# Implicit Neural Spatial Representations for Time-dependent PDEs

**Honglin Chen** [*1] **Rundi Wu** [*1] **Eitan Grinspun** [2] **Changxi Zheng** [1] **Peter Yichen Chen** [3 1]

## Abstract

Implicit Neural Spatial Representation (INSR) has emerged as an effective representation of spatially-dependent vector fields. This work explores solving time-dependent PDEs with INSR. Classical PDE solvers introduce both temporal and spatial discretizations. Common spatial discretizations include meshes and meshless point clouds, where each degree-of-freedom corresponds to a location in space. While these *explicit* spatial correspondences are intuitive to model and understand, these representations are not necessarily optimal for accuracy, memory usage, or adaptivity. Keeping the classical temporal discretization unchanged (e.g., explicit/implicit Euler), we explore INSR as an alternative spatial discretization, where spatial information is *implicitly* stored in the neural network weights. The network weights then evolve over time via time integration. Our approach does *not* require any training data generated by existing solvers because our approach is the solver itself. We validate our approach on various PDEs with examples involving large elastic deformations, turbulent fluids, and multi-scale phenomena. While slower to compute than traditional representations, our approach exhibits higher accuracy and lower memory consumption. Whereas classical solvers can dynamically adapt their spatial representation only by resorting to complex remeshing algorithms, our INSR approach is intrinsically adaptive. By tapping into the rich literature of classic time integrators, e.g., operator-splitting schemes, our method enables challenging simulations in contact mechanics and turbulent flows where previous neural-physics approaches struggle. Videos and codes are available on the project page.[1]

## 1. Introduction

Implicit neural spatial representation (INSR) (Park et al., 2019; Xie et al., 2021) parameterizes a spatially-dependent vector field with a neural network. It has been proven to be instrumental in computer graphics and vision applications, including volumetric rendering (Mildenhall et al., 2020), 3D reconstruction (Mescheder et al., 2019), signal processing (Du et al., 2021), and geometry processing (Yang et al., 2021). While existing works have mostly focused on static representations, this work aims to explore these neural representations for dynamic simulations where the vector fields evolve over time. In particular, we explore a fundamental class of physics simulation tasks governed by partial differential equations (PDEs) with both *spatial* and *temporal* dependence,

$$\begin{aligned} \boldsymbol{\mathcal{F}}(\boldsymbol{f}, \boldsymbol{\nabla} \boldsymbol{f}, \boldsymbol{\nabla}^2 \boldsymbol{f}, \ldots, \dot{\boldsymbol{f}}, \ddot{\boldsymbol{f}}, \ldots) &= \mathbf{0}, \\ \boldsymbol{f}(\boldsymbol{x}, t) : \Omega \times \mathcal{T} &\to \mathbb{R}^d , \end{aligned} \tag{1}$$

where $\Omega \in \mathbb{R}^m$ and $\mathcal{T} \in \mathbb{R}$ are the spatial and temporal domains, respectively. Examples include the Navier-Stokes equations for fluid dynamics and the elastodynamic equation for solid mechanics.

To computationally simulate these problems, classical methods introduce both *spatial* and *temporal* discretizations. On the one hand, *temporal* discretization breaks down the entire temporal range into a finite number of time steps $\{t_n\}_{n=0}^{T}$, where $T$ is the number of temporal discretization samples, and $\Delta t = t_{n+1} - t_n$ is the time step size. The solution to Equation (1) then becomes a list of spatially-dependent vector fields: $\{\boldsymbol{f}^n(\boldsymbol{x})\}_{n=0}^{T}$. Classical methods then sequentially integrate from one time step $(n)$ to the next $(n+1)$, using a wide range of time integrators, such as explicit/implicit Euler (Ascher & Petzold, 1998). On the other hand, *spatial* discretization represents these spatially-dependent vector fields $\boldsymbol{f}^n(\boldsymbol{x})$ using grids, meshes, or point clouds (meshless particles). For example, the grid-based linear finite element method (FEM) (Hughes, 2012) defines a shape function $N^i$ on each grid node and represents the spatially depen-

[*]Equal contribution [1]Department of Computer Science, Columbia University, New York, USA [2]Department of Computer Science, University of Toronto, Toronto, Canada [3]Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, USA. Correspondence to: Honglin Chen <honglin.chen@columbia.edu>, Rundi Wu <rundi.wu@columbia.edu>.

*Proceedings of the $40^{th}$ International Conference on Machine Learning*, Honolulu, Hawaii, USA. PMLR 202, 2023. 

[1]Project webpage: http://www.cs.columbia.edu/cg/INSR-PDE/

dent vector field as $\boldsymbol{f}^n(\boldsymbol{x}) = \sum_{i=1}^{P} \boldsymbol{f}_i^n N^i$, where $P$ is the number of spatial samples.

While widely adopted in scientific computing applications, these classic *spatial* discretizations are not without drawbacks:

1. Spatial discretization errors abound in fluid simulations as artificial numerical diffusion (Lantz, 1971), dissipation (Fedkiw et al., 2001), and viscosity (Roache, 1998). These errors also appear in solid simulations as inaccurate collision resolution (Müller et al., 2015) and numerical fractures (Sadeghirad et al., 2011).
2. Memory usage spikes with the number of spatial samples $P$ (Museth, 2013).
3. Adaptive meshing (Narain et al., 2012) and data structures (Setaluri et al., 2014) can reduce memory footprints but are often computationally expensive and challenging to implement.

In this work, we ask: *what are the (dis)advantages of replacing a classical numerical method's spatial discretization with INSR while keeping intact the time integrator?* Unlike traditional representations that *explicitly* discretize the spatial vector via spatial primitives (e.g., points), INSRs *implicitly* encode the field through neural network weights. In other words, the field is parameterized by a neural network (typically multilayer perceptrons), i.e., $\boldsymbol{f}^n(\boldsymbol{x}) = \boldsymbol{f}_{\theta^n}(\boldsymbol{x})$ with $\theta^n$ being the network weights. As such, the memory usage for storing the spatial field is independent of the number of spatial samples, but rather it is determined by the number of neural network weights. We show that under the same memory constraint, INSRs indeed achieve higher accuracy than traditional discrete representations. Furthermore, INSRs are adaptive by construction (Xie et al., 2021), allocating the network weights to resolve field details at *any* spatial location without changing the network architecture.

We emphasize that our contribution is orthogonal to the choice of *temporal* discretization. Notably, our INSR approach works with various classic time integrators, including explicit/implicit/midpoint Euler, variational time integrators (Kane et al., 2000b), and even operator splitting schemes (Chorin, 1968). Indeed, our focus contrasts with other neural approaches, e.g., physics-informed neural network (PINN) (Raissi et al., 2019). Whereas prior work has focused on overcoming low data availability, efficiently solving inverse problems, or addressing high-dimensional problems, our unique focus is on exploring incorporating various existing classical time integrators. This ability to leverage classic time integrators is particularly effective in highly nonlinear problems, e.g., turbulence, where previous neural-PDE approaches struggle, e.g., PINN.

In summary, we make the following contributions:

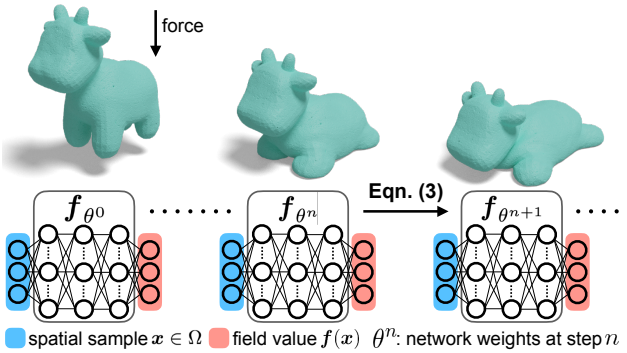


*Figure 1.* **Evolve neural field over time.** We represent the field of interest using a neural network $\boldsymbol{f}_{\theta^n}$, whose weights $\theta^n$ are updated at each time step via optimization-based time integration (Equation (3)). In this case, the spatial domain $\Omega$ is the volume encompassed by the undeformed object, and the represented field $\boldsymbol{f}$ is the deformation map. The governing PDE is the elastodynamic equation.

