Learning Lagrangian Interaction Dynamics with Sampling-Based Model Order Reduction

Anonymous authors
Paper under double-blind review

Abstract

Simulating physical systems governed by Lagrangian dynamics often entails solving partial differential equations (PDEs) over high-resolution spatial domains, leading to significant computational expense. Reduced-order modeling (ROM) mitigates this cost by evolving low-dimensional latent representations of the underlying system. While neural ROMs enable querying solutions from latent states at arbitrary spatial points, their latent states typically represent the global domain and struggle to capture localized, highly dynamic behaviors such as fluids. We propose a sampling-based reduction framework that evolves Lagrangian systems directly in physical space, over the particles themselves, reducing the number of active degrees of freedom via data-driven neural PDE operators. To enable querying at arbitrary spatial locations, we introduce a learnable kernel parameterization that uses local spatial information from time-evolved sample particles to infer the underlying solution manifold. Empirically, our approach achieves a 6.6×–32× reduction in input dimensionality while maintaining high-fidelity evaluations across diverse Lagrangian regimes, including fluid flows, granular media, and elastoplastic dynamics. We refer to this framework as GIOROM (**G**eometry-**I**nf**O**rmed **R**educed-**O**rder **M**odeling).

1 Introduction

Various physical simulations involve simulating the spatio-temporal evolution of continuous fields governed by partial differential equations (PDEs) of the form

$$\boldsymbol{\mathcal{J}}(\boldsymbol{f}, \boldsymbol{\nabla} \boldsymbol{f}, \boldsymbol{\nabla}^2 \boldsymbol{f}, \ldots, \dot{\boldsymbol{f}}, \ddot{\boldsymbol{f}}, \ldots) = \boldsymbol{0}, \tag{1}$$

$$\boldsymbol{f}(\boldsymbol{X}, t) : \Omega \times \mathcal{T} \to \mathbb{R}^d \tag{2}$$

where $\boldsymbol{f}$ represents a multidimensional continuous vector field that depends on both space and time (e.g., position field, velocity field, etc.). The symbols $\boldsymbol{\nabla}$ and $\dot{(\cdot)}$ signify the spatial gradient and time derivative, respectively. Here, $\Omega \subset \mathbb{R}^d$ and $\mathcal{T} \subset \mathbb{R}$ denote the spatial and temporal domains, respectively. To solve such a system, we define the solution operator $\boldsymbol{\mathcal{J}} : \boldsymbol{a} \to \boldsymbol{f}$, that maps a function of known observations $\boldsymbol{a} : \Omega' \times \mathcal{T}' \to \mathbb{R}^{d'}$ to $\boldsymbol{f}$, where we assume $\boldsymbol{a}$ and $\boldsymbol{f}$ to lie on Banach function spaces defined on bounded domains $D' \in \mathbb{R}^{d'}$ and $D \in \mathbb{R}^d$ respectively. We let $\mathcal{M}$ denote the manifold of solutions over all parameters and time and d denotes the dimension (2 or 3).

To numerically solve the equation, the system is discretized spatially and temporally. Spatial discretization allows for the approximations $\boldsymbol{f}(\boldsymbol{X}, t) \approx \boldsymbol{f}_P(\boldsymbol{X}, t)$, and, $\boldsymbol{a}(\boldsymbol{X}, t) \approx \boldsymbol{a}_P(\boldsymbol{X}, t)$, where P represents the P-point spatial discretization in $\mathbb{R}^d$, transforming $\boldsymbol{X}$ to a $(P \cdot d)$-dimensional vector. Similarly, we introduce temporal samples $\{t_n\}_{n=1}^{T}$, so that, for a given sample $x \in \{x^i\}_{i=1}^{P} \in \mathbb{R}^{P \cdot d}$, we can compute the spatiotemporal evolution using a set of input state variables $\boldsymbol{a}_t = \boldsymbol{a}_P(x, t)$ to obtain the solution $\boldsymbol{f}_{t+1} = \boldsymbol{f}_P(x, t+1)$ at the next state.

In this work, we focus on **particle interaction dynamics** modeled using Lagrangian systems Zefran & Bullo (2005), in which the temporal evolution of interacting particles (represented as point-clouds) is governed by equations of motion derived from a Lagrangian functional encoding kinetic and potential energy contributions.

It has been shown that the computational cost of solving $\mathcal{J}$ scales with the resolution P, making it computationally expensive to use full-order PDE solvers, which operate directly on $P \cdot d$ degrees of freedom Chen et al. (2023; 2021).

Reduced order modeling approaches are used to overcome this computational bottleneck. Traditional ROMs achieve computational speedup by projecting high dimensional state variables onto low dimensional latent subspaces, typically via methods like Proper Orthogonal Decomposition (POD) or reduced basis techniques. This enables approximation of these dynamics on fewer degrees of freedom (e.g. $Q \ll P \cdot d$), reducing the computational cost Lucia et al. (2004). These approaches operate under an offline-online paradigm: the expensive offline phase constructs a global basis from full-order simulations, and the inexpensive online phase evolves low-dimensional coefficients spatio-temporally using reduced forms of governing equations. To recover the solution field from the evolved latents, modern ROM approaches typically leverage continuous implicit neural representations such as neural fields. While effective in many regimes, these global latent representations often fail to capture complex, fast-evolving local dynamics within dynamic systems such as fluids, because of their lack of locality. Furthermore these approaches are not fully data-driven. They still rely on access to the underlying governing equations for spatiotemporal evolution.

To address this, we propose a new ROM framework that replaces projection-based reduction with a more direct, **sampling-based reduction** that enables reduced evolution directly in physical space. Specifically, we evolve the spatio-temporal dynamics of a small set of representative particles (samples) embedded in the physical domain, bypassing the need for a global latent state. This evolution can be facilitated using data-driven neural Lagrangian solvers Sanchez-Gonzalez et al. (2020), reducing dependencies on explicit forms of governing equations.

Furthermore, this direct "particle-space" evolution preserves locality while also reducing computational complexity by minimizing the number of active degrees of freedom (i.e. number of particles processed) during the evolution. This allows for a distinct advantage in constructing the solution manifold: discretization invariant kernel-based manifold parameterization can be defined over the evolved particle sample set to query arbitrary points, thus retaining the modular approach of traditional ROMs without sacrificing the locality of the physical space.

The framework remains **fully agnostic** to the underlying neural operator architecture, enabling it to exploit the strengths of established neural physics models that accurately model particle dynamics and generalize across PDE parameterizations. Through reduced sampling of particle sets, GIOROM further enhances these models, leveraging their robustness to discretization variations to improve their inference speeds.

Thus, our proposed GIOROM framework introduces a fully data driven ROM framework that retains essential ROM characteristics: modularity, reduced-order evolution, and continuous implicit parameterization of the solution field, while performing time-stepping directly in physical space. The formulation remains independent of the underlying discretization and employs a locally defined kernel operator to parameterize the solution field.

2 Related Works

Reduced-order models ROMs reduce computational cost by projecting high-dimensional systems onto low-dimensional manifolds Berkooz et al. (1993); Holmes et al. (2012); Lee & Carlberg (2020); Peherstorfer (2022), often via sample selection An et al. (2008). These low dimensional states are explicitly and **independently** evolved using time-steppers. Neural field-based ROMs Pan et al. (2023); Yin et al. (2023); Wen et al. (2023); Chen et al. (2023); Chang et al. (2023) enable discretization-agnostic learning, supporting generalization across geometric discretizations, but are still intrusive, requiring exact PDE formulation to time-step. Data-driven reduced-order modeling use deep-learning based techniques to achieve order-reduction Guo & Hesthaven (2019); Peherstorfer & Willcox (2015); Besabe et al. (2025), but these are however, not discretization invariant and require retraining to adapt to different discretizations. Kernel-based ROM techniques Honeine (2011); Kou & Zhang (2017); Raveh (2001); Munteanu et al. (2005); Salvador et al. (2021) leverage kernel methods such as Kernel-PCA or Kernel-POD, but they are discretization-dependent. Additional works are in section B.

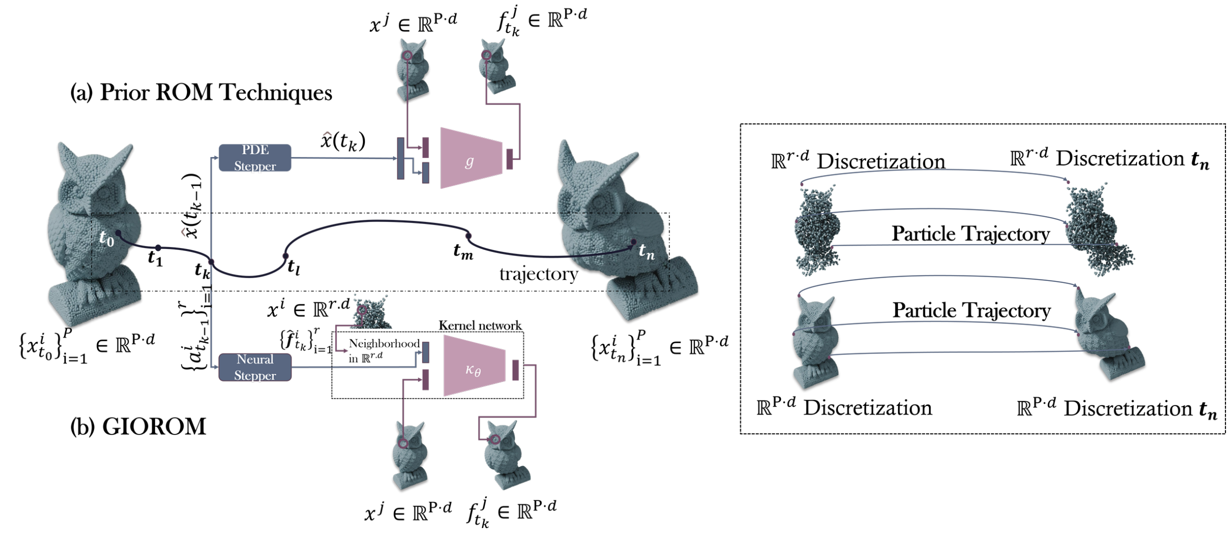

(a) Prior ROM Techniques
(b) GIOROM
PDE Stepper
Neural Stepper
Kernel network
Neighborhood in $\mathbb{R}^{r \cdot d}$
$\hat{x}(t_k)$
$\hat{x}(t_{k-1})$
$x^j \in \mathbb{R}^{P \cdot d}$
$f^j_{t_k} \in \mathbb{R}^{P \cdot d}$
$x^i \in \mathbb{R}^{r \cdot d}$
$\{\hat{f}^i_{t_k}\}^r_{i=1}$
$\{a^i_{t_{k-1}}\}^r_{i=1}$
$\{x^i_{t_0}\}^P_{i=1} \in \mathbb{R}^{P \cdot d}$
$\{x^i_{t_n}\}^P_{i=1} \in \mathbb{R}^{P \cdot d}$
g
κ_θ
t_0 t_1 t_k t_l t_m t_n
trajectory
$\mathbb{R}^{r \cdot d}$ Discretization
$\mathbb{R}^{r \cdot d}$ Discretization t_n
Particle Trajectory
Particle Trajectory
$\mathbb{R}^{P \cdot d}$ Discretization
$\mathbb{R}^{P \cdot d}$ Discretization t_n

Figure 1: **Overview of GIOROM.** This figure illustrates the temporal evolution of Lagrangian interaction systems using reduced-order modeling. (a) Prior discretization invariant methods evolve a global latent state $\hat{\mathbf{x}}$ using PDE time-steppers and reconstruct the full-field solution using neural decoders g. (b) In contrast, our approach employs a data-driven neural stepper to compute sparse field estimates $\{\hat{\boldsymbol{f}}^i\}_{i=1}^r \in \mathbb{R}^{r \cdot d}$, which are then used by the kernel κ to parameterize the solution manifold. $\boldsymbol{f}$ represents the deformation field depicted above.

Our work can be seen as discretization invariant kernel-ROM parameterization for reduced space that is both discretization invariant and non-intrusive.

Neural physics solvers Neural solvers have advanced simulations across fluid dynamics Sanchez-Gonzalez et al. (2020); Kochkov et al. (2021); Vinuesa & Brunton (2021); Mao et al. (2020); Shukla et al. (2024); Hao et al. (2024), solid mechanics Geist & Trimpe (2021); Capuano & Rimoli (2019); Jin et al. (2023), climate modeling Pathak et al. (2022), and robotics Ni & Qureshi (2022); Kaczmarski et al. (2023). These models range from data-driven to physics-informed architectures Raissi et al. (2019); Sirignano & Spiliopoulos (2018); Richter & Berner (2022); Nam et al. (2024). CNNs are effective on regular grids Lee & Carlberg (2020); Maulik et al. (2021); Stoffel et al. (2020); Bamer et al. (2021), while GNNs generalize to irregular meshes, enabling applications in mesh-based dynamics Pfaff et al. (2020); Cao et al. (2022); Han et al. (2022); Fortunato et al. (2022), Lagrangian systems Sanchez-Gonzalez et al. (2020), parametric PDEs Pichi et al. (2024), and rigid body physics Kneifl et al. (2024). Lagrangian modeling, which depends on particle-interactions are typically modeled with GNN based message passing. However, GNNs such as GNS and Meshgraphnets scale poorly with graph size due to node-wise message passing defined over all degrees of freedom. While traditional ROMs parameterize latent-spaces, our proposed reduction strategy can leverage these models directly, while achieving order reduction.

Neural operators Neural operators are a class of discretization-invariant models for learning mappings between infinite-dimensional function spaces. They have been applied to parametric PDEs Lu et al. (2021); Li et al. (2020c; 2023); Azizzadenesheli et al. (2024); Rahman et al. (2024); Liu-Schiaffini et al. (2024); Kovachki et al. (2023); Rahman et al. (2022a); Liu et al. (2022); Viswanath et al. (2023); Shih et al. (2024); Goswami et al. (2023), fluid simulations Di Leoni et al. (2023); Wang et al. (2024); Peyvan et al. (2024), 3D physics Xu et al. (2024); White et al. (2023); Bonev et al. (2023); Rahman et al. (2022b); He et al. (2024), and even cross-domain tasks in biology, robotics, and vision Liu et al. (2024a); Dharuman et al. (2023); Bhaskara et al. (2023); Peng et al. (2023); Guibas et al. (2021); Rahman & Yeh (2024); Viswanath et al. (2022). While operators such as FNOs, and Transolver Wu et al. (2024a), use projections within their frameworks, they learn a direct end-to-end function mapping rather than evolving a reduced state. Their architectures couple the projection and processing steps, meaning the online evaluation phase still depends on the full-order degrees of freedom of the input. However, DeepONet Lu et al. (2021) can be viewed as a form of Eulerian ROM solver. On the other hand, our work, bridges the gap between neural Reduced Order Models (ROMs), neural Lagrangian solvers and neural operators. We propose a principled framework for integrating operator learning within order reduction paradigm.

Our proposed reduction strategy can be used in Geometric Informed Neural Operator (GINO) Li et al. (2024) and the Unified Physics Transformer (UPT) Alkin et al. (2024) to reduce effective degrees of freedom, however, GINO is unsuitable for particle-interactions. The UPT family of architectures follow the encode-process-decode framework, with graph based layers facilitating Lagrangian inputs. But their architectures evaluate the solution field, rather than particle interactions and remain unsuitable for autoregressive particle dynamics.

Table 1: **Contrasting neural PDE operators and neural ROM solvers**: The upper half of the table represent features associated with neural physics solvers while the lower half represents features in neural ROM solvers

Category	FNO	GNS / MeshgraphNet	Transolver	GINO	UPT	DINo	CROM / LICROM	**Ours**
Discretization Invariance	✓[1]	✓	✓	✓	✓	✓	✓	✓
Irregular Domain Support	✗	✓	✓	✓	✓	✓	✓	✓
Neural PDE Operator	✓	✓	✓	✓	✓	✗	✗	✓
Locality Preserving Reduced Space	✗	✗	✗	✓	✗	✗	✗	✓
Particle Interaction Dynamics	✗	✓	✗[5]	✗	✗[6]	✗	✓	✓
Low-Dim State Representation	✓[2]	✗	✓	✓	✓	✓	✓	✓
Decoupled Latent Space Dynamics	✗	✗	✗	✗	✓	✓	✓	✓
Autoregressive Reduced Evolution	✗	✗	✗	✗	✗[6]	✓	✓	✓
Modularity	✗	✗	✗	✗	✓	✓	✓	✓
Effective Active DOFs (Online)	N	N	N	[4]$r \ll N$	[4]$r \ll N$	$r \ll N$	$r \ll N$	$r \ll N$
Non-Intrusive	✓	✓	✓	✓	✓	Partially[3]	✗	✓

Notes:

[1] **FNO:** While designed to be discretization-invariant, standard implementations perform best on regular grids and may require adaptation for irregular meshes.

[2] **FNO:** The "reduction" is in the learned operator, which is parameterized by a small number of Fourier modes, serving as the effective latent space, but not decoupled from the projection operator.

[3] **DINo:** Can be considered "partially intrusive" as they are designed around a neural ODE formulation governing the latent space. DINo considers Initial Value Problems (IVP) where the trajectories follow the same dynamics but have different initial conditions.

[4] **GINO/UPT:** Due to their ability to extrapolate inputs over arbitrary query points, their active degrees of freedom can be effectively reduced to a sparse set of points using our proposed reduction strategy.

[5] **Transolver:** The physics attention mechanism of Transolver does not support Lagrangian inputs, but can be combined with GNN based pre-processing layers to handle Lagrangian systems.

[6] **UPT:** The UPT architecture models field characteristics of Lagrangian systems instead of tracking individual particles and is therefore unable to model autoregressive evolution, wherein the encoder requires particle positions as input.

3 Method: Kernel parameterization

In this section, we discuss the mathematical formulation of the learnable kernel framework, which enables the evaluation of the function $\boldsymbol{f}$ at arbitrary query locations using a learned kernel function over a local neighborhood. The kernel serves as the backbone of our ROM framework, parameterizing the solution manifold from particles themselves. We provide the definitions for the notations in Appendix section C.

