \in \{40, 48\}$ is attributed to ill-conditioning of the least-squares encoding as K approaches the training set size $M = 80$.

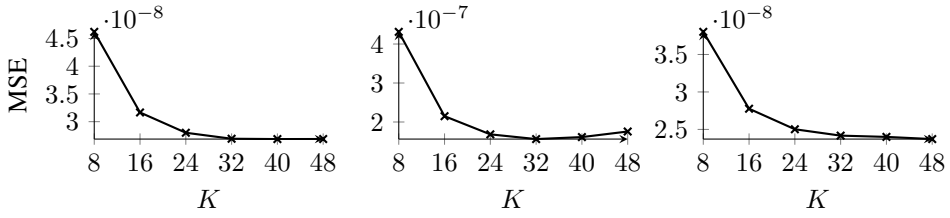


Figure 2: Reconstruction Error of particle's position by number of basis function K: Overall (*left*), Projectile (*center*), and Plate (*right*).

5.2 Eulerian setting

We test the method on 2D Navier-Stokes to explore scope beyond Lagrangian dynamics. Table 3 (Appendix) shows competitive performance at sparse sampling ($s = 5\%$) but significant underperformance at denser sampling ($s = 25\%$). Notably, our error is essentially unchanged between the two sampling ratios (test In-t: 1.01×10^{-3} vs. 0.98×10^{-3}), whereas DINo and MP-PDE improve by factors of roughly 5 and 20, respectively. The underlying vorticity field is identical in both regimes; only the number of observed points changes. This insensitivity suggests that our error is limited by the representation rather than by the observations. With a fixed number of basis functions K, every state is approximated by its orthogonal projection onto a K-dimensional linear subspace, and the truncation error of equation 5 is a bound that additional samples cannot lower. *FunctionalODE* is therefore best suited to Lagrangian simulations rather than to general Eulerian PDEs.

6 Conclusion and Discussion

We introduced *FunctionalODE*, a reduced-order framework for Lagrangian simulations that represents the system state as a function over the reference configuration rather than a set of interacting particles. By restricting this function space to a learned orthonormal basis, we obtain the state through closed-form projection and evolve its coefficients with a NeuralODE, reducing the degrees of freedom by two orders of magnitude on our impact dataset (from about $6{,}500$ particles to 32 coefficients per channel) and decoupling the cost of the dynamics from the particle count. This linear-in-coefficients construction distinguishes our approach from nonlinear-manifold methods and bridges classical POD reduced-order modeling and neural-field representations in the Lagrangian setting.

Limitations and future work. Most pressing, the inner product is defined on a single reference configuration Ω_0, which precludes the varying geometries that realistic impact-shielding studies demand; conditioning the basis on a permutation-invariant geometry code $g(\Omega_0)$ could lift this restriction. Second, the per-channel basis is not rotation-equivariant by construction; enforcing $E(3)$-equivariance, following Brandstetter et al. (2021), is a natural extension. Third, with a fixed K, the truncation error equation 5 bounds the accuracy regardless of how densely the state is observed, which limits the method on Eulerian PDEs (Section 5.2). Finally, separating representation from dynamics invites physics-structured dynamics, such as conservation constraints or Hamiltonian architectures for the coefficients.

REPRODUCIBILITY STATEMENT

We are committed to enabling reproducible research. Code for *FunctionalODE* is provided as submitted supplementary materials (supplemented code `src.zip`). Our complete implementation includes:

- JAX implementation of Function Encoder (Section 3.2) and NeuralODE dynamics (Section 3.3)
- Full dataset of 192 SPH hypervelocity plate-impact simulations used in experiments (Section 4) (Will be provided upon acceptance).
- Training and evaluation scripts with fixed random seeds for reproducibility (See `scripts/` in supplemented code).
- Complete hyperparameter specifications: basis dimension K, example set sizes ($M = 80$ projectile, $M = 512$ plate), regularization weight λ, network architectures (MLP depths/widths), and learning rates documented in Appendix C
- Jupyter notebooks reproducing all figures, tables, and ablations from the paper. (See `jaxfd/notebooks/` in supplemented code)
- Environment specifications: JAX 0.4.1+, Flax, CUDA 11.8, Python 3.10+. (See `containers/` in supplemented code)

Computational requirements: Training the full pipeline (basis learning + dynamics) on the plate-impact dataset requires approximately 6 GPU hours (V100) with batch size 32. Inference on a single 50-timestep rollout takes less than a second per trajectory. All hyperparameters and architectural choices are documented in Appendix C. We provide scripts to regenerate all results from the paper and to train new models on custom datasets.

Data availability: The SPH simulation dataset consists of particle trajectories, velocities, and densities for 192 distinct impact configurations (varying velocity in $[1.5, 5]$ km/s and obliquity in $[-\pi/6, \pi/6]$). The dataset is provided in HDF5 format with detailed metadata. The Navier-Stokes datasets are taken from the public benchmark in Yin et al. (2023).