- We present INSR as an alternative spatial representation for various time-dependent physics simulation problems.
- Compared to the classic grid, mesh, point cloud (meshless) spatial representations, our INSR approach trades wall-clock runtime in favor of three benefits: lower discretization error, lower memory usage, and built-in adaptivity.
- Utilizing various classic time integrators, including variational time integrators and operator splitting schemes, INSR-based simulations capture challenging cases in contact mechanics and turbulent flows where previous neural-physics approaches fail.

## 2. Related Works

**Implicit Neural Spatial Representation** (INSR) uses neural networks to parameterize spatially-dependent functions (Chen & Zhang, 2019; Mescheder et al., 2019; Mildenhall et al., 2020; Dupont et al., 2022; Chen et al., 2021a; Pan et al., 2022; Chen et al., 2022), e.g., a signed-distance-field, where the input is an arbitrary spatial location and the output is its distance to the surface (Park et al., 2019). The nonlinear neural network's enormous expressivity makes INSR more accurate than its classic mesh-based and meshless counterparts under the same memory constraint. For example, with the same number of neural network weights as the number of mesh vertices (or meshless particles), INSR-SDF captures more geometric details than a triangle mesh (Takikawa et al., 2021). Indeed, memory consumption of traditional representations scales poorly with spatial resolutions. Adaptive discretizations can reduce memory, but their generations are expensive. By contrast, neural representations are adaptive by construction and can use their representation capacities at arbitrary locations of interest without memory increases or data structure alterations (Xie

et al., 2021). Because of the above-mentioned advantages, researchers have used INSR for many other applications such as image processing (Chen et al., 2021b; Shaham et al., 2021), 3D reconstruction (Wang et al., 2021a; Yariv et al., 2020), generative modeling (Schwarz et al., 2020; Wu & Zheng, 2022; Chan et al., 2022) and geometry processing (Yang et al., 2021; Sharp & Jacobson, 2022). Besides, time-dependent implicit representations have also been explored for capturing scene dynamics (Park et al., 2021a;b).

Regarding PDE applications, INSR has been successively deployed in strictly spatially dependent PDEs, including elastostatics (Zehnder et al., 2021), elliptic PDEs (Chiaramonte et al., 2013), and geometry processing (Yang et al., 2021). For time-dependent PDEs, Mehta et al. (2022) propose a framework for evolving INSR weights over time. However, their approach specializes in level sets. Du & Zaki (2021); Bruna et al. (2022) also evolve INSR's network weights over time, with the goal of removing PINN's limited time range as well as solving high-dimensional problems where meshing is impossible. By contrast, our work focuses on low-dimensional problems that heavily rely on classical FEM methods and explore the INSR solver as a more accurate, memory-efficient, and adaptive alternative.

**Machine Learning (ML) for PDEs** One line of ML-PDE works train on data from classical solvers (or real experiments), e.g., graph neural network (GNN) approaches (Sanchez-Gonzalez et al., 2020; Pfaff et al., 2020), neural operator approaches (Li et al., 2020b;c), and DeepONet (Lu et al., 2019). After training, these methods solve a new problem faster than the solver on which it was trained. However, these methods typically do not generalize to arbitrary initial/boundary conditions, material parameters, or geometries that are drastically different from the ones presented in the training data (Wang & Perdikaris, 2021). Another line of ML-PDE works does not require training data from classical solvers at all since these methods are *solvers* themselves, i.e., given the PDE and initial/boundary conditions, these methods can directly solve the PDE just like the classical solvers. These methods usually employ a physics-informed loss term (e.g., $\|\boldsymbol{\mathcal{F}}\|^2$) that incorporates the governing-PDE (Raissi et al., 2019; Wandel et al., 2020; Wang et al., 2021b). Our work also belongs to this "solver-type".

Previous physics-informed approaches generally treat the temporal domain as a continuous variable, and the PDE loss term would incorporate the entire spatiotemporal-dependent PDE ($\boldsymbol{\mathcal{F}}$). Krishnapriyan et al. (2021) break down the temporal domain into several subdomains and obtain better long-term temporal integration. However, the time variable is still treated as a continuous variable within each subdomain. By contrast, our approach discretizes the temporal domain just like the classical solvers. While this explicit temporal discretization can address the long-term prediction limitation of PINN, it is not the primary goal of this work. Instead, we show that by tapping into the rich literature of classical time integration schemes, we can model challenging problems in contact mechanics and turbulent flows where previous neural-PDE approaches struggle. Raissi et al. (2019); Wessels et al. (2020) also explore a time-discrete approach, but their methods specialize in Runge-Kutta schemes, while our general formulation supports a wide range of classical time integrators, including operator-splitting schemes.

## 3. Method: Time Integration on Neural Spatial Representations

Our goal is to solve time-dependent PDEs on neural-network-based spatial representations. In Section 3.1, we first discuss representing spatial vector fields with neural networks. Afterward, we will describe how to time-step the network weights with classic time integrators.

### 3.1. Neural Networks as Spatial Representations

We parameterize each time-discretized spatial vector field with a neural network: $\boldsymbol{f}^n = \boldsymbol{f}_{\theta^n}$, where $\theta^n$ are the neural network weights at time $t_n$. Specifically, the field quantity at an arbitrary spatial location $\boldsymbol{x} \in \Omega$ can be queried via network inference $\boldsymbol{f}_{\theta^n}(\boldsymbol{x})$.

Traditional representations *explicitly* discretize the spatial vector field using primitives such as meshes. These primitives *explicitly* correspond to spatial locations due to their compactly supported basis functions (Hughes, 2012). By contrast, INSRs *implicitly* encode the vector field via neural network weights, and each weight affects the vector field globally. Such global support is also an attribute of spectral methods (Canuto et al., 2007a;b). Compared to spectral methods, our approach does not need to know the required complexity ahead of time in order to determine the ideal basis functions (Xie et al., 2021). INSR automatically optimizes its parameters to where field detail is present.

Whereas memory consumption of *explicit* representations scales poorly with the number of spatial samples, memory consumption for INSR is independent of the number of spatial samples. Instead, memory usage (for storing the vector field) is determined by the number of network weights.

**Network Architecture** Following the INSR literature, we adopt SIREN (Sitzmann et al., 2020) (MLP with sinusoidal activation) as our network architecture for its accuracy and quick convergence speed advantages. Each MLP has a total of $\alpha$ hidden layers, each layer of width $\beta$. The specific choice of these hyper-parameters is described in Section 4.

**Spatial Gradients** Classic spatial representations compute spatial gradients via basis functions. Higher-order gradients require higher-order basis functions. By contrast, INSR is $C^\infty$ by construction. We evaluate the gradients via computation-graph-based auto-differentiation with respect

to the input (not the weights).

### 3.2. Time integration

Given previous-time spatial vector fields $\{\boldsymbol{f}^n(\boldsymbol{x})\}_{k=0}^n$, we can compute the next time-step ($t_{n+1}$) by solving an optimization problem:

$$\boldsymbol{f}^{n+1} = \underset{\boldsymbol{f}^{n+1}}{\operatorname{argmin}} \sum_{\boldsymbol{x}\in\mathcal{M}\subset\Omega} \boldsymbol{\mathcal{I}}(\Delta t, \{\boldsymbol{f}^k(\boldsymbol{x})\}_{k=0}^{n+1}, \{\nabla \boldsymbol{f}^k(\boldsymbol{x})\}_{k=0}^{n+1}, \{\nabla^2 \boldsymbol{f}^k(\boldsymbol{x})\}_{k=0}^{n+1}, \ldots) \,. \tag{2}$$

For example, the classic explicit/implicit Euler methods can be formulated in this form (Kharevych et al., 2006) as well as variational time integrators derived from Hamilton's principle (Kane et al., 2000b). Operator-splitting style integrators (Chorin, 1968) also weave seamlessly into this formulation by solving multiple optimization problems. Note that the particular choice of the objective function $\boldsymbol{\mathcal{I}}$ depends on the PDE of interest and the time integrator choice.

This optimization formulation applies to *any* spatial representation. It has been explored thoroughly for classic spatial discretizations (Batty et al., 2007; Bouaziz et al., 2014; Gast et al., 2015), which is defined over a finite number of the spatial integration samples $\mathcal{M} := \{\boldsymbol{x}^j \in \Omega \mid 1 \leq j \leq |\mathcal{M}|\}$, e.g., grids.