Let $\boldsymbol{f}$ be the underlying spatiotemporal solution function, assumed to belong to Hilbert space $\mathcal{H}$ of functions defined over $\Omega \in \mathbb{R}^d$ and $\mathcal{T} \in \mathbb{R}$. For a fixed $t \in \mathcal{T}$, we denote the spatial slice $\boldsymbol{f}_t : \Omega \rightarrow \mathbb{R}^d$. For brevity, we will omit the subscript t. The function $\boldsymbol{f}$ is assumed to lie on a continuous, smooth solution manifold $\mathcal{M}$. We do not make assumptions regarding access to its closed form and instead are provided with a set of pointwise approximations $\{x^i, \hat{\boldsymbol{f}}^i\}_{i=1}^r$, where r is the number of sparse samples.

We seek to construct an estimator $\kappa(x, \hat{\boldsymbol{f}}) \approx \boldsymbol{f}(x,t)$ for *any arbitrary* $x \in \Omega$, where the approximation is achieved by an implicit kernel parameterization operating between a point in $\mathbb{R}^d$ and the neighborhood in $\mathbb{R}^{r \cdot d}$, denoted as $\psi : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$. This forms a manifold parameterization for the solution field. Formally, this map $\kappa : \mathbb{R}^Q \to \mathbb{R}^{P \cdot d}$, is defined as $\kappa(x, \hat{\boldsymbol{f}}) := \sum_{y \in \mathcal{N}(x)} \psi_\theta(x,y)\hat{\boldsymbol{f}}(y,t)$, where ψ_θ is the kernel function parameterized by the neural network and the neighborhood $\mathcal{N}(x^i, \delta) = \{y \in \Omega : \|x^i - y\| \leq \delta\}$, where $\delta > 0$ is the neighborhood radius (training hyperparameter). Crucially, the estimator κ forms a continuous operator, with the domain and co-domain both being continuous and the estimator being discretization invariant. Unlike global latent representations in Chen et al. (2023); Chang et al. (2023); Yin et al. (2023) that embed the entire continuous field $\boldsymbol{f} \in \mathcal{M}$ into the latent vector, our kernel-based approach localizes evaluation around each query point. Specifically, the kernel restricts estimation to a *local region* within the manifold $\mathcal{M}$ near the query location x^i, allowing us to reconstruct $\boldsymbol{f}^i$ using the known neighboring samples obtained from the PDE solution operator ϕ_Θ.

In practice, we implement this as a discretization invariant kernel integral transform Li et al. (2024). Leveraging the PDE operator ϕ_Θ (discussed in section 4) to obtain the estimates $\hat{\boldsymbol{f}}$, the complete kernel network parameterization can be given by:

$$\boldsymbol{f}(x,t) \approx \sum_{y \in \mathcal{N}(x)} \psi_\theta(x,y)\phi_\Theta(y,t), \quad x \in \{x^i\}_{i=1}^P, y \in \{x^j\}_{j=1}^r \tag{3}$$

where ψ_θ is the neural kernel parameterization and κ is analogous to the neural field in Chen et al. (2023), $\phi_\Theta : a \to \hat{\boldsymbol{f}}$. The derivation is provided in section D.2.

We train the kernel network by minimizing the losses as follows

$$\mathcal{L}_\theta = \min_\theta \sum_{j=1}^P \sum_{t=1}^T \| \left(\sum_{\substack{y \in \mathcal{N}(x^j) \\ y \in \mathbb{R}^{r \cdot d}}} \psi_\theta(x^j, y)\phi_\Theta(y,t) \right) - \boldsymbol{f}_t^j \|_2^2 \tag{4}$$

This estimator, while effectively serving as a continuous neural implicit representation, differs from the neural fields proposed in Chen et al. (2023); Chang et al. (2023) in two ways - (1) We restrict the kernel to local neighborhoods of the evaluation point within the solution manifold to better capture fluid interfaces; (2) The framework is discretization invariant to the function in $(r.d)$-point discretization. The second follows from the discretization invariant properties of this kernel formulation, as discussed in Li et al. (2024). While the CROM and LiCROM frameworks rely on a discretization dependent projection operator to obtain the reduced latent state.

4 Method: Time-stepping on reduced sample set

We make a deviation in this section to focus on obtaining the sample set $\{x^i, \hat{\boldsymbol{f}}^i\}_{i=1}^r$, which is done using the operator ϕ_Θ. With $r \ll P$, we only require evaluating the *full-order dynamics* on a sparse set of points, achieving computational speedups. Following the UPT family of transformer architectures Alkin et al. (2024), we adapt the UPT backbone to enable autoregressive particle dynamics using the interaction network based encoder and decoder Battaglia et al. (2016). The interaction modules allow the operator framework to capture interaction dynamics and can be used in autoregressive generations akin to Sanchez-Gonzalez et al. (2020). However, the use of transformer processor enables speedups over GNS and MeshgraphNet style architectures. The modular nature of the architecture ensures the operator is agnostic to the choice of the transformer and is compatible with any PDE operator transformers such as Hao et al. (2023); Wu et al. (2024a); Lee & Oh (2024); Alkin et al. (2024).

4.1 Step 1: Input representation

We consider a reduced set of material points $\{x^i\}_{i=1}^r$, with $r \ll P$, for which we seek full-order evaluations without assuming a specific sampling scheme. Importantly, we do not train ϕ_Θ for a fixed r. The model ϕ_Θ is trained on coarse discretizations $\{x^i, a^i, \boldsymbol{f}^i\}_{i=1}^{|\mathbf{V}|}$, where $|\mathbf{V}| \ll P \cdot d$, defined on a radius graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$. Here, $\mathbf{V}$ denotes particles and $\mathbf{E}$ contains undirected edges between nodes within radius ρ_s, i.e., $(x, y) \in \mathbf{E} \iff d(x, y) \leq \rho_s$. The edges represent particle interactions.

For a subgraph $\mathbf{H} = (\upsilon, \varepsilon)$, defined over $\upsilon \subseteq \mathbf{V}$ with edges induced by radius ρ_r, we view $\mathbf{G}$ and $\mathbf{H}$ as discrete approximations of an underlying continuous manifold $\mathcal{C} \subset \mathbb{R}^d$ Jin et al. (2020); Burago et al. (2015). We observe that for reasonable sizes of $\mathbf{H}$, we can tune the radius ρ_r and enforce that for $x^i \in \mathbf{V} \cap \upsilon$, $\phi_{\Theta,\mathbf{H}}(x^i, t) \approx \phi_{\Theta,\mathbf{G}}(x^i, t) := \hat{\boldsymbol{f}}^i$, where $\phi_{\Theta,\mathbf{G}}$ represents the model operating on graph $\mathbf{G}$. This is achieved by enforcing a discrete neural operator behavior within the GNN architecture. We require tuning of ρ_r as fewer nodes alter local neighborhoods, which can disrupt message passing Garg et al. (2020); Gao & Isufi (2022).

4.2 Step 2: Heuristic for graph construction

For a subgraph $\mathbf{H}$ defined on $\upsilon \subseteq \mathbf{V}$, we require the connectivity radius $\rho_r \geq \rho_s$ to preserve paths between originally connected nodes, even after subsampling; r is chosen to optimize the kernel network performance/cost tradeoff, and consequently, ρ_r is tuned at inference to trade off accuracy and speed.

We use mean aggregation in message passing to ensure consistent node features across discretizations. This is shown to behave like a Monte Carlo approximation of kernel integrals, enabling the GNN to function as a neural operator under suitable graph construction Li et al. (2020b).

Prior sampling-based ROM approaches rely on stochastic, residual-guided strategies to select sample sets Chen et al. (2023). In contrast, we adopt simple sampling methods such as random or farthest-point selection. To ensure invariance to the sampling ratio for $r := |\upsilon|/|\mathbf{V}|$, we construct l random subgraphs per input domain to evaluate during training. This can be seen as a Nyström-type approximation, reducing variance and improving generalization across discretizations.

Empirically, we find that overly small $|\upsilon|$ degrades accuracy even with large ρ_r, implying a lower bound on r to retain spatial structure. This threshold guides inference-time subgraph construction (section L). Conversely, excessive ρ_r introduces redundant edges and deteriorates performance due to loss of locality.

4.3 Step 3: Spatio-temporal time-stepping

Equipped with our data structure $\mathbf{H}$, we now discuss the architecture of ϕ_Θ that enables the mapping $a_t^j \to \hat{\boldsymbol{f}}_{t+1}^j$. We define a^j and f^j to be point-wise function values defined on the node set $\upsilon \in \mathbb{R}^{r \cdot d}$.

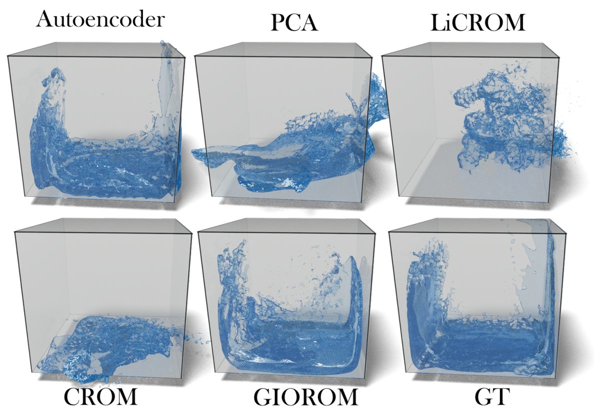
Extending the UPT family of architectures, our formulation can then be defined as

$$\phi_\Theta := IN^{dec,\mathbf{H}} \circ NOT^{process} \circ IN^{enc,\mathbf{H}} \tag{5}$$

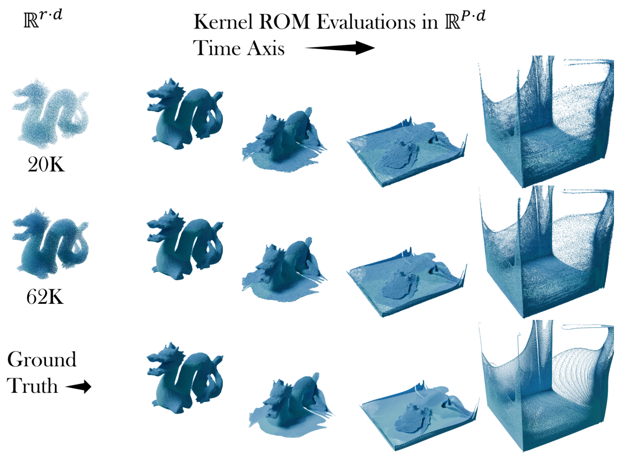
Where *enc* and *dec* denote the encoder and decoder, which operate on the graph $\mathbf{H}$. The NOT however operates only on the latent embeddings. The input for the model is the velocity sequence $\mathbf{v}$ defined over a time window w. Formally, $a_t^j = \mathbf{v}_{t-w:t}^j, \forall j \in \mathbb{R}^{r \cdot d}$. The output is then given as $\ddot{\hat{\boldsymbol{f}}}_t^j = \mathbf{a}_t^j$, where $\mathbf{a}$ is the pointwise acceleration defined on υ. We obtain $\hat{\boldsymbol{f}}_{t+1}^j$ by Euler integration. The model is trained following Sanchez-Gonzalez et al. (2020) with data-driven losses as follows

$$\mathcal{L}_\Theta = \min_\Theta \sum_j \|\phi_\Theta(a_t^j) - \ddot{\boldsymbol{f}}_t^j\|_2^2 \tag{6}$$

Additional training details and hyperparameter information is provided in section G and section I



(a)



(b)

Figure 2: **(a) Comparison of reduced-order modeling (ROM) techniques on fluid-in-container simulations**: The visualization reveals that the baseline models PCA (rollout MSE: 0.083), Autoencoder (0.091), LiCROM (0.033), and CROM (0.079), exhibit noticeable deviation from the true fluid boundaries, including overshooting beyond the container. GIOROM maintains physical fidelity with a lower rollout MSE of 0.0091. (b) **Demonstration of discretization convergence of kernel-ROM on high-resolution particle systems**: Contrasting the behavior of the kernel-ROM on a large-scale simulation containing approximately **2 million particles**, discretized as voxelized grids. Using randomly sampled $(r \cdot d)$-points of 62K and 20K particles (sampling percentages of 3% and 1%), the rollout MSEs are 0.0097 and 0.0088, respectively. These results indicate discretization convergence of κ under resolution refinement.

Table 2: This table showcases the performance of GIOROM on several physical systems. These results are computed on $\mathbb{R}^{P \cdot d}$.

PHYSICAL SYSTEM	**DURATION** ($5e^{-3}$**s**)	**EVAL PTS FOR** κ **IN** $\mathbb{R}^{P \cdot d}$	**INPUT GRAPH SIZE** $\mathbb{R}^{r \cdot d}$	**SCALE**	**ONE STEP-MSE** ($\times e^{-9}$)	**ROLLOUT MSE** ($\times e^{-3}$)
WATER-3D	1000	55k	1.7k	32×	5.23	0.386
WATER-2D	1000	1k	0.12k	8.3×	0.524	6.7
SAND-3D	400	32k	1k	32×	4.87	0.0025
SAND-2D	320	2k	0.3k	6.6×	8.5	1.34
GOOP-2D	400	1.9k	0.2k	9.5×	1.31	0.94
PLASTICINE	320	5k	1.1k	4.5×	0.974	0.5
ELASTICITY	120	78k	2.6k	30×	0.507	0.2
MULTI-MATERIAL 2D	1000	2k	0.25k	8×	2.3	9.43

5 Experiments

5.1 Dataset

Our dataset consists of four 3D physical systems - Newtonian fluids (Water), Drucker-Prager elastoplasticity (Sand), von Mises yield (Plasticine) and purely Elastic deformations. Unless otherwise noted, we assume the discretization to be point clouds.

We used the nclaw simulator (Ma et al., 2023) to generate 100 trajectories for each of these systems with random initial velocity conditions and a fixed boundary $[0, 1], [0, 1], [0, 1]$, with a free-slip boundary condition. The Δt between consecutive time frames is $5e^{-3}$s. We additionally used datasets provided by Sanchez-Gonzalez et al. (2020) - WaterDrop, Water-3D, Sand-3D, Sand, Goop and MultiMaterial. Additional training details and model hyperparameters can be found in Appendix G.2.

5.2 Results

Performance on different physical systems We evaluate GIOROM on a held-out validation set of unseen trajectories from multiple physical systems. Table 2 reports mean-squared error (MSE) for both single-step predictions and long-horizon rollouts, where the sample evaluations $\{x^i, \hat{\boldsymbol{f}}^i\}_{i=1}^r$ produced by ϕ_Θ are used by the kernel network to compute high-resolution evaluations in $\mathbb{R}^{P \cdot d}$. These are compared against ground-truth trajectories at the same resolution. Qualitative results in Figure 4 illustrate rollout fidelity, with Figure 7 demonstrating stability until equilibrium. Additional results and discussions are provided in section J and section L.

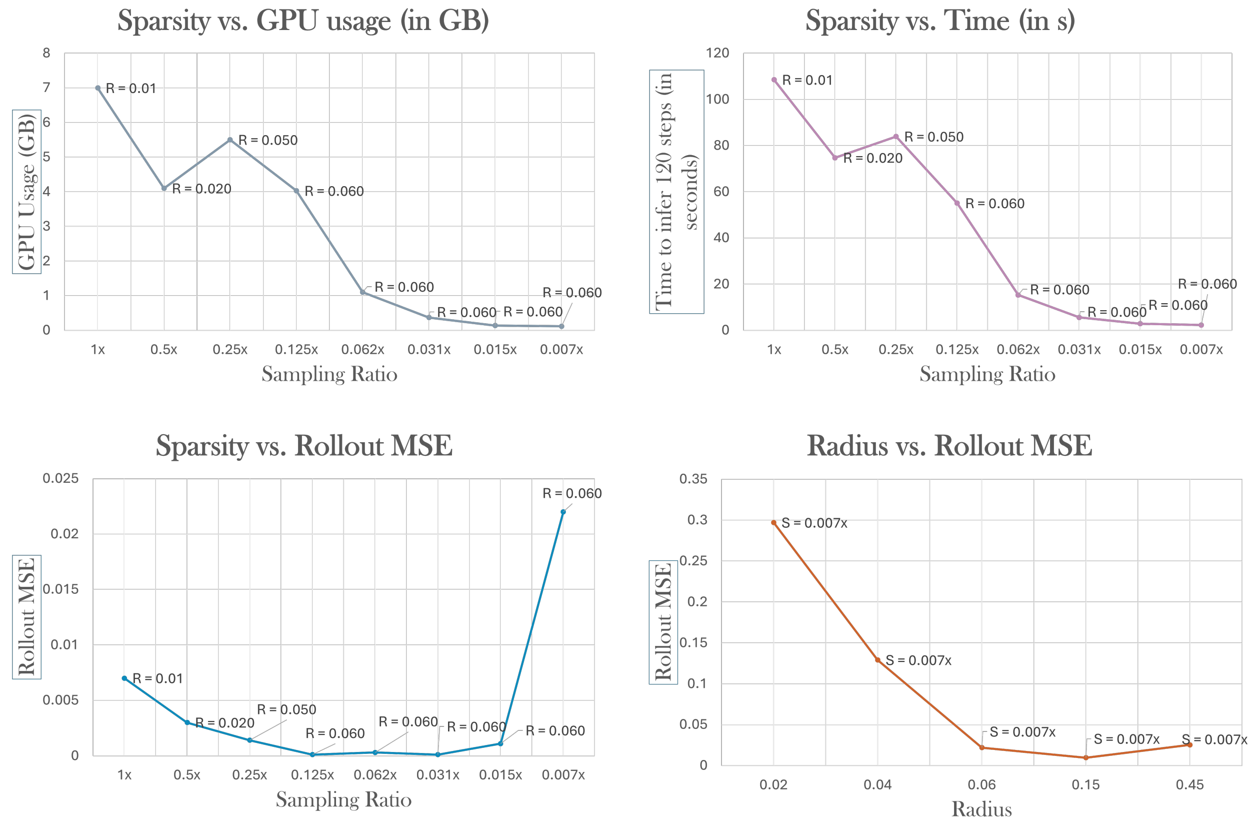


Figure 3: **Effect of sparsity on rollout error (Elasticity dataset, 78K particles).** Plots show performance of ϕ_Θ across varying sparsity levels, expressed as a fraction of the full system. GPU usage and computation time include pre-processing overhead for graph construction at each rollout step. Top-left plot reports peak GPU usage across graph sizes (R=radius, S=sample ratio); top-right shows net computation time for a rollout as a function of sparsity at fixed radius. Bottom-left plot reports rollout MSE, showing that performance remains stable ($\sim$1e-4) for sparsity levels 0.125×, 0.062×, and 0.031×. Below this, performance degrades due to oversparsification. Bottom-right plot shows that for high sparsification (0.007×), increasing the graph radius beyond 0.15 does not improve performance.