AI USE STATEMENT

We used language models to assist with manuscript preparation, specifically for reviewing literature coverage and refining exposition. We have not used generative AI for: (a) scientific claims or novel theoretical contributions, (b) experimental design or data generation, (c) primary code implementation of FunctionalODE components. All AI-generated text was manually reviewed, verified for accuracy against the source material, and incorporated only where it improved clarity without altering technical content. We take full responsibility for all scientific claims and experimental results presented in this work.

REFERENCES

Peter Benner, Serkan Gugercin, and Karen E. Willcox. A survey of projection-based model reduction methods for parametric dynamical systems. 2015. URL http://hdl.handle.net/1721.1/100939.

G Berkooz, P Holmes, and J L Lumley. The proper orthogonal decomposition in the analysis of turbulent flows. 25(1):539–575, 1993. ISSN 0066-4189, 1545-4479. doi: 10.1146/annurev.fl.25.010193.002543. URL https://www.annualreviews.org/doi/10.1146/annurev.fl.25.010193.002543.

Johannes Brandstetter, Rob Hesselink, Elise van der Pol, Erik Bekkers, and Max Welling. Geometric and physical quantities improve e(3) equivariant message passing. 2021.

Johannes Brandstetter, Daniel Worrall, and Max Welling. Message passing neural pde solvers, 2023. URL https://arxiv.org/abs/2202.03376.

Shuhao Cao. Choose a transformer: Fourier or galerkin, 2021. URL http://arxiv.org/abs/2105.14995.

Peter Yichen Chen, Jinxu Xiang, Dong Heon Cho, Yue Chang, G A Pershing, Henrique Teles Maia, Maurizio M. Chiaramonte, Kevin Carlberg, and Eitan Grinspun. Crom: Continuous reduced-order modeling of pdes using implicit neural representations, 2023. URL https://arxiv.org/abs/2206.02607.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations, 2019. URL https://arxiv.org/abs/1806.07366.

Anthony Collé, Jérôme Limido, Thomas Unfer, and Jean-Paul Vila. An accurate sph scheme for hypervelocity impact modeling. In *2019 15th Hypervelocity Impact Symposium*, HVIS2019. American Society of Mechanical Engineers, April 2019. doi: 10.1115/hvis2019-078. URL http://dx.doi.org/10.1115/HVIS2019-078.

Nicola Farenga, Stefania Fresca, Simone Brivio, and Andrea Manzoni. On latent dynamics learning in nonlinear reduced order modeling, 2024. URL https://arxiv.org/abs/2408.15183.

Robert A Gingold and Joseph J Monaghan. Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Monthly notices of the royal astronomical society*, 181(3):375–389, 1977.

Zhongkai Hao, Zhengyi Wang, Hang Su, Chengyang Ying, Yinpeng Dong, Songming Liu, Ze Cheng, Jian Song, and Jun Zhu. GNOT: A general neural operator transformer for operator learning. 2023.

Tyler Ingebrand, Adam J. Thorpe, Somdatta Goswami, Krishna Kumar, and Ufuk Topcu. Basis-to-basis operator learning using function encoders, 2024. URL https://arxiv.org/abs/2410.00171.

Tyler Ingebrand, Adam J. Thorpe, and Ufuk Topcu. Function encoders: A principled approach to transfer learning in hilbert spaces, 2025. URL https://arxiv.org/abs/2501.18373.

Minji Kim, Tianshu Wen, Kookjin Lee, and Youngsoo Choi. Physics-informed reduced order model with conditional neural fields, 2024. URL https://arxiv.org/abs/2412.05233.

Krishna Kumar and Joseph Vantassel. Gns: A generalizable graph neural network-based simulator for particulate and fluid modeling, 2022. URL https://arxiv.org/abs/2211.10228.

Kookjin Lee and Kevin T Carlberg. Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders. *Journal of Computational Physics*, 404:108973, 2020.

Zijie Li, Saurabh Patil, Francis Ogoke, Dule Shu, Wilson Zhen, Michael Schneier, Jr. John R. Buchanan, and Amir Barati Farimani. Latent neural pde solver: a reduced-order modelling framework for partial differential equations, 2025. URL https://arxiv.org/abs/2402.17853.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations, 2021. URL https://arxiv.org/abs/2010.08895.

Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, March 2021. ISSN 2522-5839. doi: 10.1038/s42256-021-00302-5. URL http://dx.doi.org/10.1038/s42256-021-00302-5.

Shaowu Pan, Steven L. Brunton, and J. Nathan Kutz. Neural implicit flow: a mesh-agnostic dimensionality reduction paradigm of spatio-temporal data, 2023. URL https://arxiv.org/abs/2204.03216.

Benjamin Peherstorfer. Breaking the kolmogorov barrier with nonlinear model reduction. *Notices of the American Mathematical Society*, 69(5):725–733, 2022.

Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer, 2017. URL https://arxiv.org/abs/1709.07871.

Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W. Battaglia. Learning mesh-based simulation with graph networks, 2021. URL https://arxiv.org/abs/2010.03409.

Yulia Rubanova, Ricky T. Q. Chen, and David Duvenaud. Latent odes for irregularly-sampled time series, 2019. URL https://arxiv.org/abs/1907.03907.

Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter W. Battaglia. Learning to simulate complex physics with graph networks, 2020. URL https://arxiv.org/abs/2002.09405.

Victor Garcia Satorras, Emiel Hoogeboom, and Max Welling. E(n) equivariant graph neural networks, 2022. URL https://arxiv.org/abs/2102.09844.

Vincent Sitzmann, Julien N. P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions, 2020. URL https://arxiv.org/abs/2006.09661.

D. Sulsky, Z. Chen, and H. L. Schreyer. A particle method for history-dependent materials. 118(1):179–196, 1994. ISSN 0045-7825. doi: https://doi.org/10.1016/0045-7825(94)90112-0. URL https://www.sciencedirect.com/science/article/pii/0045782594901120.

Artur Toshev, Gianluca Galletti, Fabian Fritz, Stefan Adami, and Nikolaus Adams. Lagrangebench: A lagrangian fluid mechanics benchmarking suite. *Advances in Neural Information Processing Systems*, 36, 2024a.

Artur P. Toshev, Jonas A. Erbesdobler, Nikolaus A. Adams, and Johannes Brandstetter. Neural sph: Improved neural modeling of lagrangian fluid dynamics, 2024b. URL https://arxiv.org/abs/2402.06275.

Haixu Wu, Huakun Luo, Haowen Wang, Jianmin Wang, and Mingsheng Long. Transolver: A fast transformer solver for PDEs on general geometries, 2024. URL http://arxiv.org/abs/2402.02366.

Zipeng Xiao, Zhongkai Hao, Bokai Lin, Zhijie Deng, and Hang Su. Improved operator learning by orthogonal attention, 2024. URL http://arxiv.org/abs/2310.12487.

Yuan Yin, Matthieu Kirchmeyer, Jean-Yves Franceschi, Alain Rakotomamonjy, and Patrick Gallinari. Continuous pde dynamics forecasting with implicit neural representations, 2023. URL https://arxiv.org/abs/2209.14855.

Hang Zhou, Haixu Wu, Haonan Shangguan, Yuezhou Ma, Huikun Weng, Jianmin Wang, and Mingsheng Long. Transolver-3: Scaling up transformer solvers to industrial-scale geometries, 2026. URL https://arxiv.org/abs/2602.04940. Version Number: 2.

A APPENDIX: METHODOLOGY EXTENDED

A.1 LEARNING THE ORTHONORMAL BASIS FUNCTIONS (EXTENDED)

Justification for sampling. Equation equation 7 justifies sampling the set of discretization points. The coefficient matrix C approximately characterizes the snapshot function u. For each channel, we solve for $C^{(i)}$ by solving a system of M linear equations. Therefore, we only need to sample M out of N points, with $K < M \leq N$ for the system of linear equations to be overdetermined; M is chosen by the number of basis functions rather than by the resolution, and stays fixed as N grows. A similar argument appears in Chen et al. (2023). The cost of encoding a snapshot to a coefficient matrix is $\mathcal{O}(M \times K^2)$, independent of N. Decoding the full state at all N particles costs $\mathcal{O}(N \times K)$

per channel, but it is needed only at the times where the full state is queried, not at every integration step. In our experiments M is a substantial fraction of N for the projectile ($M = 80$, $N = 136$) and about 8% for the plate ($M = 512$, $N = 6{,}400$).

Per-part bases. The reference configuration is the disjoint union of two material bodies, the projectile and the plate, $\Omega_0 = \Omega_0^{\mathrm{prj}} \cup \Omega_0^{\mathrm{plt}}$. We exploit this structure and learn a separate Function Encoder per part; this allows the discontinuity at the material interface, and enables the number of basis functions and example points to be chosen per part. The coefficient state passed to the dynamics model is the concatenation of the two parts' coefficients.