Applying this formulation to a neural spatial representation, we optimize for

$$\theta^{n+1} = \underset{\theta^{n+1}}{\operatorname{argmin}} \sum_{\boldsymbol{x}\in\mathcal{M}\subset\Omega} \boldsymbol{\mathcal{I}}(\Delta t, \{\boldsymbol{f}_{\theta^k}(\boldsymbol{x})\}_{k=0}^{n+1}, \{\nabla \boldsymbol{f}_{\theta^k}(\boldsymbol{x})\}_{k=0}^{n+1}, \{\nabla^2 \boldsymbol{f}_{\theta^k}(\boldsymbol{x})\}_{k=0}^{n+1}, \ldots) \tag{3}$$

where $\{\theta^k\}_{k=0}^n$ are the (fixed, not variable) neural network weights from previous time steps. Figure 1 illustrates our time integration process, and Algorithm 1 provides the corresponding pseudocode. In all the examples presented in this work, we solve this time-integration optimization problem via Adam (Kingma & Ba, 2014), a first-order stochastic gradient descent method.

**Spatial Sampling** *Explicit* spatial representations (e.g., tetrahedra mesh) are often tied to a particular spatial sampling; remeshing is sometimes possible but can also have drawbacks, especially in higher dimensions (Alliez et al., 2002; Narain et al., 2012). By contrast, *implicit* spatial representations allow for arbitrary spatial sampling by construction (Equation (3)). Following Sitzmann et al. (2020), we dynamically sample $\mathcal{M}$ during optimization. For every gradient descent iteration in every time step, we use a stochastic sample set $\mathcal{M}$ from the spatial domain $\Omega$; $\mathcal{M}$ corresponds to the "mini-batch" in stochastic gradient descent, with batch size $|\mathcal{M}|$.

**Algorithm 1** Time integration

**Input:** initial network weights $\theta^0$, timestep size $\Delta t$, number of timesteps $N$, time integrator $\boldsymbol{\mathcal{I}}$, spatial domain $\Omega$

```
n ← 0
while n < N do
  θ^{n+1} ← θ^n
  while not converged do
    randomly sample M ⊂ Ω
    L_{θ^{n+1}} = Σ_{x∈M} I(Δt, {f_{θ^k}(x)}_{k=0}^{n+1}, {∇f_{θ^k}(x)}_{k=0}^{n+1}, ...)
    θ^{n+1} ← θ^{n+1} − α∇L_{θ^{n+1}}
    n ← n + 1
  end while
end while
```

**Boundary Condition** PDEs are typically accompanied by spatial (e.g., Dirichlet or Neumann) boundary conditions, which we formulate as additional penalty terms in the objective Equation (3),

$$\theta^{n+1} = \underset{\theta^{n+1}}{\operatorname{argmin}} \sum_{\boldsymbol{x}\in\mathcal{M}\subset\Omega} \boldsymbol{\mathcal{I}}(\Delta t, \{\boldsymbol{f}_{\theta^k}(\boldsymbol{x})\}_{k=0}^{n+1}, \{\nabla \boldsymbol{f}_{\theta^k}(\boldsymbol{x})\}_{k=0}^{n+1}, \ldots) + \lambda \sum_{\boldsymbol{x}^b\in\mathcal{M}^b\subset\partial\Omega} \boldsymbol{\mathcal{C}}(\boldsymbol{f}_{\theta^{n+1}}(\boldsymbol{x}^b), \nabla \boldsymbol{f}_{\theta^{n+1}}(\boldsymbol{x}^b), \ldots) \,, \tag{4}$$

where $\lambda$ is the weighting factor and $\partial\Omega$ is the boundary of the spatial domain. The particular choice of the boundary constraint function $\boldsymbol{\mathcal{C}}$ depends on the problem of interest.

**Initial Condition** The neural network is initialized using the given initial condition, i.e., the field value at time $t = 0$, by optimizing

$$\theta^0 = \underset{\theta^0}{\operatorname{argmin}} \sum_{\boldsymbol{x}\in\mathcal{M}\subset\Omega} ||\boldsymbol{f}_{\theta^0}(\boldsymbol{x}) - \hat{\boldsymbol{f}}^0(\boldsymbol{x})||_2^2 \,, \tag{5}$$

where $\hat{\boldsymbol{f}}^0$ is the given initial condition. Similar to Equation (3), we solve this optimization problem using Adam (Kingma & Ba, 2014) and stochastically sample $\mathcal{M}$ at each gradient descent iteration.

## 4. Experiments

In this section, we evaluate our method on three classic time-dependent PDEs: the advection equation, the incompressible Euler equations, and the elastodynamic equation.

**Baselines** From classical solvers, we compare with three baselines: (1) the grid-based finite difference method (Fedkiw et al., 2001), (2) the tetrahedral-mesh-based finite element method (Hughes, 2012), (3) the meshless-particle-based material point method (Jiang et al., 2016). From neural-network-based, physics-informed approaches, we compare with another three baselines: (4) the original PINN (Raissi et al., 2019), (5) PINN with temporal sub-domains (Krishnapriyan et al., 2021), (6) physics-informed DeepONet (Wang et al., 2021b; Wang & Perdikaris, 2021). We fo-

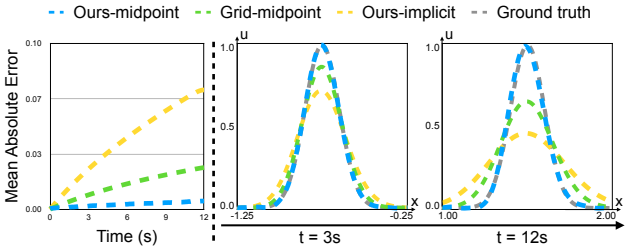


*Figure 2.* **1D advection example:** A Gaussian-shaped wave initially centered at $x = -1.5$ moves rightward with a constant velocity of 0.25. From left to right, we show the mean absolute error plot over time and solutions at $t = 3s$ and $t = 12s$. Error is computed using 500 uniform spatial points. Using an energy-preserving midpoint time integrator, our solution (blue) well approximates the ground truth (grey) over time, while the grid-based finite difference method (green) tends to diffuse over time.

*Table 1.* **Quantitative results for the 1D advection example (Figure 2).** Error: mean absolute error over a total of 240 time steps, compared to the ground truth analytical solution. Error is evaluated over 500 uniform spatial samples. Time: runtime for a total of 240 time steps. Memory: memory usage for storing the spatial representations.

| Methods | Error | Time | Memory |
|---|---|---|---|
| Ours | **0.0030** | 5.33h | **3.520KB** |
| Grid (same memory) | 0.0146 | 1.13s | 3.520KB |
| Grid (same error) | 0.0029 | 1.80s | 27.35KB |

cus on comparing these physics-informed neural approaches because, like our method, they do not require any training data from the classic solvers.

**Comparison** To strike an apple-to-apple comparison, we use the same time integrator and contact model for our approach and all the classical baselines. The only difference is the spatial representation. Notably, one can also use more advanced time integrators and contact models than the ones used in this work. Nevertheless, since the baselines and our approach adopt the same time integrators, our advantages on spatial discretization remain. For both classic and neural baselines, we ensure that they use the same amount of memory for storing the spatial representation as our method, e.g., grid size and the number of network layers.

We refer readers to Appendices A and B for other implementation details (e.g., initial / boundary conditions, baseline setups) and additional results. The temporal evolutions of the PDEs are best illustrated by the **supplementary video**.

### 4.1. Advection Equation

Consider the classic 1D advection equation,

$$\frac{\partial u}{\partial t} + (a \cdot \nabla)u = 0\,, \tag{6}$$

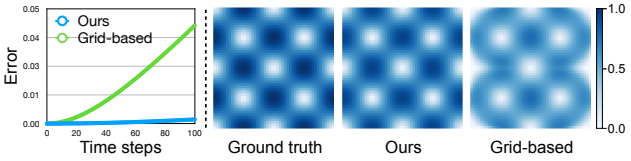


*Figure 3.* **2D Taylor-Green vortex simulation.** Left: the mean squared error of the velocity field for 100 timesteps. Error is computed using $48^2$ uniform spatial points. Right: velocity magnitude of solutions from the ground truth, ours, and the grid-based method (grid size 48) at timestep $n = 100$. Under the same memory usage (for storing the spatial representation), our solution has a significantly smaller error than the grid-based method.