3D simulations Visualizations of 3D dynamics are shown in fig. 4, illustrating the output of GIOROM across multiple material models. Figure 5 provides side-by-side comparisons of predicted versus ground-truth simulations for water and sand. Figure 6 further contrasts predictions in the sampled subspace against reconstructions over the full discretized domain.

Robustness to sparse sample size r We evaluate the robustness of ϕ_Θ to varying sampling resolutions r on the elasticity dataset (full-order: 78K particles), as shown in Table 3. The first row reports performance on validation graphs closely matching training resolution ($\sim 1.2\times$ particle count). We assess generalization at

Table 3: This table highlights resolution and discretization generalization of ϕ_Θ in different settings of the Elasticity dataset.

SETTING	AVERAGE NUM. POINTS	SCALE W.R.T TRAINING DATA	ONE-STEP MSE (x e^{-9})	ROLLOUT MSE (x e^{-3})
SIMILAR DISC.	2.5k	1.25	0.8	0.2
LOWER RES.	1k	0.5	1.9	0.5
LOWER RES.	0.5k	0.25	2.34	0.6
HIGHER RES.	5k	2	0.319	0.7
HIGHER RES.	10k	4	0.88	0.9
INFERENCE ON $\mathbb{R}^{P\cdot d}$	78k	52	94.4	2

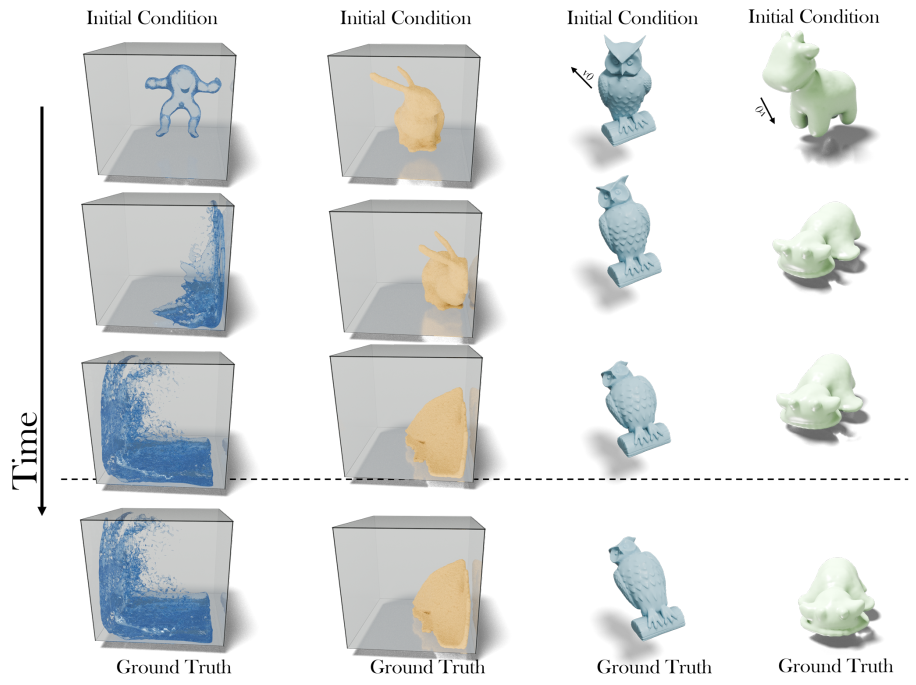


Figure 4: **3D physical simulations:** Time-evolved predictions for Newtonian fluid (water), Drucker–Prager (sand), von Mises (plasticine), and elastoplastic material models.

coarser (0.25×, 0.5×) and finer (2×, 4×) resolutions, with trends shown in fig. 3. Inference at $(P \cdot d)$-point discretization shows minor degradation, underscoring the scalability limitations of GNNs and the need for reduced-order models.

5.3 Ablations

Comparison with ROM architectures We evaluate our proposed kernel-ROM against reduced-order models. **Importantly, we restrict to modular projection based ROM models that support particle dynamics**. While Transolver, FNO and GINO include projection layers, they are trained end-to-end, and do not support Lagrangian inputs without attaching message passing layers. We evaluate baseline ROM models on highly dynamic water simulations with neighborhood changes, a scenario that is difficult for projection-based ROM techniques to model.We discuss the results in fig. 2a. The baselines we compared include (1) PCA (Principal Component Analysis), also known as POD (Proper Orthogonal Decomposition), (2) Autoencoders Lee & Carlberg (2020), (3) LICROM Chang et al. (2023), and (4) CROM Chen et al. (2023).

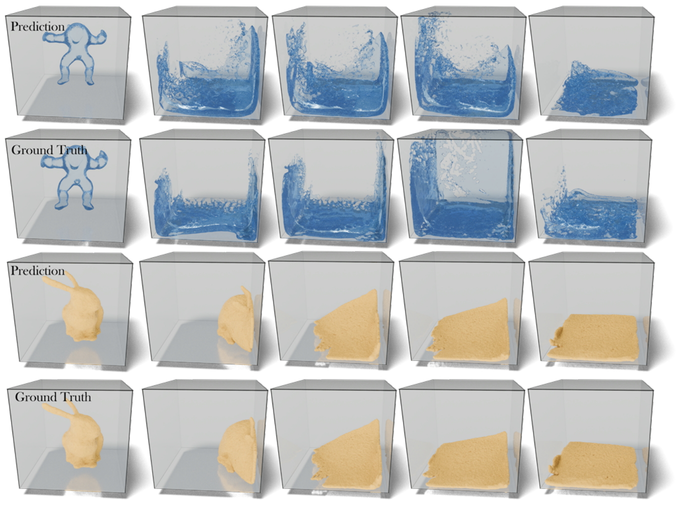


Figure 5: **Ground-truth comparisons:** Rendered outputs for water and sand simulations contrasted against reference solutions.

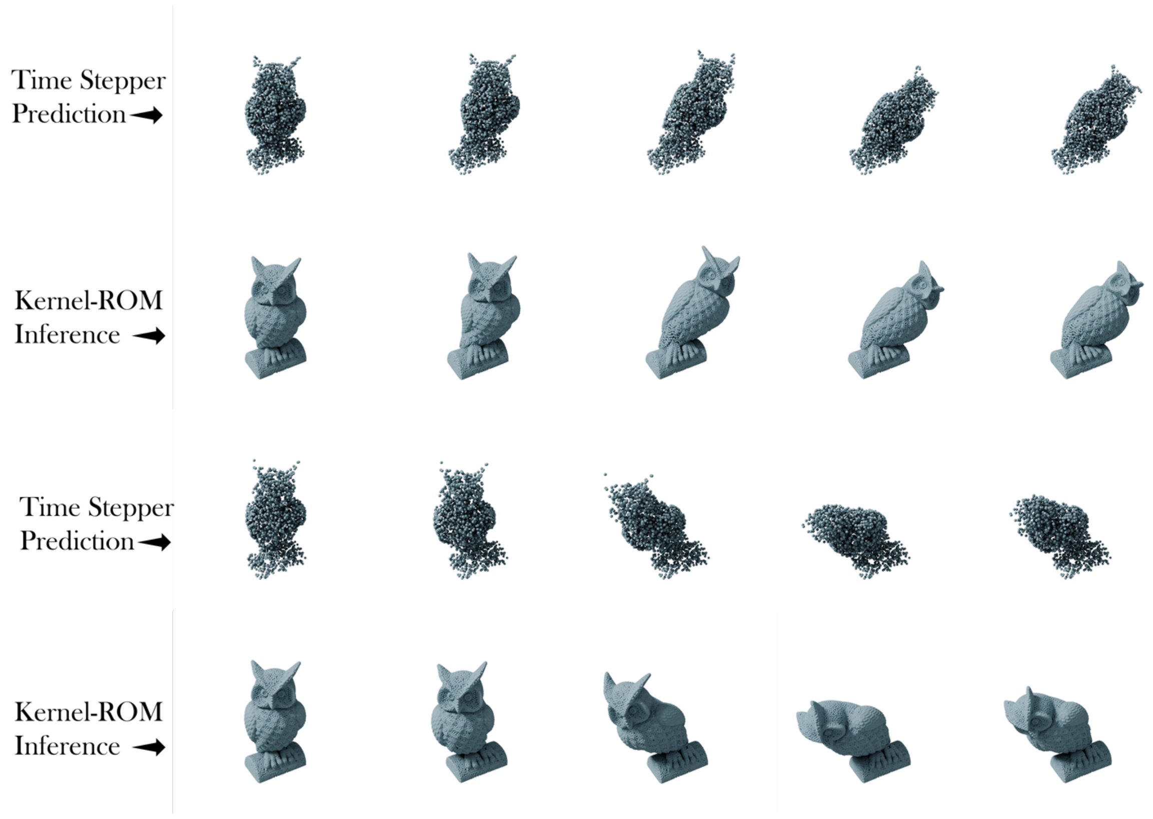


Figure 6: **Reduced space vs full-space inference:** Rows 1 and 3 show predictions from the PDE operator (autoregressive time-stepper) on the sampled space; Rows 2 and 4 depict Kernel output evaluated on 30× discretization.

While DINo is a ROM forecasting model that is PDE agnostic, it does not support interaction dynamics. Additionally we consider systems with varying time dynamics as shown in fig. 6.

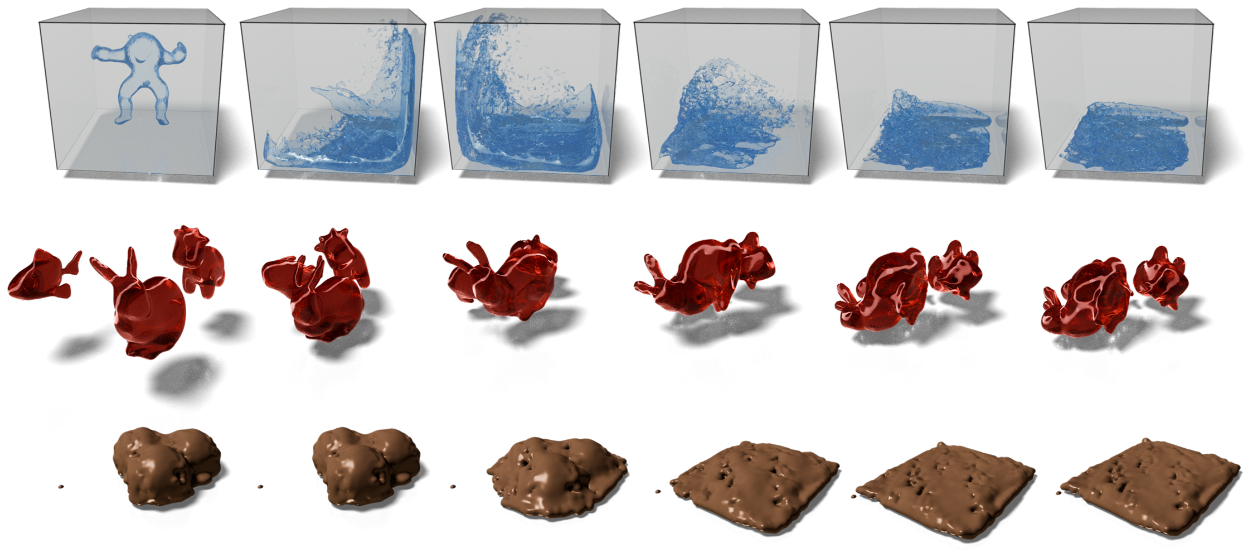


Figure 7: Rollout trajectories for three physical systems evaluated until equilibrium. (**Top**) Water simulation with high dynamism, rolled out for 1000 time steps over 55K particles; rollout MSE: 1.1×10^{-2}. (**Middle**) Elastic collisions (Jelly) over 500 time steps with 84K particles; rollout MSE: 6.4×10^{-5}. (**Bottom**) Phase transition in plasticine-like material (Chocolate) over 2000 time steps with 24K particles; rollout MSE: 3.2×10^{-4}.

Ablations on PDE operator Following Wu et al. (2024a); Alkin et al. (2024), we utilize interaction networks to model Lagrangian inputs. To justify this choice, we compare against GNS and neural operator models built for Eulerian regimes. Table 4 represents the rollout performance of different Neural Operator models on reduced sample sets. The performance is measured as the average MSE accumulated over the entire duration. We compare against **GINO (Eulerian)** Li et al. (2024), General Neural Operator Transformer **GNOT (Eulerian)** Hao et al. (2023) and Inducing Point Operator Transformer **IPOT (Eulerian)** Lee & Oh (2024). Additionally, we compare against graph neural network based model **GNS** Sanchez-Gonzalez et al. (2020). We do not consider the Lagrangian field variant of the UPT family as a baseline as it neither encodes particle interaction dynamics nor supports autoregression, making it incompatible with the datasets used in this work. All the operators struggle to generalize to fluid simulations but interaction network based models (GNS and GIOROM) exhibit the best performance.

Table 4: This table compares the rollout MSE of our PDE operator ϕ_Θ against other neural PDE operators on reduced sampled sets $\mathbb{R}^{r \cdot d}$. This highlights the importance of interaction network encoders and decoders for modeling particle dynamics.

MODEL	WATER-3D	PLASTICINE	ELASTIC	SAND-3D
GNS	0.0108	0.0038	0.0003	0.0008
GINO	0.38	0.09	0.18	0.07
GNOT	0.046	0.0052	0.0028	0.0085
IPOT	0.15	0.097	0.084	0.0075
ϕ_Θ	0.0106	0.0008	0.0004	0.0009

Speedup against graph neural networks Graph neural networks can effectively capture spatial interactions in point clouds. However, the message passing operation adds a computational overhead, which is improved by UPT style encode-process-decode structure. We show, in Table 5 and Table 6, that this formulation has faster inference times compared to GNNs.

Table 5: Contrasting the change in computation time with the increase in connectivity radius for a graph with 7056 points. The times shown represent the overall time needed to infer all 200 time steps. We compare our time-stepper with other neural network based physics solvers.

MODEL	TIME STEPS	NUMBER OF SPATIAL POINTS	CONNECTIVITY RADIUS						
			0.040	0.050	0.060	0.070	0.080	0.090	0.100
OURS	200	7056 points	**20.1s**	**34.3s**	**47.6s**	**65.8s**	**89.7s**	**104.1s**	**109.3s**
GNS	200	7056 points	43.5s	73.5s	111.6s	162s	226.2s	305.9s	386.0s

Table 6: Contrasting the change in computation time with an increase in graph size at a fixed radius of 0.060. The times shown represent the overall time needed to infer 200 time steps. We compare our time-stepper against other neural network based physics solvers

MODEL	PARAMETERS	CONNECTIVITY	MATERIAL	TIME STEPS	GRAPH SIZE			
					1776 POINTS	4143 POINTS	5608 POINTS	7056 POINTS
OURS	4,312,247	0.060	Plasticine	200	**3.9s**	**14.5s**	**27.3s**	**47.6s**
GNS	1,592,987	0.060	Plasticine	200	7.8 s	38.3s	68.7s	111.6s

Table 7: Contrasting the inference times (in seconds) for highly dense point clouds up-sampled from highly sparse graphs (1776 points).

TIME STEPS	ROLLOUT SIZE	ROLLOUT TIME ϕ_Θ (s)	FULL-ORDER SIZE	UPSCALE TIME κ (s)
200	1776	3.9	7,000	5e-3
200	1776	3.9	40,000	3e-3
200	1776	3.9	60,000	8e-3
200	1776	3.9	100,000	9e-3

6 Discussions and Conclusion

Computational Efficiency We evaluated the relationship between graph sparsity and both computational speed and memory consumption. As shown in fig. 3, increasing sparsity yields significant performance gains without compromising accuracy, up to a reduction factor of 0.031×, even for systems with approximately 78k particles.

Robustness to Sparsity Variations We tested the model under varying sparsity levels with appropriate graph structure. The model remains stable under super-sampling; however, performance degrades when sparsity exceeds the 0.031× threshold. These results, visualized in fig. 3, indicate a practical lower bound on sample density. We provide more discussions and a detailed analysis of limitations in section L.

Conclusion We show that our reduced-order modeling framework for learning Lagrangian dynamics on sparse inputs achieves computational improvements over existing neural solvers (table 6) while preserving high simulation fidelity across diverse physical systems, outperforming traditional ROM approaches, particularly, on dynamic fluid simulations. The end-to-end data-driven approach can open doors to incorporating model-order reduction to simulations generated by generative models, with no PDE priors.

Nonetheless, several limitations remain. GIOROM, like other learning-based ROMs, exhibits reduced performance under extreme out-of-distribution conditions (Li et al., 2020b; Chen et al., 2023) and extreme sparsifications (section L.2). Furthermore, while currently designed for continuous systems, future work may extend GIOROM to explicitly address discontinuities (Belhe et al., 2023; Goswami et al., 2022) and unbounded flows. While we currently only leverage Euler integration schemes, studying the impacts of higher-order methods, such as Runge-Kutta 4th order on accuracy or stability of ROM is an exciting future direction. We also restrict our problem setups to consistent time-discretizations. Understanding the behavior of neural time-steppers on varying time discretization is another consideration for future work.

References

Benedikt Alkin, Andreas Fürst, Simon Schmid, Lukas Gruber, Markus Holzleitner, and Johannes Brandstetter. Universal physics transformers. *CoRR*, 2024.

Steven S An, Theodore Kim, and Doug L James. Optimizing cubature for efficient integration of subspace deformations. *ACM transactions on graphics (TOG)*, 27(5):1–10, 2008.