Affine Basis Augmentation. We consider a deformation function $x(\cdot, t)$, a central concept in continuum mechanics, which describes the current position of each particle. In the homogeneous deformation regime, where only translation, rotation, and stretch are present, the function x can be parameterized by an affine transformation: $x(X, t) = b(t) + A(t)X$, with $b(t) \in \mathbb{R}^3$ representing translation and $A(t) \in \mathbb{R}^{3\times 3}$ representing rotation and stretch. To capture this behavior, we introduce an inductive bias by augmenting the basis functions with an affine transformation,

$$x^{(i)} = b_i + A_i^\top X + \sum_{k=1}^{K} c_k^{(i)} \phi_k^{(i)}, \qquad i = 1, 2, 3. \tag{16}$$

The augmentation is implemented by augmenting the design matrix $\mathbf{B}^{(i)}$ defined in equation 7,

$$\tilde{\mathbf{B}}^{(i)} := \begin{bmatrix} 1 & X_1^{(1)} & X_1^{(2)} & X_1^{(3)} & \phi_1^{(i)}(X_1) & \cdots & \phi_K^{(i)}(X_1) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & X_M^{(1)} & X_M^{(2)} & X_M^{(3)} & \phi_1^{(i)}(X_M) & \cdots & \phi_K^{(i)}(X_M) \end{bmatrix} \tag{17}$$

The coefficient (b, A, C) is jointly solved via least squares. Note that the orthonormality constraint applies only to the learned basis functions $\{\phi_k^{(i)}\}$: the affine columns $1, X^{(1)}, X^{(2)}, X^{(3)}$ are in general not orthogonal to them, so the Gram matrix of $\tilde{\mathbf{B}}^{(i)}$ is not the identity and the coefficients (b, A, C) are obtained from the full normal equations equation 8 with $\mathbf{B}^{(i)}$ replaced by $\tilde{\mathbf{B}}^{(i)}$, rather than from the empirical inner product. The augmentation allows us to account for homogeneous deformation in addition to the fragmentation regime. Function x is now characterized by the tuple (b, A, C).

Sharing basis between position and velocity fields Taking the derivative of both sides of the equation equation 16 with respect to time gives us the velocity vector field,

$$v^{(i)}(X, t) := \dot{x}^{(i)}(X, t) = \dot{b}_i + \dot{A}_i^\top X + \sum_{k=1}^{K} \dot{c}_k^{(i)}(t) \phi_k^{(i)}(X), \qquad i = 1, 2, 3. \tag{18}$$

Therefore, the velocity vector field has to lie within the space spanned by the set of basis functions $\{1, X^{(1)}, X^{(2)}, X^{(3)}, \phi_1^{(i)}, \ldots, \phi_K^{(i)}\}$ for each channel i. This motivates sharing the basis between position and velocity. We use the shared basis only to represent both fields: in variant A, the velocity coefficients are predicted directly by the NeuralODE as separate channels, rather than obtained by differentiating the position coefficients in time, so $v = \dot{x}$ holds only approximately, to the extent the learned dynamics are consistent with it.

B APPENDIX: MORE RESULTS

B.1 LAGRANGIAN SETTING

Qualitative comparison against baselines.