*Table 2.* **Quantitative results for the 2D Taylor-Green fluid example (Figure 3).** Error: mean squared error of velocity field over total 100 time steps, compared to the ground truth analytical solution. Time: runtime for a total of 100 time steps. Memory: memory usage for storing the spatial representations.

| Methods | Error | Time | Memory |
|---|---|---|---|
| Ours | **3.35e-4** | 14.02h | **25.887KB** |
| Grid (same memory) | 4.83e-3 | 2.91s | 27.00KB |
| Grid (same error) | 3.24e-4 | 189.4s | 12.00MB |

where $a$ is the advection velocity, and the vector field of interest is the advected quantity $\boldsymbol{f} = u$.

**Time Integration** We adopt the same time integration scheme in both the discrete grid representation and ours. Choosing the energy-preserving midpoint method (Mullen et al., 2009) yields the time integration operator,

$$\boldsymbol{\mathcal{I}} = \| \frac{u^{n+1}(\boldsymbol{x}) - u^n(\boldsymbol{x})}{\Delta t} + (a \cdot \nabla)(\frac{u^{n+1}(\boldsymbol{x}) + u^n(\boldsymbol{x})}{2}) \|_2^2 \,. \tag{7}$$

**Results** Figure 2 shows an example where a Gaussian-shaped wave moves with constant velocity $a = 0.25$. Under the same memory usage for storing the spatial representations, our approach uses $\alpha = 2$ hidden layers of width $\beta = 20$, and the finite difference grid resolution is 901.

Using midpoint time integrator, the grid-based method (`grid-midpoint`) diffuses over time due to its spatial discretization, which is a well-known numerical issue (Courant et al., 1952; Selle et al., 2008). On the contrary, our result (`ours-midpoint`) does not suffer from numerical dissipation and agrees well with the ground truth at all frames. We also tried the implicit Euler time integrator (see `ours-implicit`) and found it inherits its property of energy dissipation. Choosing the midpoint time integrator helps us preserve energy and obtain high-accuracy results. In Table 1, we report the quantitative evaluation result for our 1D advection example in Figure 2.

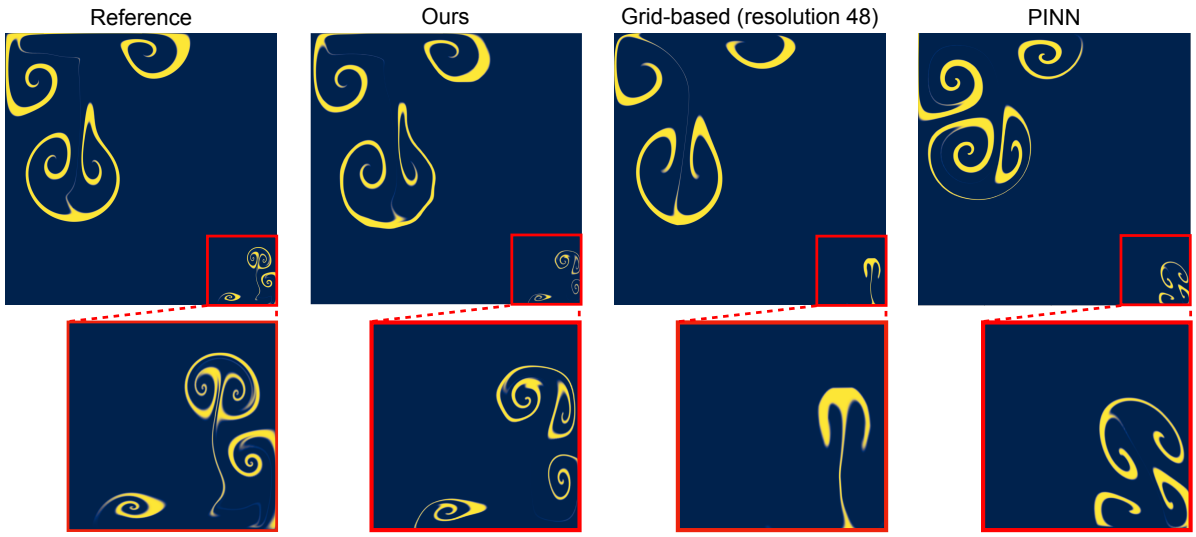


*Figure 4.* **Two vortices of different scales.** We show the advected density field after 2.5 seconds from the reference (top-left), our method (top-right), the grid-based method of resolution 48 (bottom-left), and PINN (bottom-right). The reference is obtained using the high-resolution grid-based method (we use resolution 1024) and serves as a good approximation of the ground truth. Our MLP ($\alpha = 3, \beta = 32$) has the same memory footprint as grids of resolution 48. PINN uses the same MLP network as ours. Under the same memory constraint, our approach suffers less dissipation, captures more vorticity, and best resembles the reference solution, whose grids take $\sim 450\times$ memory compared to our network. See Figure 10 for the initial condition of this example.

*Table 3.* **Quantitative results for the two-vortices fluid example in Figure 4.** Error: average absolute error of kinetic energy over a total of 50 timesteps, compared to the reference solution. Kinetic energy is computed using $1024^2$ uniform samples. Time: runtime for a total of 50 timesteps. Memory: memory usage for storing spatial representations. `Ours` and `Grid-based` (Stam, 1999) use the operator splitting scheme; `Ours-residual`, `PINN` (Raissi et al., 2019), `PINN-sub` (Krishnapriyan et al., 2021) and `piDeepONet` (Wang et al., 2021b) use the residual of the Euler equation as the objectives (see Equation (23) and Equation (24)).

| Methods | Error | Time | Memory |
|---|---|---|---|
| Ours | **2.24e2** | 10.81h | 25.887KB |
| Ours-residual | 2.97e4 | 10.07h | 25.887KB |
| Grid-based | 1.07e4 | 1.78s | 27.00KB |
| PINN | 2.25e4 | 7.88h | 26.137KB |
| PINN-sub | 3.21e4 | 20.83h | 26.137KB |
| piDeepONet | 2.11e4 | 9.42h | 65.855KB |

### 4.2. Incompressible Euler Equations

In the incompressible Euler Equations

$$\rho_f(\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u}\cdot\nabla\boldsymbol{u}) = -\nabla p + \rho_f \boldsymbol{g}, \qquad \nabla\cdot\boldsymbol{u} = 0, \tag{8}$$

the vector field of interest is the fluid velocity field $\boldsymbol{f} = \boldsymbol{u}$; $p$ is the pressure, $\boldsymbol{g}$ is the external force, and $\rho_f$ is the fluid density. In our experiments, we consider $\rho_f = 1$ and $\boldsymbol{g} = 0$. The pressure field $p$ is represented with another MLP network.

**Time Integration** We apply the Chorin-style operator splitting scheme (Chorin, 1968; Stam, 1999) to both the neural spatial and finite-difference grid representations. This scheme converts the highly nonlinear PDE into three linear PDEs, which significantly eases the challenge of solving. The entire scheme breaks down into three sequential steps: advection (adv), pressure projection (pro), and velocity correction (cor).

**Advection** uses a semi-Lagrangian method, encoded by the operator (Staniforth & Côté, 1991)

$$\boldsymbol{\mathcal{I}}_{adv} = \|{\boldsymbol{u}_{adv}}^{n+1}(\boldsymbol{x}) - \boldsymbol{u}^n(\boldsymbol{x}_{backtrack})\|_2^2\,, \tag{9}$$

whose optimization yields the advected velocity ${\boldsymbol{u}_{adv}}^{n+1}$. The backtracked location is given by $\boldsymbol{x}_{backtrack} = \boldsymbol{x} - \Delta t \boldsymbol{u}^n(\boldsymbol{x})$. While traditional discrete representations compute the backtracked velocity using interpolation (e.g., linear basis function), our approach *requires no interpolation*, only direct evaluation via network inference at $\boldsymbol{x}_{backtrack}$.