Uri M Ascher and Linda R Petzold. *Computer methods for ordinary differential equations and differential-algebraic equations*. SIAM, 1998.

Kamyar Azizzadenesheli, Nikola Kovachki, Zongyi Li, Miguel Liu-Schiaffini, Jean Kossaifi, and Anima Anandkumar. Neural operators for accelerating scientific simulations and design. *Nature Reviews Physics*, pp. 1–9, 2024.

Franz Bamer, Denny Thaler, Marcus Stoffel, and Bernd Markert. A monte carlo simulation approach in non-linear structural dynamics using convolutional neural networks. *Frontiers in Built Environment*, 7, 2021. ISSN 2297-3362. doi: 10.3389/fbuil.2021.679488. URL `https://www.frontiersin.org/articles/10.3389/fbuil.2021.679488`.

Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. *Advances in neural information processing systems*, 29, 2016.

Yash Belhe, Michaël Gharbi, Matthew Fisher, Iliyan Georgiev, Ravi Ramamoorthi, and Tzu-Mao Li. Discontinuity-aware 2d neural fields. *ACM Transactions on Graphics (TOG)*, 42(6):1–11, 2023.

Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, 15(6):1373–1396, 2003.

Gal Berkooz, Philip Holmes, and John L Lumley. The proper orthogonal decomposition in the analysis of turbulent flows. *Annual review of fluid mechanics*, 25(1):539–575, 1993.

Alain Berlinet and Christine Thomas-Agnan. *Reproducing kernel Hilbert spaces in probability and statistics*. Springer Science & Business Media, 2011.

Tyrus Berry and Timothy Sauer. Local kernels and the geometric structure of data. *Applied and Computational Harmonic Analysis*, 40(3):439–469, 2016.

Lander Besabe, Michele Girfoglio, Annalisa Quaini, and Gianluigi Rozza. Data-driven reduced order modeling of a two-layer quasi-geostrophic ocean model. *Results in Engineering*, 25:103691, 2025.

Rashmi Bhaskara, Hrishikesh Viswanath, and Aniket Bera. Trajectory prediction for robot navigation using flow-guided markov neural operator. *arXiv preprint arXiv:2309.09137*, 2023.

Boris Bonev, Thorsten Kurth, Christian Hundt, Jaideep Pathak, Maximilian Baust, Karthik Kashinath, and Anima Anandkumar. Spherical fourier neural operators: Learning stable dynamics on the sphere. In *International conference on machine learning*, pp. 2806–2823. PMLR, 2023.

Dmitri Burago, Sergei Ivanov, and Yaroslav Kurylev. A graph discretization of the laplace–beltrami operator. *Journal of Spectral Theory*, 4(4):675–714, 2015.

Yadi Cao, Menglei Chai, Minchen Li, and Chenfanfu Jiang. Efficient learning of mesh-based physical simulation with bsms-gnn. *arXiv preprint arXiv:2210.02573*, 2022.

German Capuano and Julian J Rimoli. Smart finite elements: A novel machine learning application. *Computer Methods in Applied Mechanics and Engineering*, 345:363–381, 2019.

Yue Chang, Peter Yichen Chen, Zhecheng Wang, Maurizio M. Chiaramonte, Kevin Carlberg, and Eitan Grinspun. Licrom: Linear-subspace continuous reduced order modeling with neural fields. In *SIGGRAPH Asia 2023 Conference Papers*, SA '23, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400703157. doi: 10.1145/3610548.3618158. URL `https://doi.org/10.1145/3610548.3618158`.

Peter Yichen Chen, Maurizio Chiaramonte, Eitan Grinspun, and Kevin Carlberg. Model reduction for the material point method via learning the deformation map and its spatial-temporal gradients. *URL: http://arxiv. org/abs/2109.12390 v1*, 2021.

Peter Yichen Chen, Jinxu Xiang, Dong Heon Cho, Yue Chang, G A Pershing, Henrique Teles Maia, Maurizio M Chiaramonte, Kevin Thomas Carlberg, and Eitan Grinspun. CROM: Continuous reduced-order modeling of PDEs using implicit neural representations. In *The Eleventh International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=FUORz1tG8Og`.

Ronald R Coifman and Stéphane Lafon. Diffusion maps. *Applied and computational harmonic analysis*, 21 (1):5–30, 2006.

Gautham Dharuman, Logan Ward, Heng Ma, Priyanka V Setty, Ozan Gokdemir, Sam Foreman, Murali Emani, Kyle Hippe, Alexander Brace, Kristopher Keipert, et al. Protein generation via genome-scale language models with bio-physical scoring. In *Proceedings of the SC'23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, pp. 95–101, 2023.

Patricio Clark Di Leoni, Lu Lu, Charles Meneveau, George Em Karniadakis, and Tamer A Zaki. Neural operator prediction of linear instability waves in high-speed boundary layers. *Journal of Computational Physics*, 474:111793, 2023.

Meire Fortunato, Tobias Pfaff, Peter Wirnsberger, Alexander Pritzel, and Peter Battaglia. Multiscale meshgraphnets. *arXiv preprint arXiv:2210.00612*, 2022.

Zhan Gao and Elvin Isufi. Learning stable graph neural networks via spectral regularization. In *2022 56th Asilomar Conference on Signals, Systems, and Computers*, pp. 01–05. IEEE, 2022.

Vikas Garg, Stefanie Jegelka, and Tommi Jaakkola. Generalization and representational limits of graph neural networks. In *International conference on machine learning*, pp. 3419–3430. PMLR, 2020.

A René Geist and Sebastian Trimpe. Structured learning of rigid-body dynamics: A survey and unified view from a robotics perspective. *GAMM-Mitteilungen*, 44(2):e202100009, 2021.

Somdatta Goswami, Minglang Yin, Yue Yu, and George Em Karniadakis. A physics-informed variational deeponet for predicting crack path in quasi-brittle materials. *Computer Methods in Applied Mechanics and Engineering*, 391:114587, 2022.

Somdatta Goswami, Aniruddha Bora, Yue Yu, and George Em Karniadakis. Physics-informed deep neural operator networks. In *Machine Learning in Modeling and Simulation: Methods and Applications*, pp. 219–254. Springer, 2023.

John Guibas, Morteza Mardani, Zongyi Li, Andrew Tao, Anima Anandkumar, and Bryan Catanzaro. Adaptive fourier neural operators: Efficient token mixers for transformers. *arXiv preprint arXiv:2111.13587*, 2021.

Mengwu Guo and Jan S Hesthaven. Data-driven reduced order modeling for time-dependent problems. *Computer methods in applied mechanics and engineering*, 345:75–99, 2019.

Xu Han, Han Gao, Tobias Pfaff, Jian-Xun Wang, and Li-Ping Liu. Predicting physics in mesh-reduced space with temporal attention. *arXiv preprint arXiv:2201.09113*, 2022.

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. In *International Conference on Machine Learning*, pp. 12556–12569. PMLR, 2023.

Zhongkai Hao, Chang Su, Songming Liu, Julius Berner, Chengyang Ying, Hang Su, Anima Anandkumar, Jian Song, and Jun Zhu. DPOT: Auto-regressive denoising operator transformer for large-scale pde pre-training. *arXiv preprint arXiv:2403.03542*, 2024.

Junyan He, Seid Koric, Diab Abueidda, Ali Najafi, and Iwona Jasiuk. Geom-deeponet: A point-cloud-based deep operator network for field predictions on 3d parameterized geometries. *arXiv preprint arXiv:2403.14788*, 2024.

Philip Holmes, John L Lumley, Gahl Berkooz, and Clarence W Rowley. *Turbulence, coherent structures, dynamical systems and symmetry*. Cambridge university press, 2012.

Paul Honeine. Online kernel principal component analysis: A reduced-order model. *IEEE transactions on pattern analysis and machine intelligence*, 34(9):1814–1826, 2011.

Thomas JR Hughes. *The finite element method: linear static and dynamic finite element analysis*. Courier Corporation, 2012.

Alan Julian Izenman. Introduction to manifold learning. *Wiley Interdisciplinary Reviews: Computational Statistics*, 4(5):439–446, 2012.

Steeven Janny, Madiha Nadri, Julie Digne, and Christian Wolf. Space and time continuous physics simulation from partial observations. In *International Conference on Learning Representation*, Vienna, Austria, May 2024. URL `https://hal.science/hal-04464153`.

Joongoo Jeon, Juhyeong Lee, Ricardo Vinuesa, and Sung Joong Kim. Residual-based physics-informed transfer learning: A hybrid method for accelerating long-term cfd simulations via deep learning. *International Journal of Heat and Mass Transfer*, 220:124900, 2024.

Chenfanfu Jiang, Craig Schroeder, Joseph Teran, Alexey Stomakhin, and Andrew Selle. The material point method for simulating continuum materials. *ACM SIGGRAPH 2016 Courses*, pp. 1–52, 2016.

Ruoxi Jiang, Peter Y Lu, Elena Orlova, and Rebecca Willett. Training neural operators to preserve invariant measures of chaotic attractors. *Advances in Neural Information Processing Systems*, 36, 2024.

Hanxun Jin, Enrui Zhang, and Horacio D Espinosa. Recent advances and applications of machine learning in experimental solid mechanics: A review. *Applied Mechanics Reviews*, 75(6):061001, 2023.

Yu Jin, Andreas Loukas, and Joseph JaJa. Graph coarsening with preserved spectral properties. In *International Conference on Artificial Intelligence and Statistics*, pp. 4452–4462. PMLR, 2020.

Bartosz Kaczmarski, Alain Goriely, Ellen Kuhl, and Derek E Moulton. A simulation tool for physics-informed control of biomimetic soft robotic arms. *IEEE Robotics and Automation Letters*, 8(2):936–943, 2023.

Jonas Kneifl, Jörg Fehr, Steven L Brunton, and J Nathan Kutz. Multi-hierarchical surrogate learning for structural dynamics of automotive crashworthiness using graph convolutional neural networks. *arXiv preprint arXiv:2402.09234*, 2024.

Dmitrii Kochkov, Jamie A Smith, Ayya Alieva, Qing Wang, Michael P Brenner, and Stephan Hoyer. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118 (21):e2101784118, 2021.

Jiaqing Kou and Weiwei Zhang. Multi-kernel neural networks for nonlinear unsteady aerodynamic reduced-order modeling. *Aerospace Science and Technology*, 67:309–326, 2017.

Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces with applications to pdes. *Journal of Machine Learning Research*, 24(89):1–97, 2023.

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. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2019.108973. URL `https://www.sciencedirect.com/science/article/pii/S0021999119306783`.

Seungjun Lee and Taeil Oh. Inducing point operator transformer: A flexible and scalable architecture for solving pdes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 153–161, 2024.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020a.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Graph kernel network for partial differential equations. *arXiv preprint arXiv:2003.03485*, 2020b.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Andrew Stuart, Kaushik Bhattacharya, and Anima Anandkumar. Multipole graph neural operator for parametric partial differential equations. *Advances in Neural Information Processing Systems*, 33:6755–6766, 2020c.

Zongyi Li, Daniel Zhengyu Huang, Burigede Liu, and Anima Anandkumar. Fourier neural operator with learned deformations for pdes on general geometries. *Journal of Machine Learning Research*, 24(388):1–26, 2023.

Zongyi Li, Nikola Kovachki, Chris Choy, Boyi Li, Jean Kossaifi, Shourya Otta, Mohammad Amin Nabian, Maximilian Stadler, Christian Hundt, Kamyar Azizzadenesheli, et al. Geometry-informed neural operator for large-scale 3d pdes. *Advances in Neural Information Processing Systems*, 36, 2024.

Bjoern List, Li-Wei Chen, Kartik Bali, and Nils Thuerey. How temporal unrolling supports neural physics simulators. *arXiv preprint arXiv:2402.12971*, 2024.

Burigede Liu, Nikola Kovachki, Zongyi Li, Kamyar Azizzadenesheli, Anima Anandkumar, Andrew M Stuart, and Kaushik Bhattacharya. A learning-based multiscale method and its application to inelastic impact problems. *Journal of the Mechanics and Physics of Solids*, 158:104668, 2022.

Shengchao Liu, Yanjing Li, Zhuoxinran Li, Zhiling Zheng, Chenru Duan, Zhi-Ming Ma, Omar Yaghi, Animashree Anandkumar, Christian Borgs, Jennifer Chayes, et al. Symmetry-informed geometric representation for molecules, proteins, and crystalline materials. *Advances in Neural Information Processing Systems*, 36, 2024a.

Yuying Liu, Aleksei Sholokhov, Hassan Mansour, and Saleh Nabi. Physics-informed koopman network for time-series prediction of dynamical systems. In *ICLR 2024 Workshop on AI4DifferentialEquations In Science*, 2024b.

Miguel Liu-Schiaffini, Julius Berner, Boris Bonev, Thorsten Kurth, Kamyar Azizzadenesheli, and Anima Anandkumar. Neural operators with localized integral and differential kernels. *arXiv preprint arXiv:2402.16845*, 2024.

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, 2021.

David J Lucia, Philip S Beran, and Walter A Silva. Reduced-order modeling: new approaches for computational physics. *Progress in aerospace sciences*, 40(1-2):51–117, 2004.

Pingchuan Ma, Peter Yichen Chen, Bolei Deng, Joshua B Tenenbaum, Tao Du, Chuang Gan, and Wojciech Matusik. Learning neural constitutive laws from motion observations for generalizable pde dynamics. In *International Conference on Machine Learning*, pp. 23279–23300. PMLR, 2023.

Qilong Ma, Haixu Wu, Lanxiang Xing, Jianmin Wang, and Mingsheng Long. Eulagnet: Eulerian fluid prediction with lagrangian dynamics. *arXiv preprint arXiv:2402.02425*, 2024.

Harris Abdul Majid and Francesco Tudisco. Mixture of neural operators: Incorporating historical information for longer rollouts. In *ICLR 2024 Workshop on AI4DifferentialEquations In Science*, 2024.

Zhiping Mao, Ameya D Jagtap, and George Em Karniadakis. Physics-informed neural networks for high-speed flows. *Computer Methods in Applied Mechanics and Engineering*, 360:112789, 2020.

Romit Maulik, Bethany Lusch, and Prasanna Balaprakash. Reduced-order modeling of advection-dominated systems with recurrent neural networks and convolutional autoencoders. *Physics of Fluids*, 33(3):037106, 03 2021. ISSN 1070-6631. doi: 10.1063/5.0039986. URL `https://doi.org/10.1063/5.0039986`.

Joe J Monaghan. Smoothed particle hydrodynamics. *In: Annual review of astronomy and astrophysics. Vol. 30 (A93-25826 09-90), p. 543-574.*, 30:543–574, 1992.

Sorin Munteanu, John Rajadas, Changho Nam, and Aditi Chattopadhyay. A volterra kernel reduced-order model approach for nonlinear aeroelastic analysis. In *46th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference*, pp. 1854, 2005.

Hong Chul Nam, Julius Berner, and Anima Anandkumar. Solving Poisson equations using neural walk-on-spheres. *arXiv preprint arXiv:2406.03494*, 2024.

Ruiqi Ni and Ahmed H Qureshi. Ntfields: Neural time fields for physics-informed robot motion planning. *arXiv preprint arXiv:2210.00120*, 2022.

Shaowu Pan, Steven L Brunton, and J Nathan Kutz. Neural implicit flow: a mesh-agnostic dimensionality reduction paradigm of spatio-temporal data. *Journal of Machine Learning Research*, 24(41):1–60, 2023.

Jaideep Pathak, Shashank Subramanian, Peter Harrington, Sanjeev Raja, Ashesh Chattopadhyay, Morteza Mardani, Thorsten Kurth, David Hall, Zongyi Li, Kamyar Azizzadenesheli, et al. Fourcastnet: A global data-driven high-resolution weather model using adaptive fourier neural operators. *arXiv preprint arXiv:2202.11214*, 2022.

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

Benjamin Peherstorfer and Karen Willcox. Dynamic data-driven reduced-order models. *Computer Methods in Applied Mechanics and Engineering*, 291:21–41, 2015.

Juntong Peng, Hrishikesh Viswanath, Kshitij Tiwari, and Aniket Bera. Graph-based decentralized task allocation for multi-robot target localization. *arXiv preprint arXiv:2309.08896*, 2023.

Ahmad Peyvan, Vivek Oommen, Ameya D Jagtap, and George Em Karniadakis. Riemannonets: Interpretable neural operators for riemann problems. *arXiv preprint arXiv:2401.08886*, 2024.

Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W Battaglia. Learning mesh-based simulation with graph networks. *arXiv preprint arXiv:2010.03409*, 2020.

Federico Pichi, Beatriz Moya, and Jan S Hesthaven. A graph convolutional autoencoder approach to model order reduction for parametrized pdes. *Journal of Computational Physics*, 501:112762, 2024.

Md Ashiqur Rahman and Raymond A Yeh. Truly scale-equivariant deep nets with fourier layers. *Advances in Neural Information Processing Systems*, 36, 2024.

Md Ashiqur Rahman, Manuel A Florez, Anima Anandkumar, Zachary E Ross, and Kamyar Azizzadenesheli. Generative adversarial neural operators. *arXiv preprint arXiv:2205.03017*, 2022a.

Md Ashiqur Rahman, Jasorsi Ghosh, Hrishikesh Viswanath, Kamyar Azizzadenesheli, and Aniket Bera. Pacmo: Partner dependent human motion generation in dyadic human activity using neural operators. *arXiv preprint arXiv:2211.16210*, 2022b.

Md Ashiqur Rahman, Robert Joseph George, Mogab Elleithy, Daniel Leibovici, Zongyi Li, Boris Bonev, Colin White, Julius Berner, Raymond A Yeh, Jean Kossaifi, et al. Pretraining codomain attention neural operators for solving multiphysics pdes. *arXiv preprint arXiv:2403.12553*, 2024.

Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

Daniella E Raveh. Reduced-order models for nonlinear unsteady aerodynamics. *AIAA journal*, 39(8): 1417–1429, 2001.

Lorenz Richter and Julius Berner. Robust sde-based variational formulations for solving linear pdes via deep learning. In *International Conference on Machine Learning*, pp. 18649–18666. PMLR, 2022.