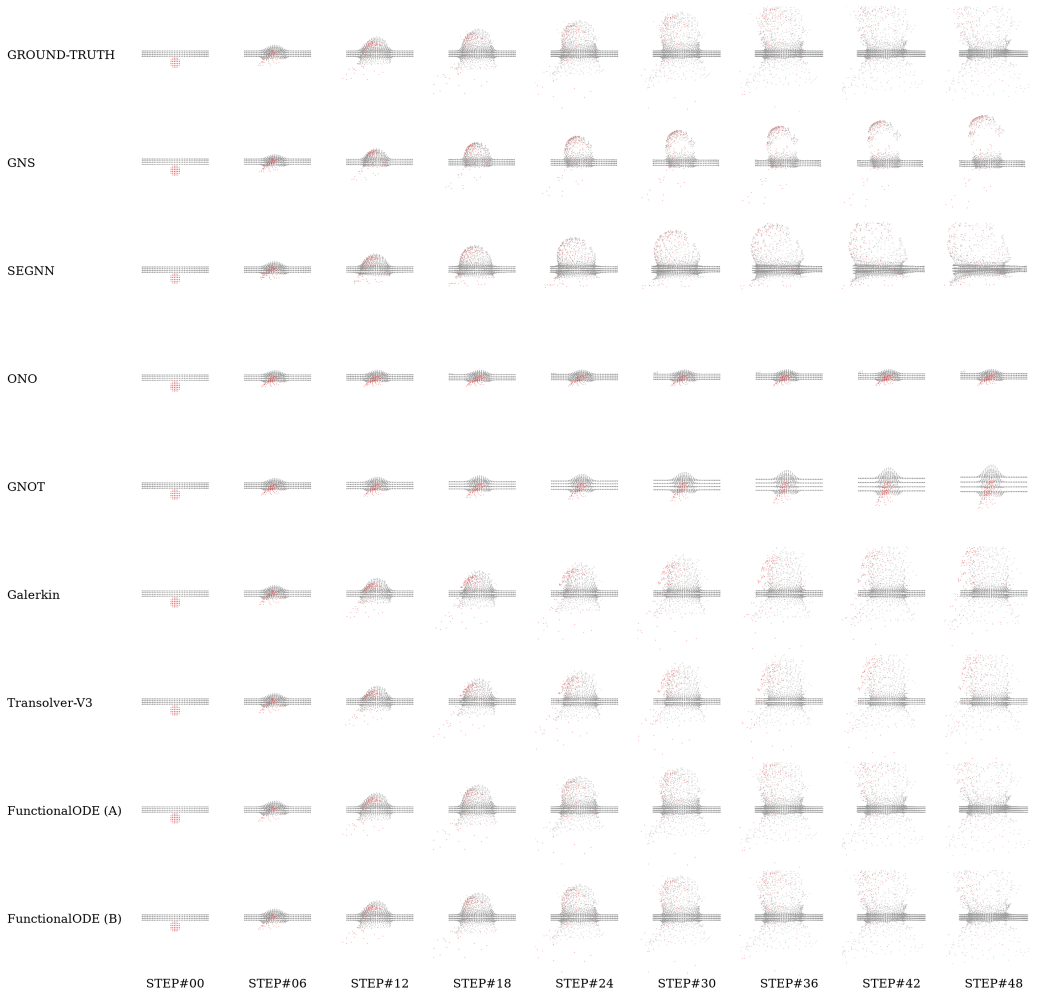


Figure 3: Visual rollout comparison of a sample taken from *dataset A* A full-sized resolution image is provided in supplementary materials.

Rollout Error Figure 4 plots the rollout error per time-step for different methods. The figure shows that the error accumulates faster for GNS and SEGNN compared to our method. We attribute this difference to the advantage that comes with the NeuralODE formulation. GNS and SEGNN's training procedure relies on 1-step-ahead prediction. Our method samples a random subsequence of time steps per trajectory, encodes the subsequence to coefficient space, and learns the dynamics on the entire sequence, hence more resistant to error accumulation.

Complexity Table 2 compares inference time for all approaches by measuring the wall-clock time taken to compute a simulation trajectory. Number was calculated on test set taken from dataset A on a single V100 GPU for all approaches. Our method is about $100\times$ faster than the graph-based methods and the numerical solver, and comparable to the transformer-based operators, while being substantially more accurate on the fragmenting projectile (Table 1). Notably, the graph-based simulators are slower than the numerical solver they are meant to replace, since they update every particle and its neighborhood at every step.

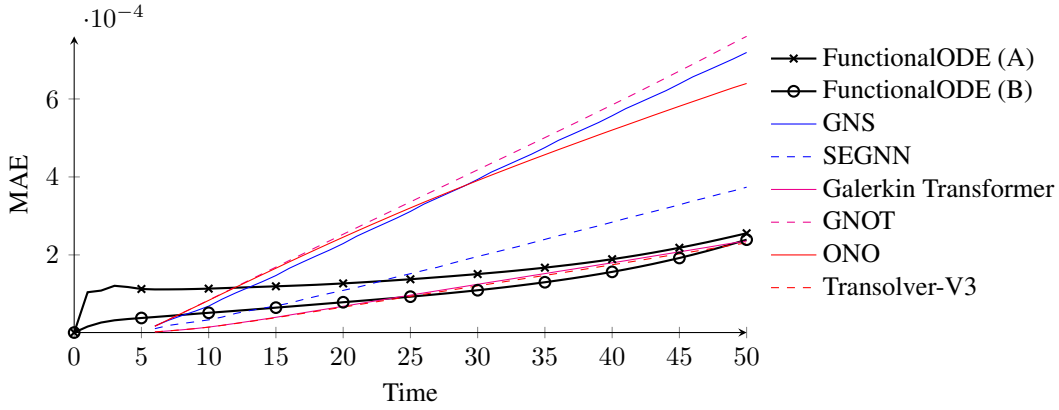


Figure 4: Rollout error by time step. Baselines are initialized with six ground-truth frames ($t = 0, \ldots, 5$) as input history and begin predicting at $t = 6$; their curves are therefore shown from $t = 5$. FunctionalODE is initialized from the encoded state at $t = 0$ only, so its curves include the reconstruction error of the initial encoding.

	# Params (million)	Inference time (second)
Simulator (ABSTRAO)	–	$35.56(\pm 3.84)$
GNS	1.21	$61.06(\pm 4.28)$
SEGNN	1.45	$59.10(\pm 3.55)$
GNOT	4.87	$1.23(\pm 0.05)$
Galerkin Transformer	1.10	$0.38(\pm 0.04)$
ONO	1.60	$1.67(\pm 0.10)$
Transolver-V3	0.97	$0.57(\pm 0.04)$
FunctionalODE (A)	2.83	$0.65(\pm 0.003)$
FunctionalODE (B)	2.64	$0.40(\pm 0.002)$

Table 2: Model's size and inference time (s/simulation). Inference time was measured on the dataset A 4.1 on a single V100 GPU for all approaches.