**Pressure projection** is encapsulated by the operator

$$\boldsymbol{\mathcal{I}}_{pro} = \|\nabla^2 p^{n+1}(\boldsymbol{x}) - \nabla\cdot{\boldsymbol{u}_{adv}}^{n+1}(\boldsymbol{x})\|_2^2. \tag{10}$$

Plugging $\boldsymbol{\mathcal{I}}_{pro}$ into the optimization solver, we obtain the pressure $p^{n+1}$ that enforces incompressibility. Note that the MLP that represents the velocity field $\boldsymbol{u}_{adv}$ is kept fixed in this step.

**Velocity correction** is formulated by the operator

$$\boldsymbol{\mathcal{I}}_{cor} = \|\boldsymbol{u}^{n+1} - ({\boldsymbol{u}_{adv}}^{n+1}(\boldsymbol{x}) - \nabla p^{n+1}(\boldsymbol{x}))\|_2^2\,, \tag{11}$$

which adds the pressure gradient to the advected velocity yielding the *incompressible* velocity $\boldsymbol{u}^{n+1}$.

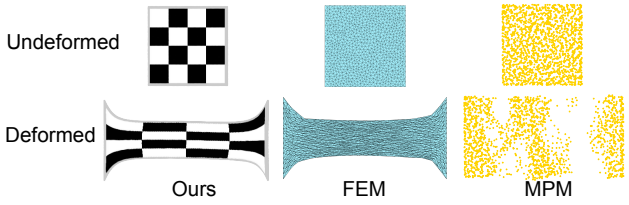


*Figure 5.* **Elastic tension test**. We use $\alpha = 3$ hidden layers of width $\beta = 68$ for our MLP, which takes the same memory as the FEM mesh (0.8K vertices, 1.5K faces) and MPM point cloud (1.7K points).

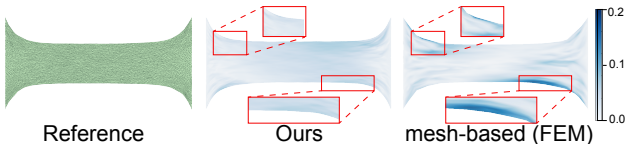


*Figure 6.* **Error of the elastic tension test.** We visualize the L2 distance error of our result and the FEM result (0.8K vertices). Errors are calculated against the reference result obtained by high-resolution FEM. Our result obtains smaller errors.

**Results** We first test our method on the 2D Taylor-Green vortex with zero viscosity (Taylor & Green, 1937; Brachet et al., 1983). The closed-form analytical solution is given by: $\boldsymbol{u}(\boldsymbol{x}, t) = (\sin x \cos y, -\cos x \sin y)$ for $\boldsymbol{x} \in [0, 2\pi] \times [0, 2\pi]$. To compare under the same memory usage (for storing the velocity field), we use $\alpha = 3$ hidden layers of width $\beta = 32$ for our MLP and set the grid size to 48 for the grid-based projection method (Stam, 1999). We set $\Delta t = 0.05$ and execute both methods for 100 timesteps. In Figure 3, we show the mean squared error of the solved velocity field over time. In Table 2, we report the quantitative evaluation result for our 2D Taylor-Green example in Figure 3. This example demonstrates that our method excellently preserves a stationary solution and obtains a much smaller error than the grid-based method.

For discrete grid representation, efficiently capturing multi-scale details usually requires difficult-to-implement adaptive data structures (Setaluri et al., 2014). Instead, INSRs are adaptive by construction (Xie et al., 2021) and enable us to capture more details under the same memory storage. We set up an example where the initial velocity field is composed of two Taylor-Green vortices of different scales (see Figure 10 for illustration).

In Figure 4 and Table 3, we show our results on this example and compare with the grid-based projection method (Stam, 1999), PINN (Raissi et al., 2019), PINN with temporal sub-domains (PINN-sub) (Krishnapriyan et al., 2021) and pi-DeepONet (Wang et al., 2021b). All methods are compared under the same memory storage for spatial representations. For PINN and PINN-sub, we use the same MLP structure as ours. Detailed setups for these baselines can be found in Appendix A.3. We execute our approach, the grid-based method, and PINN-sub for 50 timesteps with $\Delta t = 0.05$,

*Table 4.* **Quantitative results for the 2D elasticity tension example in Figure 6.** Error: infinity norm of L2 distance w.r.t. the high-resolution ground truth. Time: total runtime until convergence. Memory: memory usage for storing spatial representations.

| Methods | Error | Time | Memory |
|---|---|---|---|
| Ours | **8.82e-2** | 38.33m | 56.32KB |
| Mesh-based | 1.99e-1 | 22.04s | 54.00KB |

and train PINN and piDeepONet with the same temporal range of 2.5 seconds. Our approach can capture the fine details of the smaller vortex, best approximate the reference solution, and has the most negligible energy dissipation.

Moreover, the superior accuracy of our approach also comes from the usage of the operator-splitting time integration scheme. While PINN, PINN-sub, and piDeepONet also use implicit neural representations like ours, they treat time as a continuous variable (i.e., part of the network inputs). Therefore, they cannot use the operator-splitting scheme but only employ the residual of the Euler equations as the training objective, which is more challenging to optimize. The time-discrete PINN proposed by Raissi et al. (2019) specializes in Runge-Kutta schemes and does not support operator splitting in its current form either. We verify these observations by changing the objectives of our method to a similar training objective used by them, i.e., the residual of the Euler equation (Equation (23)) with a fully-implicit-time-discretization (Equation (24)). After such change, we obtain a significantly worse result (see `ours-residual` in Table 3), which confirms the necessity of using the operator-splitting scheme. This experiment demonstrates that just replacing PINN/SIREN's time-dependent formulation is insufficient, but the particular choice of temporal discretization matters.

### 4.3. Elastodynamic Equation

Lastly, we study the Elastodynamic equation

$$\rho_0 \ddot{\boldsymbol{\phi}} = \nabla \cdot \boldsymbol{P}(\boldsymbol{F}) + \rho_0 \boldsymbol{b} \tag{12}$$

that describe the motions of deformable solids (Gonzalez & Stuart, 2008). The vector field of interest is the deformation map $\boldsymbol{f} = \boldsymbol{\phi}$. Here $\rho_0$ is the density in the reference space, $\boldsymbol{P}$ is the first Piola-Kirchhoff stress, $\boldsymbol{F} = \nabla \boldsymbol{\phi}$ is the deformation gradient, $\dot{\boldsymbol{\phi}}$ and $\ddot{\boldsymbol{\phi}}$ are the velocity and acceleration, and $\boldsymbol{b}$ is the body force.

We assume a hyper-elasticity constitutive law, i.e., $\boldsymbol{P} = \frac{\partial \Psi}{\partial \boldsymbol{F}}$, where $\Psi$ is the energy density function. In particular, we assume a variant of the stable Neo-Hookean energy (Smith et al., 2018)

$$\Psi = \frac{\lambda}{2} \mathrm{tr}^2(\Sigma - \boldsymbol{I}) + \mu(\det(\boldsymbol{F}) - 1)^2, \tag{13}$$

where $\lambda$ and $\mu$ are the first and second lame parameters, $\Sigma$ are the singular values of the deformation gradient $\boldsymbol{F}$, and