Matteo Salvador, Luca Dede, and Andrea Manzoni. Non intrusive reduced order modeling of parametrized pdes by kernel pod and neural networks. *Computers & Mathematics with Applications*, 104:1–13, 2021.

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

Rajat Sarkar, Krishna Sai Sudhir Aripirala, Vishal Sudam Jadhav, Sagar Srinivas Sakhinana, and Venkataramana Runkana. Pointsage: Mesh-independent superresolution approach to fluid flow predictions. In *ICLR 2024 Workshop on AI4DifferentialEquations In Science*, 2024.

Bernhard Schölkopf and Alexander J Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

Benjamin Shih, Ahmad Peyvan, Zhongqiang Zhang, and George Em Karniadakis. Transformers as neural operators for solutions of differential equations with finite regularity, 2024.

Khemraj Shukla, Vivek Oommen, Ahmad Peyvan, Michael Penwarden, Nicholas Plewacki, Luis Bravo, Anindya Ghoshal, Robert M Kirby, and George Em Karniadakis. Deep neural operators as accurate surrogates for shape optimization. *Engineering Applications of Artificial Intelligence*, 129:107615, 2024.

Justin Sirignano and Konstantinos Spiliopoulos. DGM: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375:1339–1364, 2018.

Breannan Smith, Fernando De Goes, and Theodore Kim. Stable neo-hookean flesh simulation. *ACM Transactions on Graphics (TOG)*, 37(2):1–15, 2018.

Marcus Stoffel, Franz Bamer, and Bernd Markert. Deep convolutional neural networks in structural dynamics under consideration of viscoplastic material behaviour. *Mechanics Research Communications*, 108:103565, 2020. ISSN 0093-6413. doi: https://doi.org/10.1016/j.mechrescom.2020.103565. URL `https://www.sciencedirect.com/science/article/pii/S0093641320300938`.

Benjamin Ummenhofer, Lukas Prantl, Nils Thuerey, and Vladlen Koltun. Lagrangian fluid simulation with continuous convolutions. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=B1lDoJSYDH`.

Ricardo Vinuesa and Steven L Brunton. The potential of machine learning to enhance computational fluid dynamics. *arXiv preprint arXiv:2110.02085*, pp. 1–13, 2021.

Hrishikesh Viswanath, Md Ashiqur Rahman, Rashmi Bhaskara, and Aniket Bera. Adafnio: Adaptive fourier neural interpolation operator for video frame interpolation. *arXiv preprint arXiv:2211.10791*, 2022.

Hrishikesh Viswanath, Md Ashiqur Rahman, Abhijeet Vyas, Andrey Shor, Beatriz Medeiros, Stephanie Hernandez, Suhas Eswarappa Prameela, and Aniket Bera. Neural operator: Is data all you need to model the world? an insight into the impact of physics informed machine learning. *arXiv preprint arXiv:2301.13331*, 2023.

Sifan Wang, Jacob H Seidman, Shyam Sankaran, Hanwen Wang, George J Pappas, and Paris Perdikaris. Bridging operator learning and conditioned neural fields: A unifying perspective. *arXiv preprint arXiv:2405.13998*, 2024.

Tianshu Wen, Kookjin Lee, and Youngsoo Choi. Reduced-order modeling for parameterized pdes via implicit neural representations. *arXiv preprint arXiv:2311.16410*, 2023.

Colin White, Julius Berner, Jean Kossaifi, Mogab Elleithy, David Pitt, Daniel Leibovici, Zongyi Li, Kamyar Azizzadenesheli, and Anima Anandkumar. Physics-informed neural operators with exact differentiation on arbitrary geometries. In *The Symbiosis of Deep Learning and Differential Equations III*, 2023.

Alexander Wikner, Joseph Harvey, Michelle Girvan, Brian R Hunt, Andrew Pomerance, Thomas Antonsen, and Edward Ott. Stabilizing machine learning prediction of dynamics: Novel noise-inspired regularization tested with reservoir computing. *Neural Networks*, 170:94–110, 2024.

Haixu Wu, Huakun Luo, Haowen Wang, Jianmin Wang, and Mingsheng Long. Transolver: A fast transformer solver for pdes on general geometries. *arXiv preprint arXiv:2402.02366*, 2024a.

Hao Wu, Fan Xu, Yifan Duan, Ziwei Niu, Weiyan Wang, Gaofeng Lu, Kun Wang, Yuxuan Liang, and Yang Wang. Spatio-temporal fluid dynamics modeling via physical-awareness and parameter diffusion guidance. *arXiv preprint arXiv:2403.13850*, 2024b.

Minkai Xu, Jiaqi Han, Aaron Lou, Jean Kossaifi, Arvind Ramanathan, Kamyar Azizzadenesheli, Jure Leskovec, Stefano Ermon, and Anima Anandkumar. Equivariant graph neural operator for modeling 3d dynamics. *arXiv preprint arXiv:2401.11037*, 2024.

Yuan Yin, Matthieu Kirchmeyer, Jean-Yves Franceschi, Alain Rakotomamonjy, and patrick gallinari. Continuous PDE dynamics forecasting with implicit neural representations. In *The Eleventh International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=B73niNjbPs`.

Miloš Zefran and Francesco Bullo. Lagrangian dynamics. *Robotics and Automation Handbook*, pp. 5–1, 2005.

Qingnan Zhou and Alec Jacobson. Thingi10k: A dataset of 10,000 3d-printing models, 2016.

A Appendix

B Additional related works

Time series dynamical systems Simulating temporal dynamics in an auto-regressive manner is a particularly challenging task due to error accumulations during long rollout Wikner et al. (2024); List et al. (2024). There have been many works that learn temporal PDEs and CFD, including Majid & Tudisco (2024); Liu et al. (2024b); Sarkar et al. (2024); Wu et al. (2024b); Jeon et al. (2024); Jiang et al. (2024); Ma et al. (2024); Janny et al. (2024). Some works have proposed neural network-based approaches to model 3D Lagrangian dynamics, such as Ummenhofer et al. (2020), who propose a convolutional neural network-based approach to model the behavior of Newtonian fluids in 3D systems. Sanchez-Gonzalez et al. (2020) propose a more general graph-based framework, but the network suffers from high computation time on very dense graphs.

C Definitions

Full-order model A full-order model evolves the solution over all degrees of freedom present in the spatial discretization scheme (Hughes, 2012) (e.g., $P \cdot d$ in $\mathbb{R}^{P \cdot d}$ or $r \cdot d$ in $\mathbb{R}^{r \cdot d}$). These models do not perform any dimensionality reduction. They are therefore prohibitively slow when P is large. We denote the P-point discretization of the spatial field $\boldsymbol{X}$ by $\{x^j\}_{j=1}^P$, where x^j represents a particle in $\mathbb{R}^d$ with negligible mass. Similarly, a sample set $\{x^i\}_{i=1}^r \subset \{x^j\}_{j=1}^P$ forms an $(r \cdot d)$ vector.

Reduced-order model A reduced-order model evolves the solution in reduced latent space $\mathbb{R}^Q$, $Q \ll P \cdot d$. Reduced-order techniques such as Chen et al. (2023) leverage neural fields and projection-based ROM to infer the continuous spatial function at arbitrary spatial locations from the low-dimensional latents. In a similar vein, for an evaluation point x^i and its local spatial neighborhood $\tilde{x} = \mathcal{N}(x^i)$, we define our parameterization $\kappa(x^i, \phi_\Theta) \approx \boldsymbol{f}(x^i, t)$ as $\kappa(x^i, \phi_{\Theta,\tilde{x}}) \approx \boldsymbol{f}(x^i, t), \quad \text{for } x^i \in \mathbb{R}^d,\ \tilde{x} = \mathcal{N}(x^i), \phi \in \mathbb{R}^{r \cdot d}$. We note here that κ parameterizes $\mathcal{M}(\boldsymbol{f})$ and $\phi_{\Theta,\tilde{x}} := \{\hat{\boldsymbol{f}}^j\}_{j \in \mathcal{N}(x^i)}$. We also note that $\hat{\boldsymbol{f}}$ has a temporal dependency, i.e. $\hat{\boldsymbol{f}}_t$ represents the evaluation at t. However, we omit the time subscript for brevity.

Traditional ROM approaches define the latents to be a nonlinear low-dimensional manifold Chen et al. (2021). Such a low-dimension parameterization can be defined with a projection mapping, $g : \mathbb{R}^Q \to \mathbb{R}^{P \cdot d}$, where $Q \ll P \cdot d$. This defines a map to a discrete field for every low dimensional latent vector $\hat{\mathbf{x}}(t) \in \mathbb{R}^Q$ such that $g_P(\hat{\mathbf{x}}) \mapsto (\boldsymbol{f}_t^1, ..., \boldsymbol{f}_t^P)$ Chen et al. (2023); Chang et al. (2023). For point-wise evaluations, $g(x^i, \hat{\mathbf{x}}_t) \approx \boldsymbol{f}(x^i, t) = \boldsymbol{f}_t^i$, with $\hat{\mathbf{x}}$ serving as a low-dimensional latent representation of the continuous field $\boldsymbol{f}$.

Discretization invariant ROM approaches Chen et al. (2023); Chang et al. (2023) construct the latent state $\hat{\mathbf{x}}_t$ from a known set of point-wise solution values $\phi = (\boldsymbol{f}_t^1, \ldots, \boldsymbol{f}_t^P)$ through a projection map $\pi : \mathbb{R}^{P \cdot d} \to \mathbb{R}^Q$, often parameterized using PointNet-style models which enable continuous evaluation. However, temporal evolution, $\hat{\mathbf{x}}_t \mapsto \hat{\mathbf{x}}_{t+1}$, is implemented via explicit numerical time-stepping schemes with time gradients provided by the exact PDE. As a result, these methods depend on explicit access to both the PDE solution and the associated solver framework.

Time integration To compute temporal dynamics on these r-point spatial samples, we seek to evolve $\{\boldsymbol{f}_t^j\}_{j=1}^r \mapsto \{\boldsymbol{f}_{t+1}^j\}_{j=1}^r$. In the discrete setting, we leverage an explicit Euler time integrator (Ascher & Petzold, 1998) with step-size Δt,

$$\boldsymbol{f}_{t+1}^j = \boldsymbol{f}_t^j + \Delta t\, \dot{\boldsymbol{f}}_t^j \tag{7}$$

$$\dot{\boldsymbol{f}}_{t+1}^j = \dot{\boldsymbol{f}}_t^j + \Delta t\, \ddot{\boldsymbol{f}}_t^j \tag{8}$$

If $\boldsymbol{f}^j$ represents the position field, then, the one and only unknown in the equation above is the acceleration $\mathbf{A}_t^j = \ddot{\boldsymbol{f}}_t^j$, which is necessary for computing the velocity $\mathbf{V}_{t+1}^j = \dot{\boldsymbol{f}}_{t+1}^j$. We learn the acceleration field $\mathbf{A}_t^j := \ddot{\boldsymbol{f}}_t^j$ using the parameterization ϕ_Θ, which is then used in the explicit Euler update.

D Background

D.1 Operator learning

Here, we summarize the important ingredients of neural operators. For more details, please refer to Li et al. (2020a). Operator learning is a machine learning paradigm where a neural network is trained to map between infinite-dimensional function spaces. Let $\mathcal{G} : \mathcal{V} \to \mathcal{A}$ be a nonlinear map between the two function spaces $\mathcal{V}$ and $\mathcal{A}$. A neural operator is an operator parameterized by a neural network given by $\mathcal{G}_\theta \colon \mathcal{V} \to \mathcal{A}, \quad \theta \in \mathbb{R}^z$, that approximates this function mapping in the finite-dimensional space. The learning problem can be formulated as $\min_{\theta \in \mathbb{R}^z} \mathbb{E}_{v \sim D} \left[\|\mathcal{G}_\theta(v) - \mathcal{G}(v)\|_{\mathcal{V}}^2 \right]$, where $\| \cdot \|_{\mathcal{V}}$ is a norm on $\mathcal{V}$ and D is a probability distribution on $\mathcal{V}$. In practice, the above optimization is posed as an empirical risk-minimization problem, defined as $\min_{\theta \in \mathbb{R}^P} \frac{1}{N} \sum_{i=1}^N \|\mathcal{G}_\theta(v^{(i)}) - a^{(i)}\|_{\mathcal{V}}^2$.

Our **data-driven discretization invariant** reduced-order modeling framework for Lagrangian systems is constructed by parameterizing the PDE solution operator $\boldsymbol{\mathcal{J}}$ with a data-driven neural parameterization $\phi_\Theta : a \to \hat{\boldsymbol{f}}$, trained from a small set of point-wise samples $\{\boldsymbol{a}_t^i, \boldsymbol{f}_t^i\}_{i=1}^r$ such that $\hat{\boldsymbol{f}}_t \approx \boldsymbol{f}_t$ and $\phi_\Theta \in \mathbb{R}^{r \cdot d}$, where $r \cdot d \ll P \cdot d$. We interpret ϕ_Θ to be a surrogate for the solution operator, with $\{x^i\}_{i=1}^r$ representing a form of reduction of $(P \cdot d)$-point samples in $\boldsymbol{X}$. The architecture for ϕ_Θ leverages graph interaction network Battaglia et al. (2016); Sanchez-Gonzalez et al. (2020) to capture particle interaction dynamics. Figure 1 illustrates this distinction by comparing particle trajectories in elastic deformation systems modeled using our approach with discretization invariant CROM Chen et al. (2023).

D.2 Kernel methods and manifold learning

Kernel methods have been applied in manifold learning, where they serve as tools for constructing operators that act on functions defined over sampled data. Many such methods—including diffusion maps and Laplacian eigenmaps—can be interpreted as variants of kernel PCA Izenman (2012).

Belkin and Niyogi Belkin & Niyogi (2003) and Coifman et al. Coifman & Lafon (2006) studied *radially symmetric* kernels of the form $k(x, y) = h\left(\frac{\|x-y\|^2}{\varepsilon}\right)$ where $h : \mathbb{R}_+ \to \mathbb{R}$ is a decreasing function, typically Gaussian. These kernels are *local.* As such, they are isotropic and spatially invariant Berry & Sauer (2016).

Neural operator literature defines the kernel based **kernel integral transform**:

$$(\mathcal{K}f)(x) = \int_\Omega k(x, y)f(y)\, dy,$$

which is central to both manifold learning and neural operator frameworks. It maps input functions $f : \Omega \to \mathbb{R}^d$ to output functions via integration against k, and serves as the foundation for approximating function-to-function mappings.

In the context of manifold learning, local radially symmetric kernels induce a geometry on the embedded manifold. As the sample density increases, the kernel implicitly defines a metric structure through its interactions over neighborhoods on the manifold Berry & Sauer (2016). Consequently, such kernels not only enable dimensionality reduction but also act as geometric priors over data manifolds.

Finally, this kernel framework extends naturally to operator learning, where local solution maps—such as those arising from hyperbolic PDEs—can be effectively approximated using locally supported kernels Liu-Schiaffini et al. (2024).

D.3 Kernel preliminaries

Let $\Omega \subset \mathbb{R}^n$ and let $f : \Omega \to \mathbb{R}^d$ be a continuous function. We assume $f \in \mathcal{H}_K$, where $\mathcal{H}_K$ is a Reproducing Kernel Hilbert Space (RKHS) over Ω, associated with a kernel $K : \Omega \times \Omega \to \mathbb{R}$. For all $x \in \Omega$, the evaluation functional $L_x : f \mapsto f(x)$ is bounded and there exists a unique sample $K(x, \cdot) \in \mathcal{H}_K$ satisfying the reproducing property:

$$f(x) = \langle f, K(x, \cdot)\rangle_{\mathcal{H}_K}, \quad \forall f \in \mathcal{H}_K.$$

In particular, the kernel satisfies

$$\langle K(x, \cdot), K(y, \cdot)\rangle_{\mathcal{H}_K} = K(x, y), \quad \forall x, y \in \Omega,$$

which follows directly from the reproducing property.

Let $X = \{x_1, \dots, x_r\} \subset \Omega$ be a set of sample locations, and define the vector space of functions

$$\mathcal{H}_X := \text{span}\{K(x_i, \cdot) \mid x_i \in X\} = \{\sum_{i=1}^{N} c_i K(x_i, \cdot \mid x_i \in \Omega, c_i \in \mathbb{R}, N \in \mathbb{N}\} \subset \mathcal{H}_K.$$

The reproduction property implies that any $f \in \mathcal{H}_X$ allows for pointwise evaluation, given by the form

$$\langle f, K_x\rangle = f(x)$$

$$f(x) = \sum_{i=1}^{r} c_i K(x_i, x), \quad c_i \in \mathbb{R}.$$

The RKHS action is presented in the integral transform operator $\mathcal{K}$ Li et al. (2020b) defined by

$$(\mathcal{K}f)(x) := \int_\Omega K(x, y)f(y)\, dy,$$

which we discretize For a neighborhood $\mathcal{N}(x) \subset X$, we write

$$f(x) \approx \sum_{x_j \in \mathcal{N}(x)} w_j(x) K(x, x_j) f(x_j).$$

Replacing the kernel-weight product with a learned approximation, we introduce a parametric kernel $\psi_\theta : \Omega \times \Omega \to \mathbb{R}$, yielding the estimator

$$\hat{f}(x) := \sum_{x_j \in \mathcal{N}(x)} \psi_\theta(x, x_j) f(x_j).$$

This form approximates $\langle f, K(x, \cdot) \rangle_{\mathcal{H}_K}$ using a localized, data-driven surrogate ψ_θ in place of analytic kernels. Training proceeds by minimizing the empirical reconstruction loss over a set of query points:

$$\mathcal{L}_{\text{recon}} = \sum_{x \in \Omega_{\text{train}}} \left\| \hat{f}(x) - f(x) \right\|^2,$$

where $\hat{f}$ is computed via the learned kernel estimator above. When $\psi_\theta(x, x_j) \to K(x, x_j)$ and $\mathcal{N}(x) = X$, the estimator recovers the exact RKHS interpolant.

This construction provides a localized, continuous, and mesh-free approximation of the RKHS projection operator. The reproducing property guarantees the estimator recovers pointwise field values. For more background on RKHS and kernel approximation, see (Berlinet & Thomas-Agnan, 2011; Schölkopf & Smola, 2002).