B.2 EULERIAN SETTING

Model	Train		Test	
	In-t	Out-t	In-t	Out-t
	$s = 5\%$ subsampling ratio			
I-MP-PDE	8.154e−3	8.166e−3	7.926e−3	8.225e−3
DeepONet	3.330e−3	7.370e−3	1.346e−2	1.408e−2
SIREN	8.741e−3	1.767e−1	4.303e−2	2.126e−1
DINo	1.029e−3	**1.655e−3**	1.326e−3	**1.813e−3**
OUR	**8.521e−4**	2.044e−3	**1.009e−3**	1.913e−3
	$s = 25\%$ subsampling ratio			
I-MP-PDE	3.135e−4	7.245e−4	3.476e−4	7.658e−4
DeepONet	9.016e−4	5.936e−3	9.376e−3	1.328e−2
SIREN	5.180e−3	2.175e−1	2.436e−1	3.861e−1
DINo	**1.020e−4**	**4.504e−4**	**2.646e−4**	**5.951e−4**
OUR	8.244e−4	1.993e−3	9.836e−4	1.860e−3

Table 3: MSE on Navier-Stokes (dataset B) on train and test trajectories, within (In-t, $[0, 19]$) and beyond (Out-t, $(19, 39]$) the training horizon, for subsampling ratios s. Baseline results are taken from Yin et al. (2023, Table 2).

C APPENDIX: FUNCTIONALODE IMPLEMENTATION DETAILS AND HYPERPARAMETERS

C.1 FUNCTION ENCODER CONFIGURATION

Stage-1 basis learning uses separate SIREN-based Function Encoders for projectile and plate: The affine augmentation for the plate augments the design matrix with bias and linear terms ($[\mathbf{1}, X, \Phi]$), while the projectile uses least-squares projection onto the basis.

Table 4: Function Encoder hyperparameters for plate-impact experiments.

Parameter	**Projectile**	**Plate**
Basis functions (K)	32	32
Example set size (M)	80	512
Temporal samples per trajectory	7	7
Basis type	SIREN	SIREN
Network width	64	64
Network depth	5	5
Input features	3 (coordinates)	3 (coordinates)
Output features	3 (position)	3 (position)
SIREN first ω	30.0	30.0
SIREN hidden ω	30.0	30.0
Coefficient method	least_square	affine
Orthonormality regularization	$\lambda = 10^{-2}$	

C.2 NEURALODE DYNAMICS CONFIGURATION

Stage-2 learns the coefficient dynamics $\dot{C} = f(C, t; \eta)$ where η is the modulation feature (initial projectile velocity):

FiLM conditioning applies the modulation features via affine scaling and shifting in the hidden layers to adapt the vector field to different initial conditions.