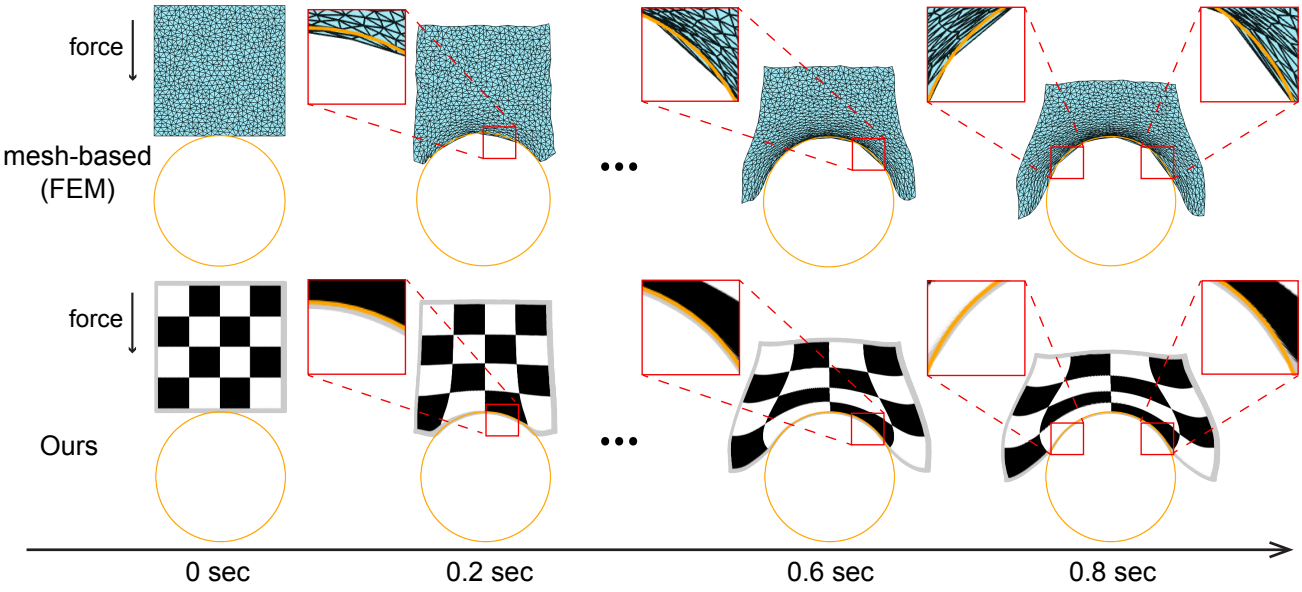


*Figure 7.* **An elastic square collides with a circle.** Under the same memory footprint (for storing the spatial representation), the result of mesh-based FEM (top) conforms poorly at the collision interface. In contrast, our result (bottom) fits the boundary more gracefully.

*Table 5.* **Quantitative results for the collision example in Figure 7.** Error: maximum overlapping distance between the square and the circle. Time: runtime for a total of 10 timesteps. Memory: memory usage for storing spatial representations.

| Methods | Error 0.2s | Error 0.6s | Error 0.8s | Time | Memory |
|---|---|---|---|---|---|
| Ours | 1.62e-2 | 1.04e-2 | 1.06e-2 | 23.0m | 56.32KB |
| Mesh-based | 3.45e-2 | 5.47e-2 | 4.23e-2 | 98.2s | 54.00KB |

$\det(\boldsymbol{F})$ is the determinant of the deformation gradient $\boldsymbol{F}$. When $\mu = 0$, the elastic energy recovers the As-Rigid-As-Possible energy (Sorkine & Alexa, 2007).

**Time Integration** We apply the variational time integration scheme (Gast et al., 2015; Kane et al., 2000a) to the (1) tetrahedral finite element method, (2) the material point method, and (3) our neural representation,

$$\mathcal{I} = \underbrace{\frac{1}{2}\rho_0(\dot{\boldsymbol{\phi}}^{n+1} - \dot{\boldsymbol{\phi}}^{n})^T(\dot{\boldsymbol{\phi}}^{n+1} - \dot{\boldsymbol{\phi}}^{n})}_{\text{kinetic energy}} + \underbrace{\Psi(\boldsymbol{\phi}^{n+1})}_{\substack{\text{elastic}\\ \text{energy}}} \underbrace{- \rho_0 \boldsymbol{b}^T \boldsymbol{\phi}^{n+1}}_{\substack{\text{external force}\\ \text{potential}}}, \tag{14}$$

where $\dot{\boldsymbol{\phi}}^{n+1} = (\boldsymbol{\phi}^{n+1} - \boldsymbol{\phi}^{n})/\Delta t$, $\rho_0$ is the density, $\boldsymbol{b}$ is the external force. We can also incorporate boundary conditions, e.g., positional and contact constraints, by introducing additional energy terms (Bouaziz et al., 2014; Li et al., 2020a) (see Appendix A.4). These energy terms allow us to simulate challenging contact problems where the material impacts a collision surface at high speed.

**Results** We first evaluate our method on a typical 2D example for elastic tension test, and compare with the traditional finite element method (FEM, (Hughes, 2012; Reddy, 2019)) using tetrahedral mesh representation. We use $\alpha = 3$ hidden layers of width $\beta = 68$ for our MLP, which takes the same memory as the meshes used by FEM (0.8K vertices, 1.5K faces). The geometry is rendered as point clouds for our method in all our examples. As shown in Figure 6 and Table 4, our method obtains a more minor error than FEM. Classic mesh-less particle technique (material point method, MPM) suffers incorrect numerical fracture in this example (see Figure 5).

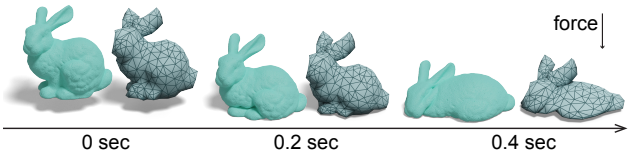


*Figure 8.* **A bunny collides with the ground in 3D.** Using INSR, our method (green, left) captures more intricate geometry details and complex dynamics compared to the traditional mesh-based FEM (blue, right) under the same memory usage.

By using INSR, our method can capture more intricate details than the traditional discrete representations under the same memory usage. In Figure 7, we show that our method allows the deformed square to gracefully fit the boundary of the sphere during the non-trivial collision. In contrast, the mesh-based FEM struggles to produce smooth results due to its insufficient mesh resolution. To alleviate such artifacts, the mesh-based FEM either needs to increase resolutions, thus inducing higher memory cost, or conducts complex remeshing (Narain et al., 2012). As shown in Figure 8 and Figure 13, our method allows for more complex dynamics and fine geometry details compared to the mesh-based FEM under the same memory footprint.

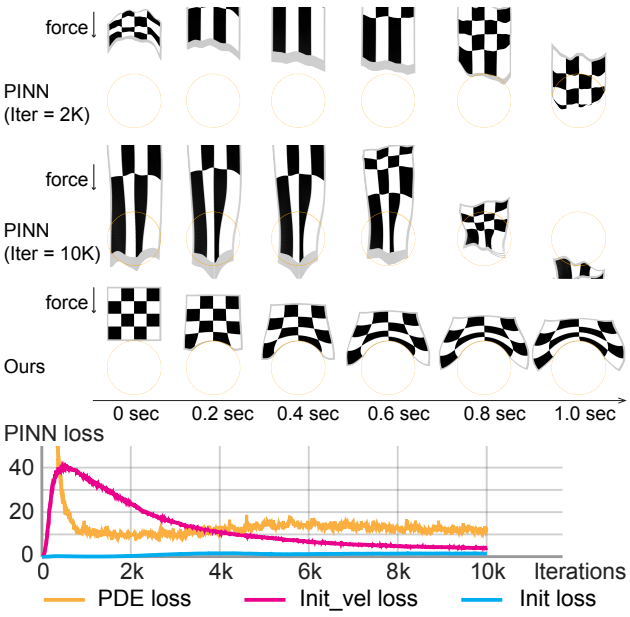


*Figure 9.* **PINN's result on the square-circle collision example in Figure 7.** Our approach successfully captures the correct contact behaviors, but PINN fails to do so under the same memory usage and network structure. To demonstrate this, we present PINN's failure results trained with both 2K and 10K ADAM iterations (top). Despite making progress in minimizing the loss (bottom) under various hyper-parameters, such as learning rate and collision ratio, PINN was unable to properly capture the highly discontinuous contact elasticity. Moreover, even with additional training iterations (i.e., more than 10K), the training loss did not improve.

Note that we adopt the same collision detection and handling strategy for both the neural representation and the mesh-based representation (FEM). Specifically, we use a spring-like penalty force and the corresponding energy to move the collided point out of its collision surface, similar to (McAdams et al., 2011; Xian et al., 2019). Since our approach and FEM share the same time integration scheme and the same collision handling method, the difference strictly stems from the underlying spatial representations.

Finally, in Figure 9, we compare our time-independent formulation to PINN's time-dependent approach on non-trivial collision cases. PINN struggles to capture the correct contact behavior when extreme nonlinearities and discontinuities involve. The most likely reason is that PINN models the time continuously, but collision is highly discontinuous in time. Therefore, the loss function of PINN is prone to abrupt change when the collision happens, and collision forces are involved. Our approach is able to adopt the variational time integrator and use the incremental potential as our loss function, which is known for its stability.