D.4 Graph interaction network

The graph interaction network proposed in Battaglia et al. (2016) learns a relation-centric function f that encodes spatial interactions between the interacting nodes within a system as a function of their interaction attributes r. This can be represented as $e_{t+1} = f_R(x_{1,t}, x_{2,t}, r)$. A node-centered function predicts the temporal dynamics of the node as a function of the spatial interactions as follows $x_{1,t+1} = f_o(e_{t+1}, x_{1,t})$. In a system of m nodes, the spatial interactions are represented as a graph, where the neighborhood is defined by a ball of radius r. This graph is represented as $G(O, R)$, where O is the collection of objects and R is the relationships between them. The interaction between them is defined as $\mathcal{I}(G) = f_o(a(G, X, f_R(\langle x_i, x_j, r_{ij} \rangle)))$, Where a is an aggregation function that combines all the interactions, X is the set of external effects, not part of the system, such as gravitational acceleration, etc.

E Algorithms

The following section presents key algorithms that are helpful for implementing the concepts presented in the paper

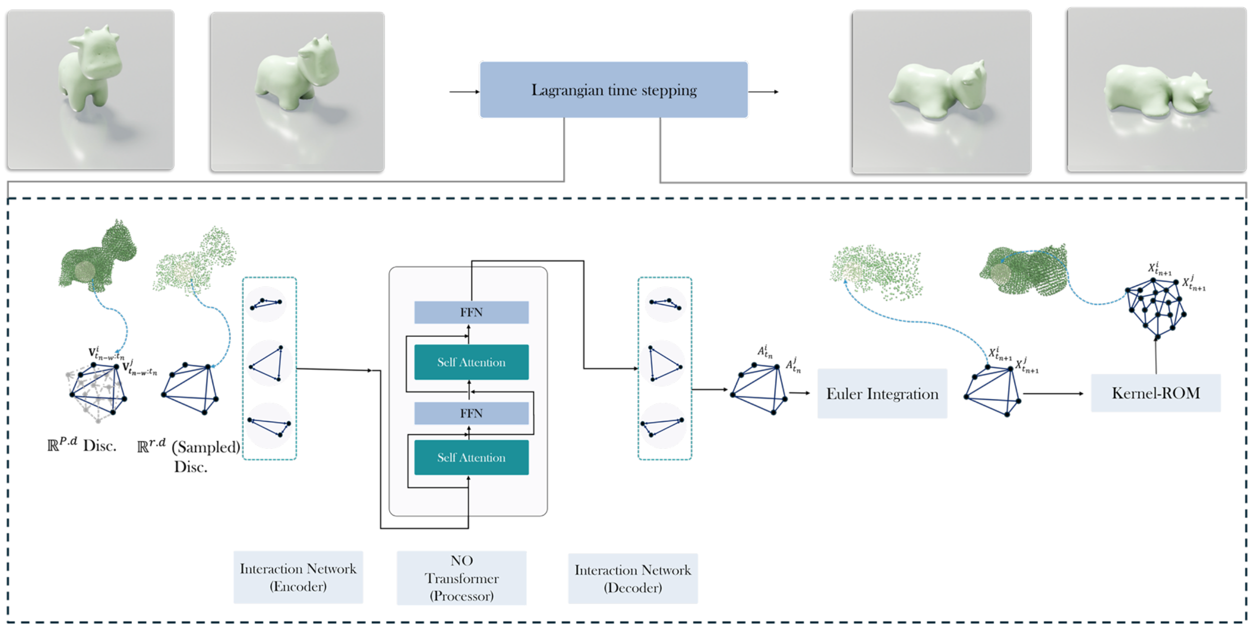


Figure 8: **Autoregressive Lagrangian PDE operator and Kernel network parameterization (Kernel-ROM).** The neural time-stepper ϕ_Θ (Encoder-Processor-Decoder) predicts the acceleration of a Lagrangian system $\mathbf{A}_t$ at time t from the past w velocity instances $\mathbf{V}_{t-w:t}$. From the predicted accelerations, we leverage standard Euler integration to obtain the predicted positions. The rightmost kernel-ROM is used to efficiently evaluate the deformation field at arbitrary locations. This is summarized in the flow diagram fig. 9

Algorithm 1 Kernel Integral Transform Li et al. (2024), Neural Operator Library

```
Input:
  • $x \in \mathbb{R}^{m\times d}$                                  // Query points
  • $y \in \mathbb{R}^{n\times d}$                                  // Support points
  • $f_y \in \mathbb{R}^{n\times r}$ (optional)                     // Function values
  • neighbors                          // CSR format: indices and row splits
  • $k(\cdot)$                                  // Parametrized kernel (e.g., MLP)
  • type $\in \{\text{linear}, \text{nonlinear}, \ldots\}$          // Transform type
  • weights $\in \mathbb{R}^n$ (optional)
Output:
  • out $\in \mathbb{R}^{m\times r}$                                // Transformed values
1: if $x$ is None then
2:     $x \leftarrow y$
3: end if
4: $y_{\text{nb}} \leftarrow y$[neighbors.indices]
5: if $f_y$ is given then
6:     $f_{\text{nb}} \leftarrow f_y$[neighbors.indices]
7: end if
8: num_reps $\leftarrow$ neighbors.row_splits$[1:]$ $-$ neighbors.row_splits$[:-1]$
9: $x_{\text{rep}} \leftarrow$ repeat_interleave($x$, num_reps)
10: $\phi \leftarrow$ concat($x_{\text{rep}}, y_{\text{nb}}$)
11: if type is nonlinear and $f_y$ is given then
12:     $\phi \leftarrow$ concat($\phi, f_{\text{nb}}$)
13: end if
14: $\kappa \leftarrow k(\phi)$                                      ▷ Apply kernel function
```

Algorithm 2 Kernel-based ROM Inference

```
Input:
  • sampled_ic $\in \mathbb{R}^{m\times d}$          // Sampled initial points
  • full_ic $\in \mathbb{R}^{n\times d}$             // Full initial state
  • sampled_f $\in \mathbb{R}^{m\times d}$           // Initial state in sampled space
  • radius                                           // Neighborhood radius
Output:
  • out $\in \mathbb{R}^{n\times d}$                 // Updated ROM state
1: neighbors $\leftarrow$ NeighborSearch(sampled_ic, full_ic, radius)
2: out $\leftarrow$ IntegralTransform.forward($x =$ full_ic, $y =$ sampled_ic, $f_y =$ sampled_f, neighbors)
3: return out
```

Algorithm 3 GNN Time-Stepper Inference

```
Input:
$G = (V, E)$: input graph with nodes $V$ and edges $E$
$x \in \mathbb{N}^{|V|\times 1}$: categorical node types
$p \in \mathbb{R}^{|V|\times d}$: recent positions and velocities
$e \in \mathbb{R}^{|E|\times d_e}$: edge attributes (displacement, distance)
edge_index $\in \mathbb{N}^{2\times|E|}$: sender and receiver indices
recent_pos $\in \mathbb{R}^{|V|\times d}$: current position of each node
Output:
$\hat{a} \in \mathbb{R}^{|V|\times r}$: predicted acceleration (or equivalent output)
1: Step 1: Node and Edge Feature Initialization
2: $h_v \leftarrow$ concat(EmbedType($x$), $p$)     ▷ Embed categorical type and concatenate with pos/velocity
3: $h_v \leftarrow$ NodeInputMLP($h_v$)
4: $h_e \leftarrow$ EdgeInputMLP($e$)
5: Step 2: Message Passing (Encoder)
6: for $i = 1$ to n_mp_layers do
7:     $h_v, h_e \leftarrow$ GNNLayer$_i$($h_v$, edge_index, $h_e$, node_dist)
8: end for
9: Step 3: Global Transformation via GNOT Layer
10: $h_v \leftarrow$ GNOTLayer($h_v$, recent_pos)
11: Step 4: Message Passing (Decoder)
12: for $i = 1$ to n_mp_layers do
13:     $h_v, h_e \leftarrow$ GNNLayerOut$_i$($h_v$, edge_index, $h_e$, node_dist)
14: end for
15: Step 5: Node-wise Output Projection
16: $\hat{a} \leftarrow$ NodeOutputMLP($h_v$)
17: return $\hat{a}$
```

F Architecture outline

The schematic in fig. 9 illustrates the full sequence of operations performed during a single time-step of the proposed method. It delineates the interaction between the time-stepping mechanism, the kernel-based reduced-order model (kernel-ROM), the Euler integration scheme, and the spatial sampling strategy used to update the system state.

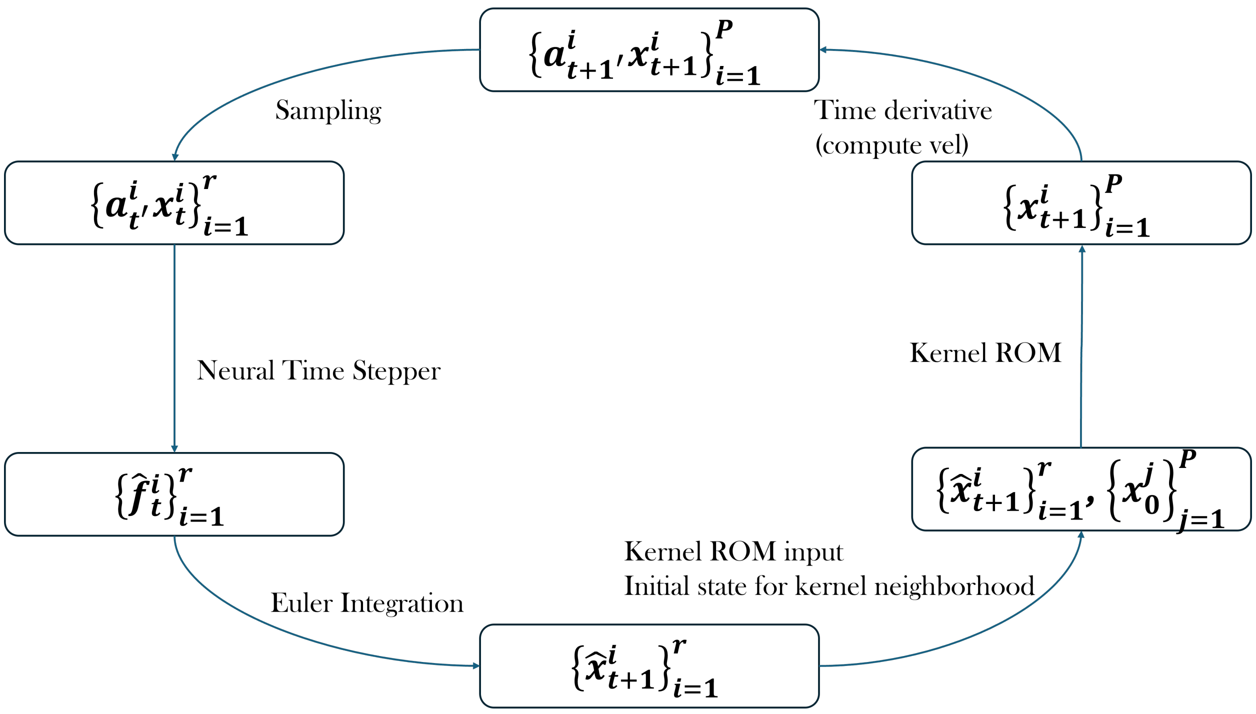

$\{a^i_{t+1}, x^i_{t+1}\}^P_{i=1}$
Sampling
Time derivative (compute vel)
$\{a^i_{t}, x^i_{t}\}^r_{i=1}$
$\{x^i_{t+1}\}^P_{i=1}$
Neural Time Stepper
Kernel ROM
$\{\hat{f}^i_t\}^r_{i=1}$
$\{\hat{x}^i_{t+1}\}^r_{i=1}, \{x^j_0\}^P_{j=1}$
Kernel ROM input
Initial state for kernel neighborhood
Euler Integration
$\{\hat{x}^i_{t+1}\}^r_{i=1}$

Figure 9: **Single-step flow diagram:** Overview of the forward pass used to infer the system state at the next time-step via Kernel-ROM, time-stepping, Euler integration, and spatial sampling. As ϕ_Θ takes as input a window of past-states, we require the same set of samples for a trajectory evolution

G Experimental setup details

G.1 PDE operator setup and hyperparameters

Data representation To train ϕ_Θ, we create a window of w point cloud velocity sequences as the input defined on the nodes, with the pointwise acceleration as the output. We define $\{\mathbf{x}^i_t\} \in \mathbb{R}^{r \cdot d}$ to be the pointwise positions of r particles within a d-dimensional system at time t. A sequence of T time steps is denoted as $\mathbf{x}_{0:T} = (\mathbf{x}_0, \ldots, \mathbf{x}_T)$. In particular, $\{\mathbf{x}^0_t, \ldots, \mathbf{x}^r_t\}$ are the individual particles within the system at time t. We define velocity at time n as $\mathbf{v}_t$ as $\mathbf{x}_t - \mathbf{x}_{t-1}$. Similarly, acceleration at time n is defined as $\mathbf{a}_t = \mathbf{v}_t - \mathbf{v}_{t-1}$. In all these cases, Δt is set to one for simplicity. To train the model, we follow the same procedure as (Sanchez-Gonzalez et al., 2020). We additionally encode the particle types (water, sand, plasticine, etc.) as embeddings, which is useful for multimaterial simulations. The graph is constructed as a `radius_graph`, defined within `pytorch geometric` library.

Boundary representation To enforce the boundaries of the system, the node feature includes the past w velocity fields as well as the distance of the most recent position field to the upper (b_u) and lower (b_l) boundaries of the computational domain, given by $\mathcal{D} = [(x - b_l)/\rho, (b_u - x)/\rho], \forall x$, where ρ is the radius of the graph.

G.2 Hyperparameters

The models were implemented using `Pytorch` library and trained on `CUDA`. The graphs were built using `Pytorch Geometric` module. All models were trained on `NVIDIA RTX 3060` GPUs.

ϕ_Θ time-stepper The graph is constructed using `radius_graph` defined in `Pytorch Geometric`. The node features and the edge features, which include the distance from the boundary points, are encoded into latent vectors of size 128 using 2 MLPs. The encoder uses two layers of interaction network. The latents are then processed by two layers of Neural Operator Transformer. The decoder layers are symmetric to the encoder layers. The NOT block uses 4 attention heads, branch and trunk sizes of 32, and an output dim of 128.

Kernel-ROM The network uses discrete integral transform, available within the `neural operator` library. The model uses `linear` kernel transformations with `gelu` activations. The model uses a sequence of `MLP` layers to parameterize the kernel, with varying channels for each physical system. It is set to [32, 64] (most systems), [128, 256] (fluid). The neighborhood radius is a tuned hyperparameter, that varies with sample size.

Optimizers Optimization is done with Adamax optimizer, with an initial learning rate of 1e-4, weight decay of 1e-6 and a batch size of 4. The learning rate was decayed exponentially from 10^{-4} to 10^{-6} using a scheduler, with a gamma of $0.1^{1/5e6}$

H Additional dataset details

We model the following classes of materials - elastic, plasticine, granular, Newtonian fluids, non-Newtonian fluids, and multi-material simulations. Notably, this framework is compatible with data generated using different solvers such as Finite Element Method (FEM) Hughes (2012), Material Point Method (MPM) Jiang et al. (2016), or Smooth Particle Hydrodynamics (SPH) Monaghan (1992).

Plasticine (von Mises Yield) Using the `NCLAW` simulator, we generated 100 trajectories of 400 time steps ($\text{dt} = 5e-4$) with random initial velocities and 4 different geometries - Stanford bunny, Stanford armadillo, blub (goldfish), and spot (cow). The trajectories are modeled using Saint Venant-Kirchoff elastic model, given by

$$\mathbf{P} = \mathbb{U}(2\mu\epsilon + \lambda tr(\epsilon))\mathbb{U}^T \tag{9}$$

where λ and μ are Lamé constants, $\mathbf{P}$ is the second Piola-Kirchoff stress and ϵ is the strain. $\mathbb{U}$ is obtained by applying SVD to the deformation gradient $\mathbf{F} = \mathbb{U}\mathbf{\Sigma}\mathbf{V}^\mathbf{T}$. The von Mises yield condition is denoted by

$$\delta\gamma = \|\hat{\epsilon}\| - \frac{\tau_Y}{2\mu} \tag{10}$$

where ϵ is the normalized Henky strain, τ_Y is the yield stress.

Granular material (Drucker Prager sand flows) We trained the model on 2 datasets to simulate granular media. We generated 100 trajectories at 300 time steps, using `NCLAW` simulator and on the 2D Sand dataset released by Pfaff et al. (2020). The Drucker-Prager elastoplasticity is modeled by the same Saint Venant–Kirchhoff elastic model, given by Equation 9. Additionally, the Drucker-Prager yield condition is applied such that

$$tr(\epsilon) > 0 \quad or \quad \delta\gamma = \|\hat{\epsilon}\| + \alpha\frac{(3\lambda + 2\mu)tr(\epsilon)}{2\mu} > 0 \tag{11}$$

where, $\alpha = \sqrt{2/3}\frac{2sin\theta}{3-sin\theta}$ and θ is the frictional angle of the granular media.

Elasticity To simulate elasticity, we generated simulations using meshes from Thingi10k dataset Zhou & Jacobson (2016). We generated 24 trajectories, with 200 time steps, for 6 geometries to train the model. The elasticity is modeled using stable neo-Hookean model, as proposed in Smith et al. (2018). The energy is denoted by

$$\Psi = \frac{\mu}{2}(I_C - 3) + \frac{\lambda}{2}(J-\alpha)^2 - \frac{\mu}{2}log(I_C + 1) \tag{12}$$

where I_C refers to the first right Cauchy-Green invariant and J is the relative volume change. μ and λ are Lamé constants. The corresponding Piola-Kirchoff stress is given by

$$\mathbf{P} = \mu\Big(1 - \frac{1}{I_C + 1}\Big)\mathbf{F} + \lambda(J-\alpha)\frac{\partial J}{\partial \mathbf{F}} \tag{13}$$

where $\mathbf{F}$ is the deformation gradient.