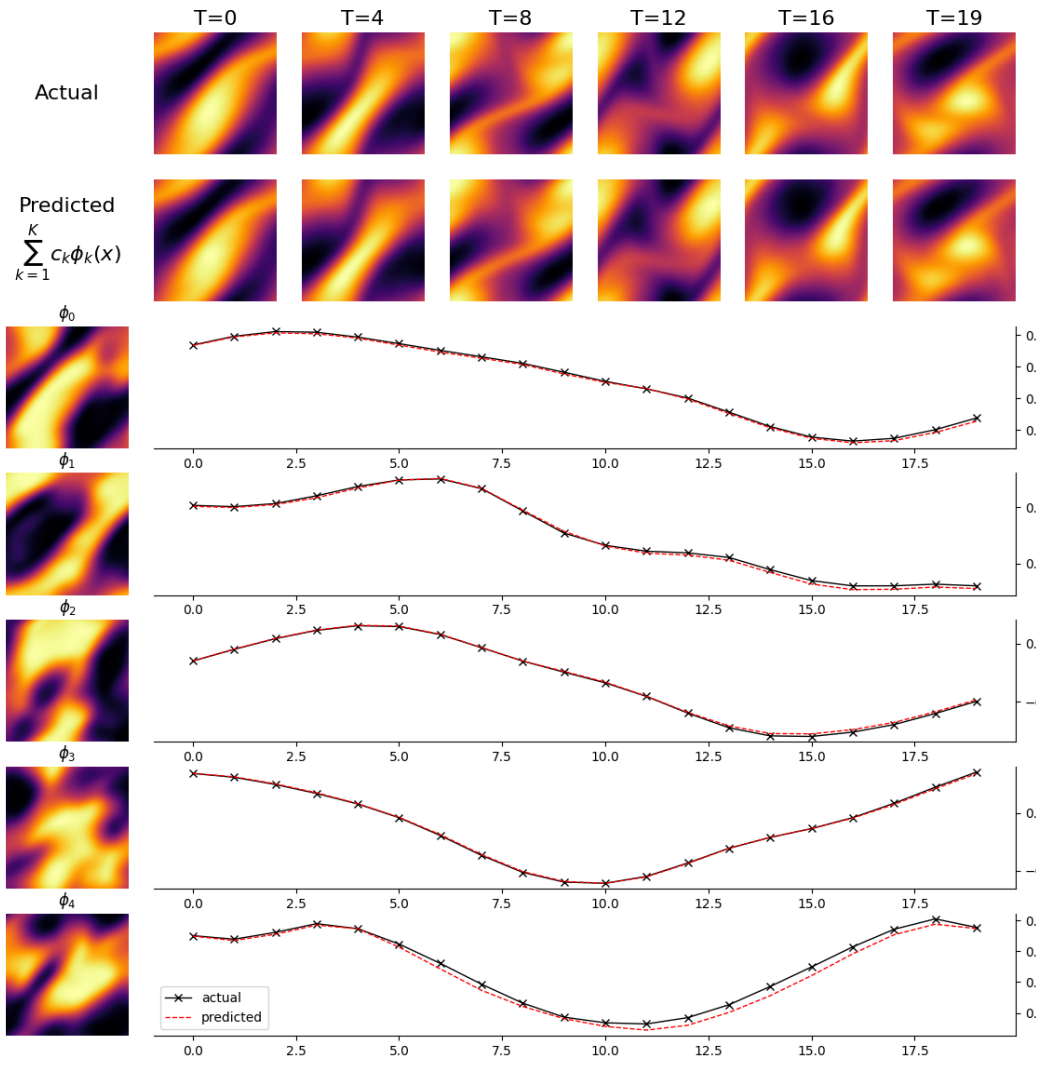


Figure 5: (*top*) Visual rollout comparison of a sample taken from *dataset B* in 5% subsampling ratio regime; (*bottom*) 5 basis functions with its corresponding coefficients.

C.3 Training Procedure

The two-stage pipeline trains basis (Stage-1) and dynamics (Stage-2) sequentially:

Stage-1: Basis Learning (Function Encoder)

- Number of epochs: 768 (per config)
- Batch size: 32 trajectories
- Loss: reconstruction MSE + orthonormality regularization (Eq. equation 10)
- $\lambda = 10^{-2}$ (Gram orthonormality)
- Learning rate schedule: exponential decay with $\eta_0 = 10^{-3}$, boundaries at 256/512 steps, scale 0.25
- Optimizer: Adam (no weight decay)
- Temporal sampling: 7 snapshots per trajectory

Table 5: NeuralODE hyperparameters for coefficient dynamics.

Parameter	Value
Input dimension	64 (32 coeff $\times$ 2 body parts)
Modulation features (η)	3 (initial velocity components)
Hidden layers	5
Hidden layer width	512
Activation function	SwiGLU (swish gated linear unit)
Output dimension	64
ODE solver (training)	JAX diffrax Dopri8
ODE solver (inference)	JAX diffrax Dopri8
Time step discretization	$\Delta t = 10^{-2}$
Learning rate schedule	exponential decay: $\eta_0 = 10^{-3}$, boundaries at 256/512 steps
Optimizer	Adam
Batch size	32
Number of epochs	768
Gradient clipping	none

- Spatial sampling: 80 points (projectile), 512 points (plate)
- PRNG seed: 7 (basis), 11 (dynamics)

Stage-2: Dynamics Learning (NeuralODE)

- Number of epochs: 768 (per config)
- Batch size: 32 trajectories
- Loss: MSE in coefficient space (Eq. equation 13)
- Learning rate schedule: same as Stage-1 (exponential decay $\eta_0 = 10^{-3}$)
- Optimizer: Adam
- ODE solver: JAX diffrax Dopri8 (8th-order Dormand–Prince Runge–Kutta, adaptive stepping)
- Time discretization: $\Delta t = 10^{-2}$
- Hardware: NVIDIA V100 GPU (16GB memory)
- Typical training time: 5h 51m on a single V100 GPU (both stages combined; Table 6)
- Checkpoint strategy: best validation checkpoint (per config)

Global Settings

- Random seeds for reproducibility: (basis=7, dynamics=11)
- Number of workers for data loading: 8
- Max files cached: 64
- Data augmentation: random trajectory sampling during training

C.4 Data Preprocessing

Data preprocessing is applied per-part (projectile and plate separately) using empirical statistics:

- **Coordinate normalization:** Particle positions mapped to reference domain $[-1, 1]^3$ using computed bounds
- **Velocity normalization:** Velocities scaled by computed per-component standard deviation (approximate max ~ 5 km/s)
- **Statistics computation:** Welford online algorithm over entire training dataset (mean, variance, min, max)

- **Spatial sampling:** Random stratified sampling of example points ($M = 80$ projectile, $M = 512$ plate) per trajectory
- **Temporal sampling:** Uniform sampling of 7 snapshots per trajectory over the 50-step rollout
- **Train/validation/test split:** 64 / 64 / 64 trajectories
- **Batch construction:** Trajectories sampled uniformly during training; no ordering bias