## 5. Discussion and Conclusion

This work explores INSR for numerically modeling time-dependent PDEs. Combined with a wide range of classical time integrators, the INSR solver captures various advection, elasticity, and fluid phenomena and outperforms previous physics-informed neural network approaches on multiscale, turbulent flows and contact mechanics problems. Compared to classic *explicit* representations (e.g., grids and meshes), our approach offers improved accuracy, reduced memory, and automatic adaptivity.

While offering important benefits, INSR-based PDE time-stepping requires longer wall-clock computation time than existing methods. (For reference, PINN also takes longer to solve forward problems than classic FEMs. See also Table 1 by Zehnder et al. (2021) and Section 7 by Yang et al. (2021).) Optimizing *globally-supported* neural network weights takes longer than optimizing *locally-supported* grid values, even if there are fewer neural network weights than the number of grid nodes. For instance, for the bunny example (Figure 8), our neural network optimization takes around 30 minutes per timestep while the corresponding FEM simulation takes less than 1 minute.

Facing this wall clock vs. memory/accuracy/adaptivity trade-off, we believe an exciting future direction is hybrid mesh-neural spatial representations that aim for the best of both worlds. Indeed, our work does not advocate INSR as the "perfect" spatial representation for solving PDEs. Instead, we view our work as a stepping stone toward future hybrid mesh-neural PDE solvers. For this matter, recent hybrid representation works (Müller et al., 2022; Takikawa et al., 2021; Martel et al., 2021) have offered promising results in reducing training time from hours to seconds while keeping the expressiveness of INSR. We hope our work will serve as a benchmark for future representations.

Our work demonstrates the effectiveness of INSR in solving time-dependent PDEs and observes empirical convergence under refinement (see Figure 12). Future work may consider a theoretical analysis of convergence and stability. More challenging physical phenomena, such as turbulence and intricate contacts, are also important future directions. Currently, our work enforces "soft" boundary conditions. Enforcing "hard" boundary conditions on a neural network is another exciting direction (Lu et al., 2021).

## Acknowledgements

This research was partially supported by National Science Foundation (1910839) and a Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant [RGPIN-2021-03733]. We thank Henrique Teles Maia for proofreading and all the anonymous reviewers for their helpful comments and suggestions. Lastly, we want to thank all our math professors (James Scott, Joseph Teran, among many others) who instilled in us a love for PDEs.

## References


Alliez, P., Meyer, M., and Desbrun, M. Interactive geometry remeshing. *ACM Transactions on Graphics (TOG)*, 21 (3):347–354, 2002.

Ascher, U. M. and Petzold, L. R. *Computer methods for ordinary differential equations and differential-algebraic equations*, volume 61. Siam, 1998.

Batty, C., Bertails, F., and Bridson, R. A fast variational framework for accurate solid-fluid coupling. *ACM Transactions on Graphics (TOG)*, 26(3):100–es, 2007.

Bouaziz, S., Martin, S., Liu, T., Kavan, L., and Pauly, M. Projective dynamics: Fusing constraint projections for fast simulation. *ACM transactions on graphics (TOG)*, 33(4):1–11, 2014.

Brachet, M. E., Meiron, D. I., Orszag, S. A., Nickel, B., Morf, R. H., and Frisch, U. Small-scale structure of the taylor–green vortex. *Journal of Fluid Mechanics*, 130: 411–452, 1983.

Bruna, J., Peherstorfer, B., and Vanden-Eijnden, E. Neural galerkin scheme with active learning for high-dimensional evolution equations. *arXiv preprint arXiv:2203.01360*, 2022.

Canuto, C., Hussaini, M. Y., Quarteroni, A., and Zang, T. A. *Spectral methods: fundamentals in single domains*. Springer Science & Business Media, 2007a.

Canuto, C., Hussaini, M. Y., Quarteroni, A., and Zang, T. A. *Spectral methods: evolution to complex geometries and applications to fluid dynamics*. Springer Science & Business Media, 2007b.

Chan, E. R., Lin, C. Z., Chan, M. A., Nagano, K., Pan, B., De Mello, S., Gallo, O., Guibas, L. J., Tremblay, J., Khamis, S., et al. Efficient geometry-aware 3d generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16123–16133, 2022.

Chen, P. Y., Chiaramonte, M., Grinspun, E., and Carlberg, K. Model reduction for the material point method via an implicit neural representation of the deformation map. *arXiv preprint arXiv:2109.12390*, 2021a.

Chen, P. Y., Xiang, J., Cho, D. H., Pershing, G., Maia, H. T., Chiaramonte, M., Carlberg, K., and Grinspun, E. CROM: Continuous reduced-order modeling of PDEs using implicit neural representations. *arXiv preprint arXiv:2206.02607*, 2022.

Chen, Y., Liu, S., and Wang, X. Learning continuous image representation with local implicit image function. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 8628–8638, 2021b.

Chen, Z. and Zhang, H. Learning implicit fields for generative shape modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5939–5948, 2019.

Chiaramonte, M., Kiener, M., et al. Solving differential equations using neural networks. *Machine Learning Project*, 1, 2013.

Chorin, A. J. Numerical solution of the navier-stokes equations. *Mathematics of computation*, 22(104):745–762, 1968.

Chuang, P.-Y. and Barba, L. A. Experience report of physics-informed neural networks in fluid simulations: pitfalls and frustration. *arXiv preprint arXiv:2205.14249*, 2022.

Courant, R., Isaacson, E., and Rees, M. On the solution of nonlinear hyperbolic differential equations by finite differences. *Communications on pure and applied mathematics*, 5(3):243–255, 1952.

Du, Y. and Zaki, T. A. Evolutional deep neural network. *Physical Review E*, 104(4):045303, 2021.

Du, Y., Collins, K., Tenenbaum, J., and Sitzmann, V. Learning signal-agnostic manifolds of neural fields. *Advances in Neural Information Processing Systems*, 34: 8320–8331, 2021.

Dupont, E., Kim, H., Eslami, S., Rezende, D., and Rosenbaum, D. From data to functa: Your data point is a function and you should treat it like one. *arXiv preprint arXiv:2201.12204*, 2022.

Fedkiw, R., Stam, J., and Jensen, H. W. Visual simulation of smoke. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pp. 15–22, 2001.

Gast, T. F., Schroeder, C., Stomakhin, A., Jiang, C., and Teran, J. M. Optimization integrator for large time steps. *IEEE transactions on visualization and computer graphics*, 21(10):1103–1115, 2015.

Gonzalez, O. and Stuart, A. M. *A first course in continuum mechanics*, volume 42. Cambridge University Press, 2008.

Hu, Y., Li, T.-M., Anderson, L., Ragan-Kelley, J., and Durand, F. Taichi: a language for high-performance computation on spatially sparse data structures. *ACM Transactions on Graphics (TOG)*, 38(6):1–16, 2019.

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

Jiang, C., Schroeder, C., Teran, J., Stomakhin, A., and Selle, A. The material point method for simulating continuum materials. In *ACM SIGGRAPH 2016 Courses*, SIGGRAPH '16. Association for Computing Machinery, 2016.

Kane, C., Marsden, J. E., Ortiz, M., and West, M. Variational integrators and the newmark algorithm for conservative and dissipative mechanical systems. *International Journal for Numerical Methods in Engineering*, 49(10): 1295–1325, 2000a.

Kane, C., Marsden, J. E., Ortiz, M., and West, M. Variational integrators and the newmark algorithm for conservative and dissipative mechanical systems. *International Journal for numerical methods in engineering*, 49(10): 1295–1325, 2000b.

Kharevych, L., Wei, W., Tong, Y., Kanso, E., Marsden, J. E., Schröder, P., and Desbrun, M. *Geometric, variational integrators for computer animation*. Eurographics Association, 2006.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Krishnapriyan, A., Gholami, A., Zhe, S., Kirby, R., and Mahoney, M. W. Characterizing possible failure modes in physics-informed neural networks. *Advances in Neural Information Processing Systems*, 34:26548–26560, 2021.

Lantz, R. Quantitative evaluation of numerical diffusion (truncation error). *Society of Petroleum Engineers Journal*, 11(03):315–320, 1971.