Newtonian fluids For Newtonian fluids, In the 2D setting, we use `WaterDrop` dataset created by Pfaff et al. (2020), which is generated using the material point method (MPM). For the 3D setting, we generated 100 trajectories with random initial velocity, each spanning 1000 time steps at a dt of $5e-3$. This dataset was prepared using the `NCLAW` framework. These are modeled as weakly compressible fluids, using fixed corotated elastic model with $\mu = 0$. The Piola-Kirchoff Stress is given by

$$\mathbf{P} = \lambda J(J-1)\mathbf{F}^{-T} \tag{14}$$

Non-Newtonian fluids To train the model on non-Newtonian fluids, we used the `Goop` and `Goop-3D` datasets.

Multimaterial We simulated multi-material trajectories in 2D using the dataset published by Pfaff et al. (2020).

I Training setup

Time-stepper We follow the exact training procedure outlined in Sanchez-Gonzalez et al. (2020) for ϕ_Θ. We use the 1-step loss function over a pair of consecutive time steps k and $k+1$, imposing a strong inductive bias towards a Markovian system. Each system is trained for 5 million steps.

The model is validated by full rollouts on 10 held-out validation sets per material simulation, with performance measured by the MSE between predicted particle positions and ground-truth particle positions.

Kernel-ROM To train the Kernel-ROM, we define the neighborhood over samples defined at $t = 0$, i.e. $\{x_0^i\}_{i=1}^P$ and $\{x_0^j\}_{j=1}^r$. For all the subsequent time-steps, we leverage this neighborhood. This enables us to evolve $\{\hat{\boldsymbol{f}}\}$ using the spatial information of the state at $t = 0$. The input function for the model is point-wise positions at a given time-step t for non-fluid systems and deformation $x_t - x_0$ for fluid systems. We generate these using ϕ_Θ. Each system is trained for 30,000 steps and evaluated on unseen discretizations and trajectories.

J Additional results

2D simulations Figure 10 presents qualitative comparisons between ground truth and $(P \cdot d)$-point predictions produced by GIOROM on various 2D simulations. The subfigures depict a range of dynamic behaviors: (a) granular flow, (b) soft body motion under gravity, (c) external force acting on a highly elastic material, and (d) coupled interactions between granular media and Newtonian fluids.

K Ablations

Number of message-passing layers We show that the key bottleneck in terms of speed is the message-passing operation within the Interaction Network encoder and decoder.

Table 8: This table shows that the number of message-passing layers results in a negligible improvement in rollout Loss.

NUM. MESSAGE PASSING LAYERS	CONNECTIVITY	INPUT SIZE	INFERENCE TIME/STEP	LOSS
2	0.077	2247	3.6	0.0008
4	0.077	2247	3.8	0.0009
6	0.077	2247	4.2	0.0014
8	0.077	2247	4.3	0.0009

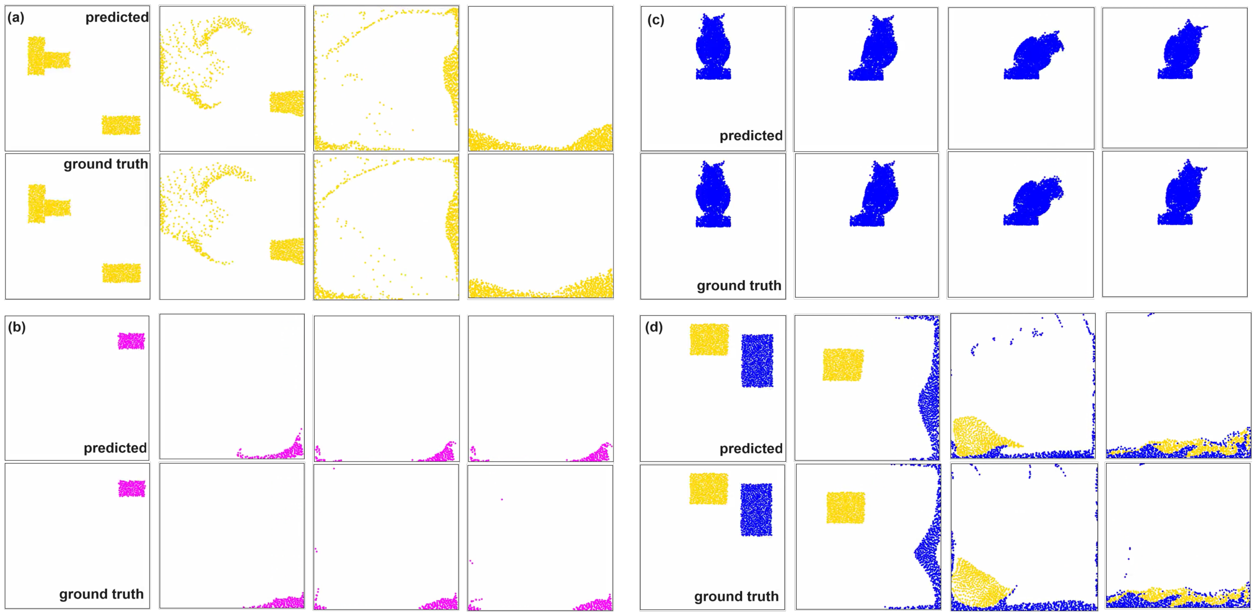


Figure 10: **2D point-cloud simulations:** $(P \cdot d)$-point inference results using GIOROM. **(a)** Granular flow; **(b)** Soft body under gravity; **(c)** Elastic response to external force; **(d)** Coupled fluid-granular interactions.

K.1 Sampling strategy and rollout loss

We compared different sampling strategies against the rollout Loss (MSE). The results are presented in Table 9.

Table 9: Comparison of different sampling and graph construction strategies against rollout MSE on Water-2D dataset

SAMPLING STRATEGY	GRAPH TYPE	ROLLOUT MSE
RANDOM	RADIUS	0.0098
RANDOM	DELAUNAY	7.017
FPS	RADIUS	0.0097
FPS	DELAUNAY	8.04

L Additional discussions

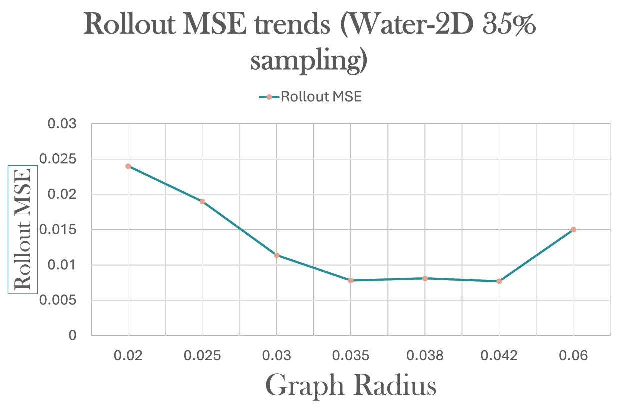
L.1 Understanding the correlation between sparsification and performance

We study the relationship between performance and sample size to understand how the structural fidelity of the interaction graph degrades the predictions under extreme sparsifications. Our analysis focuses on a small Water2D system (678 particles), where the graph structure can be explicitly visualized using `networkx`, as well as a larger Plasticine3D system. We consider different sampling levels to examine how graph sparsification impacts this relationship.

Figure 11 presents rollout MSE at two sampling ratios—35% and 11%—for the Water2D system. At 35% sampling, we observe a trend: rollout MSE consistently drops with improving the connectivity. However, with further modifications to the radius, the MSE increases. This suggests that beyond a point, the addition of edges leads to degradations. The visual rollout in fig. 12 shows that improving connectivity improves the performance upto a limit.

In contrast, at 11% sampling, the rollout MSE remain high regardless of radius, and no clear trend emerges. As shown in fig. 13, the predicted rollouts diverge significantly from the ground truth. The third column in each frame represents the reduced-space ground truth at $(r \cdot d)$ points, which itself is visually and physically distinct from the full-resolution system. This indicates that at extreme sparsity, the reduced graph fails to retain sufficient physical characteristics, rendering it unsuitable for accurate inference.

Figure 14 extends this analysis to the Plasticine3D system. At 22% sampling, we again observe that rollout MSE initially decreases, but as the radius increases further, this trend reverses. However, at 3.3% sampling, there is no correlation.



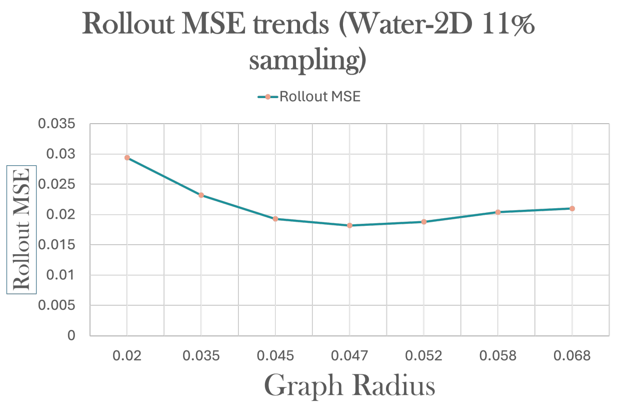


Figure 11: **Water2D: Radius vs. rollout MSE.** Left: 35% sampling. Right: At 11%, there is a weak correlation between them.

L.2 Limitations of kernel-ROM to extreme sparsifications

The kernel-ROM model employs an implicit neural representation to approximate discrete kernel integral transforms over sparse point-wise function evaluations. However, under extreme sparsification, the quality of this representation degrades significantly.

When the number of supervised function evaluations is severely limited, increasing the neighborhood radius does not lead to meaningful improvements. Instead, the model interpolates around the sparse support, leading to point clustering near known estimates or interpolation artifacts that resemble piecewise-linear transitions. These effects arise not due to kernel parameter choices, but due to a fundamental lack of information in the reduced space: the interpolated function no longer reflects a smooth or physically meaningful structure.

This limitation is evident in fig. 15, where we interpolate a 2-million-point Dragon mesh from three different supervision levels: 700 points (0.035%), 20K points (1%), and 60K points (3%). At 700 points, the interpolated output exhibits dense clustering and abrupt discontinuities. The gap between 0.035% and 1% remains substantial. The top-row result makes clear that in the regime of extreme sparsification, the model fails to maintain coherent spatial representations, regardless of neighborhood size or kernel formulation.

This example illustrates the lower bound on sparsity below which the learned representation fails to extrapolate. In such cases, the reduced-space support does not sufficiently reflect the full system's physical characteristics, making the reconstruction problem ill-posed.

L.3 Computational cost of kernel-ROM

We evaluated the computational cost of Kernel-ROM with respect to neighborhood radius, input discretization size, and MLP architecture. Figure 16 (left) shows that, with a fixed input discretization of 3000 points and MLP configuration [128, 256], inference time increases from 3.4 ms to 7.6 ms as the radius grows from 0.009 to 0.035. GPU memory usage also rises significantly, from 122 MB to approximately 4.3 GB. The right panel of Figure 16 shows inference time and memory usage as functions of input size at a fixed radius of 0.015 and

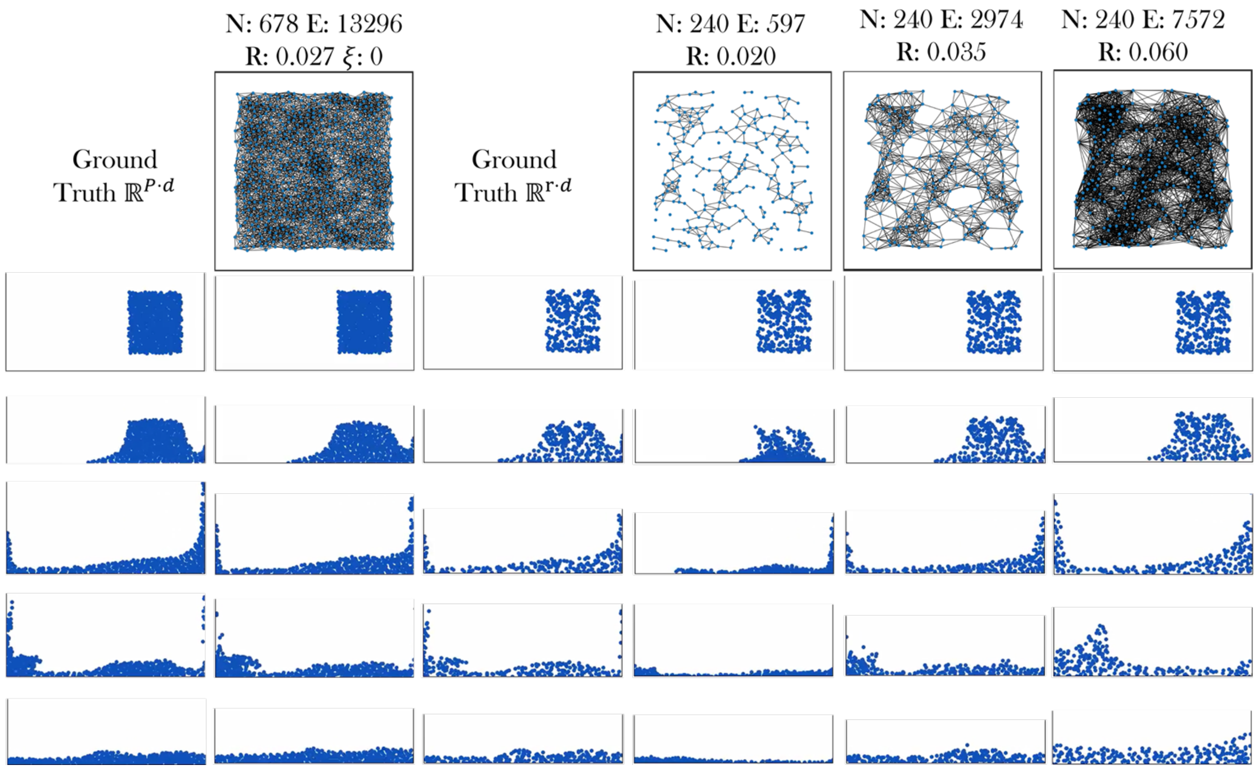


Figure 12: **Water2D at 35% sampling.** First two columns: full-space ground truth and prediction on $(P \cdot d)$ points. Third column: reduced-space ground truth at $(r \cdot d)$ points. Right columns: predictions at varying radii. Rollouts remain visually and physically consistent with the full system at R=0.035

the same MLP. Increasing input points from 3000 to 12000 results in inference time increasing from 4.1 ms to 12.8 ms, and GPU memory usage from 414 MB to 1.6 GB.

Figure 17 (left) presents inference time and memory usage for different MLP architectures at fixed radius 0.015 and input size 3000. Increasing the MLP size from [32, 64] to [1024, 1024] raises inference time from 3.8 ms to 8.2 ms and memory usage from 119 MB to 2.1 GB. The right panel shows rollout mean squared error (MSE) decreases with increasing sampling percentage, from 0.045 at 0.035% sampling to 0.0034 at 15% sampling.

These results demonstrate that increasing radius, input discretization, or model size leads to higher computational cost, while higher sampling percentages improve accuracy.

L.4 GNNs as neural operator approximations

We now argue that Graph Neural Networks (GNNs), under a suitable construction, can be interpreted as discretizations of kernel integral operators. In particular, GNNs can be viewed as data-driven approximations to a class of neural operators. However, GNNs are senstive to input graph topologies, and cannot generalize to arbitrary graph structures defined over a discretization. This can be seen in fig. 18, where the model can generalize to different sampling strategies but not to different graph structures defined over the same sample size.

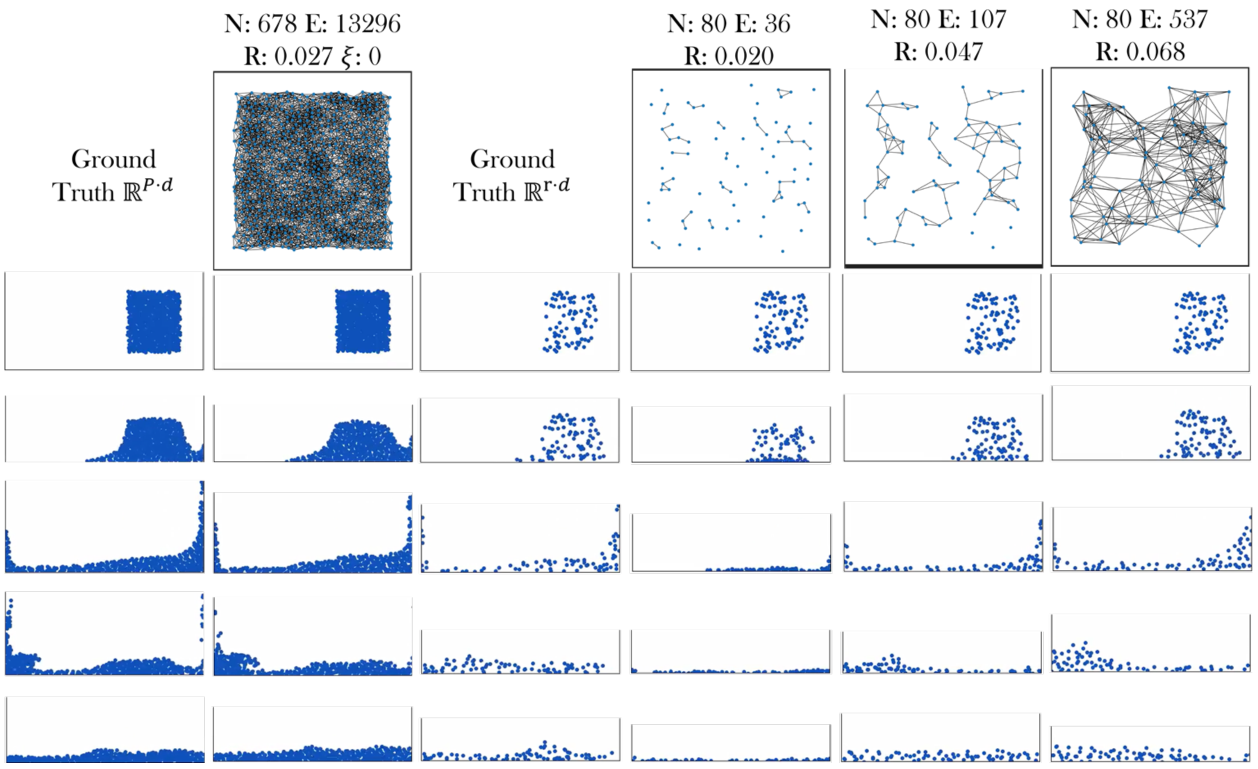
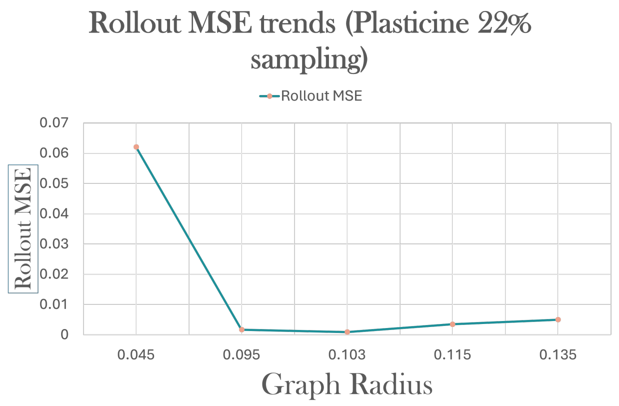


Figure 13: **Water2D at 11% sampling.** All predictions diverge from ground truth. Even at increased radii, rollout MSE remains high. Reduced-space ground truth itself is physically dissimilar to the original system, limiting learnability.