C.5 Evaluation Metrics

Given a ground-truth function $u : \Omega \to \mathbb{R}$ and a prediction $\hat{u} : \Omega \to \mathbb{R}$, observed at N discretization points $\{X_j\}_{j=1}^N$ with $u_j := u(X_j)$ and $\hat{u}_j := \hat{u}(X_j)$, we define the following error measures:

$$\text{MSE}(u, \hat{u}) := \frac{1}{|\Omega|} \int_\Omega \big(u(X) - \hat{u}(X)\big)^2 \, dX \approx \frac{1}{N} \sum_{j=1}^N (u_j - \hat{u}_j)^2,$$

$$\text{MAE}(u, \hat{u}) := \frac{1}{|\Omega|} \int_\Omega |u(X) - \hat{u}(X)| \, dX \approx \frac{1}{N} \sum_{j=1}^N |u_j - \hat{u}_j|,$$

$$R^2(u, \hat{u}) := 1 - \frac{\text{MSE}(u, \hat{u})}{\text{MSE}(u, \bar{u})}, \qquad \bar{u} := \frac{1}{|\Omega|} \int_\Omega u(X) \, dX \approx \frac{1}{N} \sum_{j=1}^N u_j,$$

$$\text{Rel-}L^2(u, \hat{u}) := \frac{\|u - \hat{u}\|}{\|u\|} \approx \left(\frac{\sum_{j=1}^N (u_j - \hat{u}_j)^2}{\sum_{j=1}^N u_j^2} \right)^{1/2}.$$

For vector-valued functions, MSE and MAE are averaged across the output channels, R^2 is averaged across channels weighted by the variance $\text{MSE}(u^{(i)}, \bar{u}^{(i)})$ of each channel, i.e. $R^2 = 1 - \sum_i \text{MSE}(u^{(i)}, \hat{u}^{(i)}) / \sum_i \text{MSE}(u^{(i)}, \bar{u}^{(i)})$, and the relative L^2 error uses the norm summed over channels.

In the Lagrangian setting, the metrics are computed on the unnormalized particle positions over the full rollout (50 time steps) of each test trajectory, and Table 1 reports their mean and standard deviation across the 64 test trajectories; R^2 and the relative L^2 error are reported in percent. In the Eulerian setting, we report MSE following Yin et al. (2023).

D Appendix: Baselines Implementation and Hyperparameters

This section documents the hyperparameter configurations used for training all baseline methods on the small-plate hypervelocity impact experiments. All baselines were trained using either the LagrangeBench or Neural Solver Library framework with identical data preprocessing, splitting (64/64/64 train/val/test), and evaluation protocols.

D.1 Graph-based methods

Graph-based methods consist of

- GNS (Sanchez-Gonzalez et al., 2020)
- SEGNN (Brandstetter et al., 2021)

For both methods, the input is a graph over particles at the current timestep; where the edges are built from a radius graph, computed from most recent position, the nodes features are histories of 5 finite difference velocities computed from a window of 6 previous frames. We follow the training strategy from Toshev et al. (2024a), where random-walk noise and push-forward tricks are added led to best long-term rollout's accuracy. Both methods have 10 layers of message passing and latent dimension of 128. For details, please see `scripts/baselines/train_gns.sh` and `scripts/baselines/train_segnn.sh`.

D.2 TRANSFORMER-BASED METHODS

Transformer-based methods consist of

- GNOT (General Neural Operator Transformer) (Hao et al., 2023),
- Galerkin Transformer (Cao, 2021),
- Transolver-V3 (Transformer Solver) (Zhou et al., 2026),
- Improved Operator Learning by Orthogonal Attention (Xiao et al., 2024).

Input is the current particle's position x; particle features are the history of 5 finite different velocity computed from previous 6 frames, and one-hot encoded particle type. Model's prediction is the next-step displacement. We follow the standard `dynamic_autoregressive` training procedure from Neural Solver Library. The baseline methods used an AdamW optimizer with learning rate of 10^{-3} and weight decay of 10^{-5} and gradient clipping with global norm of 1.0. All methods have hidden dimension of 128 with depth of 8 blocks, and 8 attention heads in each block's attention layers. For details, please see `scripts/baselines/`,

E APPENDIX: TRAINING BUDGET COMPARISON

Table 6: Training budget comparison between different methods.

	Wall time (hour)	Epochs	Gradient steps / epochs	Gradient steps
GNS	14h 21m	–	–	20.0K
SEGNN	14h 20m	–	–	20.0K
ONO	7h 17m	40	720	28.8K
Galerkin Transformer	5h 00m	40	720	28.8K
GNOT	6h 26m	40	720	28.8K
Transolver-V3	5h 56m	40	720	28.8K
FunctionalODE	5h 51m	$768 + 768$	2	3.1K

Table 6 reports the training budget for each method. All experiments were run on a single V100 GPU. The reported number for FunctionalODE includes 768 epochs to train the basis functions and 768 epochs to train the dynamic model.