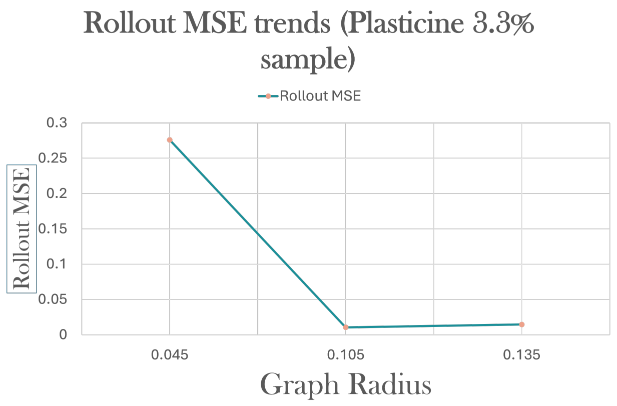


Figure 14: **Plasticine3D: rollout MSE.** Left: 22% sampling shows consistent correlation. Right: At 3.3% sampling, correlation degrades as the reduced graph fails to encode sufficient structure for accurate rollout. At 35%, rollout MSE reduces to 9.2e-4, while it plateaus at 0.01 at 3.3%

L.4.1 GNNs as Monte-Carlo estimators

We discuss the following propositions made in Li et al. (2020b). Let $D \subset \mathbb{R}^d$ be a compact domain. For each $x \in D$, let ν_x be a fixed Borel measure on D. We define a kernel integral operator of the form:

$$(\mathcal{K}_\phi v)(x) := \sigma \left(Wv(x) + \int_D \kappa_\phi(x, y) v(y) \, d\nu_x(y) \right),$$

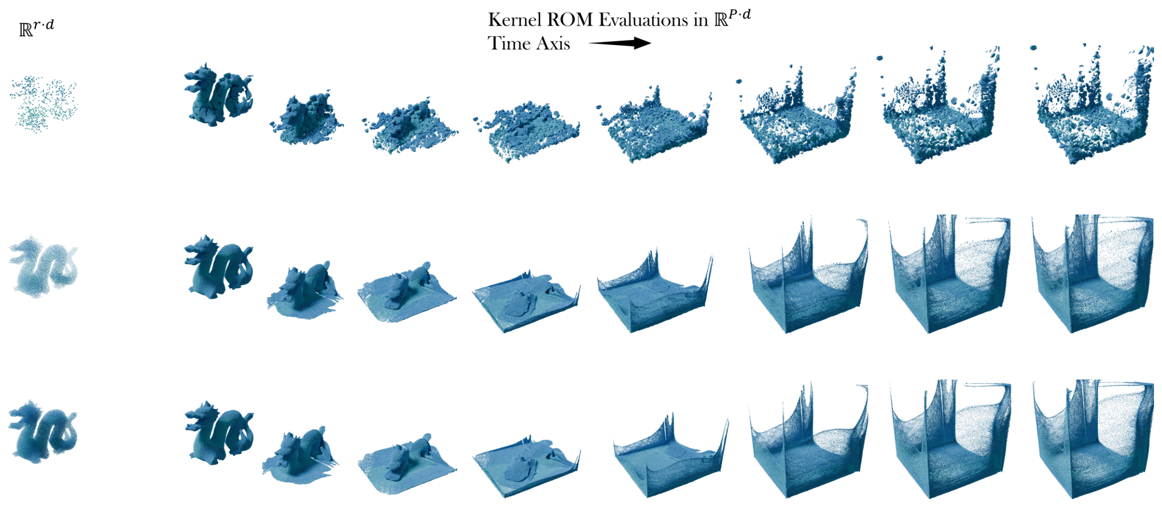
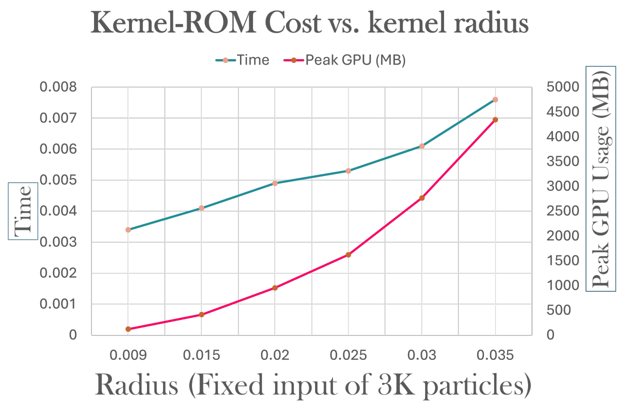


Figure 15: **Degradation under extreme sparsification.** All rows show kernel-ROM interpolation on a 2-million-point Dragon mesh. **Top:** 700 function evaluations (0.035%) lead to clustered and discontinuous outputs. **Middle:** 20K points (1%) **Bottom:** 60K points (3%) produces smoother reconstruction.



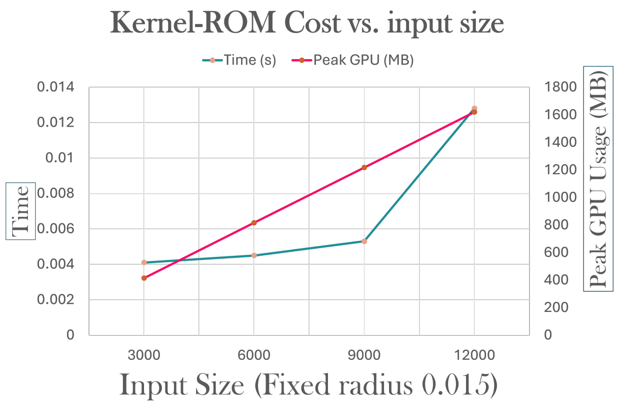


Figure 16: Kernel-ROM inference time and GPU memory usage. Left: Variation with neighborhood radius at fixed input size and MLP configuration. Right: Variation with input discretization size at fixed radius and MLP. These are computed on collision dataset with 84K particles

where $\sigma : \mathbb{R} \to \mathbb{R}$ is a fixed nonlinearity applied elementwise, $W \in \mathbb{R}^{n\times n}$ is a learned matrix, and κ_ϕ : $\mathbb{R}^{2(d+1)} \to \mathbb{R}^{n\times n}$ is a neural network parameterized by ϕ. The arguments to κ_ϕ can include geometric and positional information such as $(x, y, x - y, \|x - y\|)$.

In practice, we do not have access to the continuum D, and instead work with a finite point cloud $\{x_1, \ldots, x_K\} \subset D$. We approximate the measure ν_x by an empirical distribution over a neighborhood $\mathcal{N}(x)$ around each x, typically defined via a radius graph. Then, the integral operator is approximated by a sum:

$$(\mathcal{K}_\phi v)(x) \approx \sigma\left(W v(x) + \frac{1}{|\mathcal{N}(x)|} \sum_{y\in\mathcal{N}(x)} \kappa_\phi(e(x, y))v(y) \right),$$

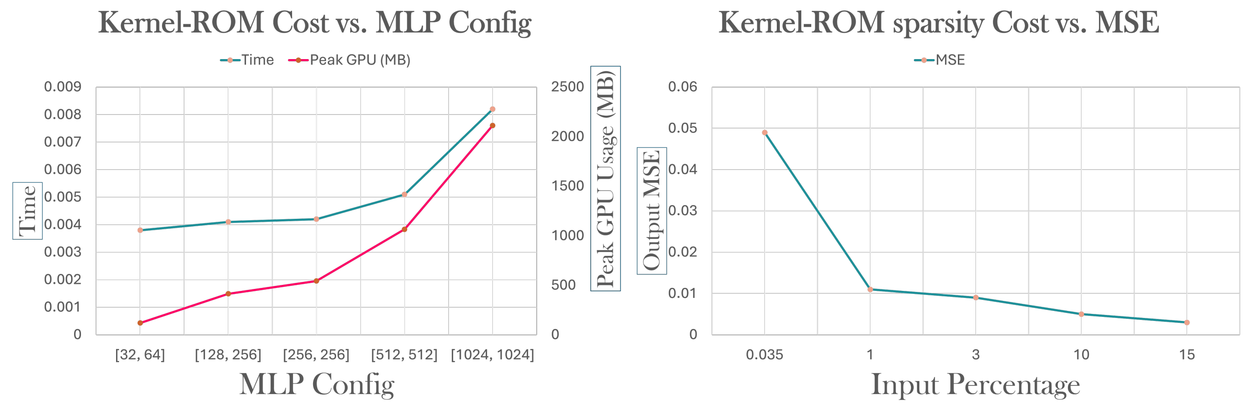


Figure 17: Left: Inference time and memory usage for different MLP sizes at fixed radius and input size. (collision dataset 84K particles) Right: Rollout mean squared error versus sampling percentage for the dragon dataset (2 million particles), providing an empirical lower bound for r

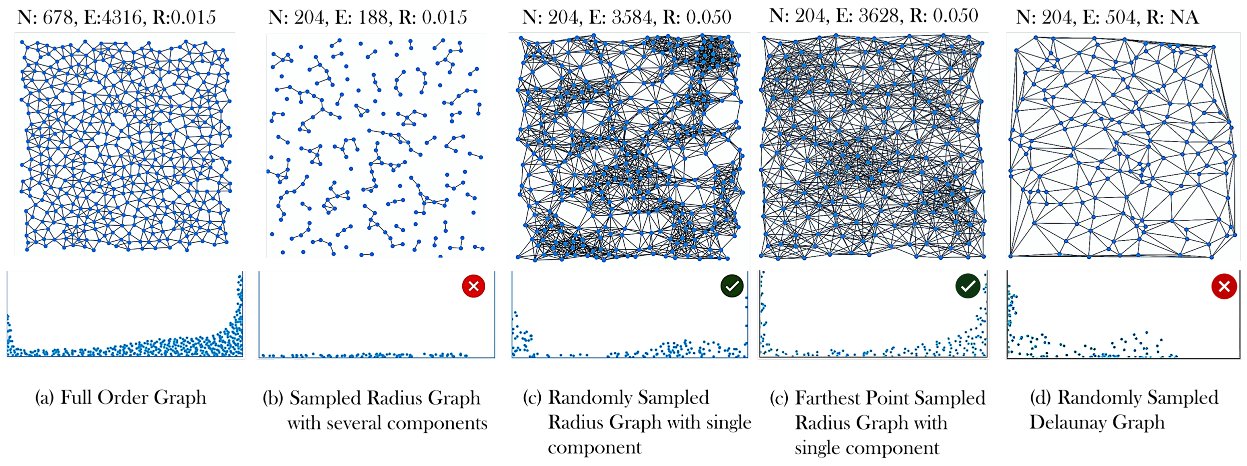

N: 678, E:4316, R:0.015
N: 204, E: 188, R: 0.015
N: 204, E: 3584, R: 0.050
N: 204, E: 3628, R: 0.050
N: 204, E: 504, R: NA
(a) Full Order Graph
(b) Sampled Radius Graph with several components
(c) Randomly Sampled Radius Graph with single component
(c) Farthest Point Sampled Radius Graph with single component
(d) Randomly Sampled Delaunay Graph

Figure 18: This figure illustrates the critical role of radius-based graphs in maintaining discretization invariance and preserving connectivity in graph structures. Both random sampling and farthest-point sampling produce consistent embeddings due to stable neighborhood definitions, whereas altering the graph construction method breaks the model.

where $e(x, y)$ denotes edge features constructed from (x, y). This yields the following message-passing update rule:

$$v^{t+1}(x) = \sigma\left(Wv^t(x) + \frac{1}{|\mathcal{N}(x)|}\sum_{y\in\mathcal{N}(x)} \kappa_\phi(e(x, y))v^t(y)\right).$$

We interpret this as a discretization of the continuum operator $\mathcal{K}_\phi$, where the integral is approximated by local aggregation, and the kernel κ_ϕ is represented by a shared neural network acting on edge features.

The key modeling decision is that κ_ϕ defines a $K \times K$ block kernel matrix, where each entry $\kappa_\phi(x_i, x_j)$ is a matrix in $\mathbb{R}^{n\times n}$. The parameters ϕ are shared across all edge pairs. This sharing ensures that the operator is independent of the number and arrangement of discrete points, so long as the neighborhoods are constructed consistently. Therefore, the learned operator exhibits discretization invariance.

L.4.2 Discretization or resolution?

Figure 19 presents four point clouds of identical size and geometry but with differing point configurations. Despite many non-overlapping locations, the predictions across all four discretizations remain consistent. This visual evidence supports the notion of learned model generalizing across different samplings of the same underlying domain. It is important to distinguish this from resolution, which concerns the density or number of points used to represent the domain. Discretization refers to the specific arrangement of a fixed number of points; resolution, by contrast, changes the number of points altogether. This is a subtle difference.

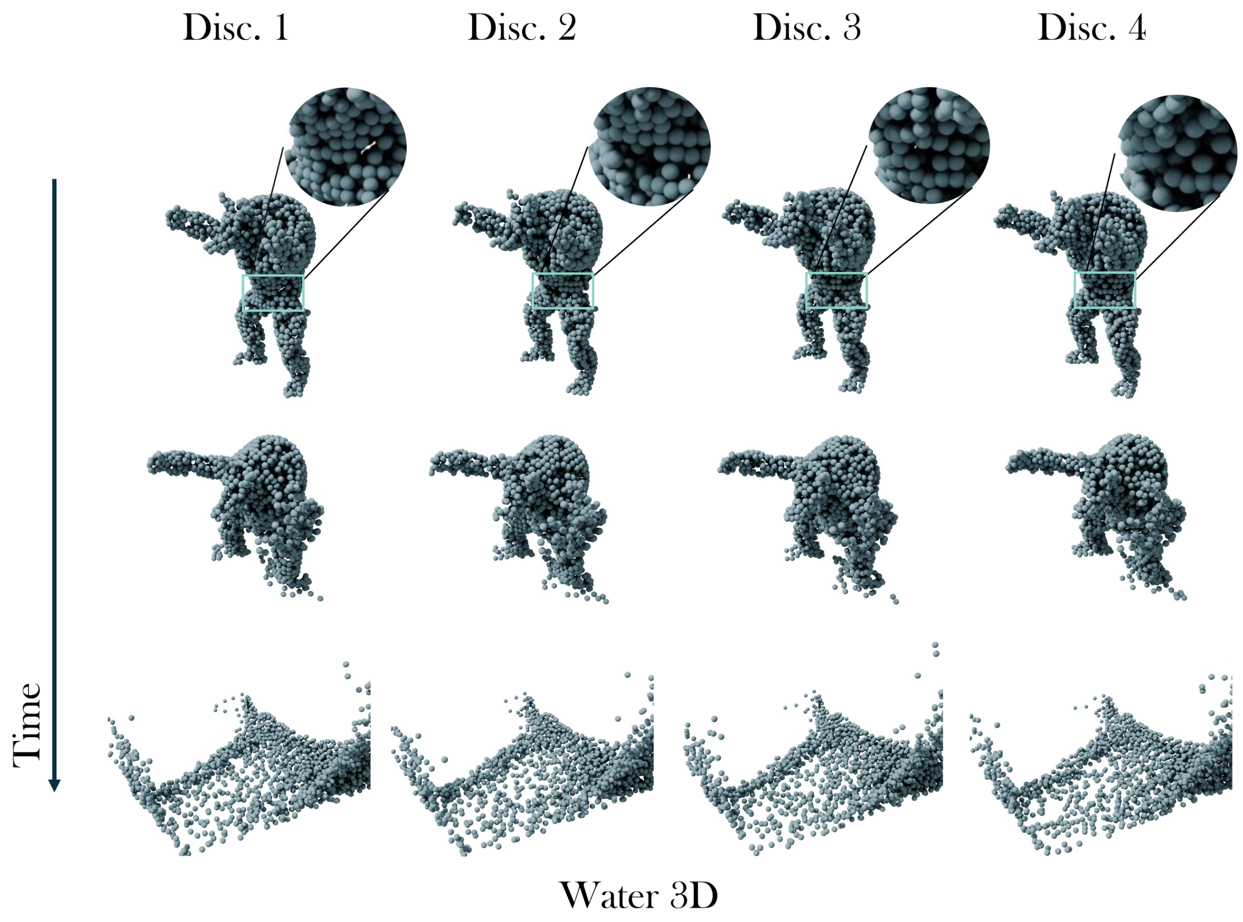


Figure 19: **Discretization invariance on the same geometry.** We construct four different point clouds sampled from the same domain, with equal size but varying locations. On each, we build a radius graph with fixed connectivity radius and apply the same learned GNN layer. Despite the different discrete realizations, the resulting outputs are consistent.

Implicit handling of self-contact Training data were generated using MPM solvers that handle self-contact implicitly via a background Eulerian grid, applicable to both solids and fluids. As a result, the model implicitly learns self-contact behavior from data. Future work could explore improved sampling strategies in regions prone to self-collision for enhanced resolution.