Levin, D. I. Bartels: A lightweight collection of routines for physics simulation, 2020. https://github.com/dilevin/Bartels.

Li, M., Ferguson, Z., Schneider, T., Langlois, T. R., Zorin, D., Panozzo, D., Jiang, C., and Kaufman, D. M. Incremental potential contact: intersection-and inversion-free, large-deformation dynamics. *ACM Trans. Graph.*, 39(4): 49, 2020a.

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020b.

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A. Neural operator: Graph kernel network for partial differential equations. *arXiv preprint arXiv:2003.03485*, 2020c.

Lu, L., Jin, P., and Karniadakis, G. E. Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint arXiv:1910.03193*, 2019.

Lu, L., Pestourie, R., Yao, W., Wang, Z., Verdugo, F., and Johnson, S. G. Physics-informed neural networks with hard constraints for inverse design. *SIAM Journal on Scientific Computing*, 43(6):B1105–B1132, 2021.

Martel, J. N., Lindell, D. B., Lin, C. Z., Chan, E. R., Monteiro, M., and Wetzstein, G. Acorn: Adaptive coordinate networks for neural scene representation. *arXiv preprint arXiv:2105.02788*, 2021.

McAdams, A., Zhu, Y., Selle, A., Empey, M., Tamstorf, R., Teran, J., and Sifakis, E. Efficient elasticity for character skinning with contact and collisions. In *ACM SIGGRAPH 2011 papers*, pp. 1–12. 2011.

Mehta, I., Chandraker, M., and Ramamoorthi, R. A level set theory for neural implicit evolution under explicit flows. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part II*, pp. 711–729. Springer, 2022.

Mescheder, L., Oechsle, M., Niemeyer, M., Nowozin, S., and Geiger, A. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4460–4470, 2019.

Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., and Ng, R. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European conference on computer vision*, pp. 405–421. Springer, 2020.

Mullen, P., Crane, K., Pavlov, D., Tong, Y., and Desbrun, M. Energy-preserving integrators for fluid animation. *ACM Transactions on Graphics (TOG)*, 28(3):1–8, 2009.

Müller, M., Chentanez, N., Kim, T.-Y., and Macklin, M. Air meshes for robust collision handling. *ACM Transactions on Graphics (TOG)*, 34(4):1–9, 2015.

Müller, T., Evans, A., Schied, C., and Keller, A. Instant neural graphics primitives with a multiresolution hash encoding. *arXiv preprint arXiv:2201.05989*, 2022.

Museth, K. Vdb: High-resolution sparse volumes with dynamic topology. *ACM transactions on graphics (TOG)*, 32(3):1–22, 2013.

Narain, R., Samii, A., and O'brien, J. F. Adaptive anisotropic remeshing for cloth simulation. *ACM transactions on graphics (TOG)*, 31(6):1–10, 2012.

Pan, S., Brunton, S. L., and Kutz, J. N. Neural implicit flow: a mesh-agnostic dimensionality reduction paradigm of spatio-temporal data. *arXiv preprint arXiv:2204.03216*, 2022.

Park, J. J., Florence, P., Straub, J., Newcombe, R., and Lovegrove, S. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 165–174, 2019.

Park, K., Sinha, U., Barron, J. T., Bouaziz, S., Goldman, D. B., Seitz, S. M., and Martin-Brualla, R. Nerfies: Deformable neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 5865–5874, 2021a.

Park, K., Sinha, U., Hedman, P., Barron, J. T., Bouaziz, S., Goldman, D. B., Martin-Brualla, R., and Seitz, S. M. Hypernerf: a higher-dimensional representation for topologically varying neural radiance fields. *ACM Transactions on Graphics (TOG)*, 40(6):1–12, 2021b.

Pfaff, T., Fortunato, M., Sanchez-Gonzalez, A., and Battaglia, P. W. Learning mesh-based simulation with graph networks. *arXiv preprint arXiv:2010.03409*, 2020.

Raissi, M., Perdikaris, P., and Karniadakis, G. E. 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.

Reddy, J. N. *Introduction to the Finite Element Method*. McGraw-Hill Education, New York, 4th edition edition, 2019.

Roache, P. J. *Fundamentals of computational fluid dynamics*. Hermosa Publishers, 1998.

Sadeghirad, A., Brannon, R. M., and Burghardt, J. A convected particle domain interpolation technique to extend applicability of the material point method for problems involving massive deformations. *International Journal for numerical methods in Engineering*, 86(12):1435–1456, 2011.

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

Schwarz, K., Liao, Y., Niemeyer, M., and Geiger, A. Graf: Generative radiance fields for 3d-aware image synthesis. *Advances in Neural Information Processing Systems*, 33: 20154–20166, 2020.

Selle, A., Fedkiw, R., Kim, B., Liu, Y., and Rossignac, J. An unconditionally stable maccormack method. *Journal of Scientific Computing*, 35(2):350–371, 2008.

Setaluri, R., Aanjaneya, M., Bauer, S., and Sifakis, E. Spgrid: A sparse paged grid structure applied to adaptive smoke simulation. *ACM Transactions on Graphics (TOG)*, 33(6):1–12, 2014.

Shaham, T. R., Gharbi, M., Zhang, R., Shechtman, E., and Michaeli, T. Spatially-adaptive pixelwise networks for fast image translation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14882–14891, 2021.

Sharp, N. and Jacobson, A. Spelunking the deep: guaranteed queries on general neural implicit surfaces via range analysis. *ACM Transactions on Graphics (TOG)*, 41(4): 1–16, 2022.

Sitzmann, V., Martel, J., Bergman, A., Lindell, D., and Wetzstein, G. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33, 2020.

Smith, B., Goes, F. D., and Kim, T. Stable neo-hookean flesh simulation. *ACM Trans. Graph.*, mar 2018.

Sorkine, O. and Alexa, M. As-rigid-as-possible surface modeling. In *Symposium on Geometry processing*, volume 4, pp. 109–116, 2007.

Stam, J. Stable fluids. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pp. 121–128, 1999.

Staniforth, A. and Côté, J. Semi-lagrangian integration schemes for atmospheric models—a review. *Monthly weather review*, 119(9):2206–2223, 1991.

Takikawa, T., Litalien, J., Yin, K., Kreis, K., Loop, C., Nowrouzezahrai, D., Jacobson, A., McGuire, M., and Fidler, S. Neural geometric level of detail: Real-time rendering with implicit 3d shapes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11358–11367, 2021.

Taylor, G. I. and Green, A. E. Mechanism of the production of small eddies from large ones. *Proceedings of the Royal Society of London. Series A-Mathematical and Physical Sciences*, 158(895):499–521, 1937.

Wandel, N., Weinmann, M., and Klein, R. Learning incompressible fluid dynamics from scratch–towards fast, differentiable fluid models that generalize. *arXiv preprint arXiv:2006.08762*, 2020.

Wang, P., Liu, L., Liu, Y., Theobalt, C., Komura, T., and Wang, W. Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. *Advances in Neural Information Processing Systems*, 34:27171–27183, 2021a.

Wang, S. and Perdikaris, P. Long-time integration of parametric evolution equations with physics-informed deeponets. *arXiv preprint arXiv:2106.05384*, 2021.

Wang, S., Wang, H., and Perdikaris, P. Learning the solution operator of parametric partial differential equations with physics-informed deeponets. *Science advances*, 7(40): eabi8605, 2021b.

Wessels, H., Weißenfels, C., and Wriggers, P. The neural particle method–an updated lagrangian physics informed neural network for computational fluid dynamics. *Computer Methods in Applied Mechanics and Engineering*, 368:113127, 2020.

Wu, R. and Zheng, C. Learning to generate 3d shapes from a single example. *ACM Transactions on Graphics (TOG)*, 41(6):1–19, 2022.

Xian, Z., Tong, X., and Liu, T. A scalable galerkin multigrid method for real-time simulation of deformable objects. *ACM Trans. Graph.*, 38(6), nov 2019.

Xie, Y., Takikawa, T., Saito, S., Litany, O., Yan, S., Khan, N., Tombari, F., Tompkin, J., Sitzmann, V., and Sridhar, S. Neural fields in visual computing and beyond. *arXiv preprint arXiv:2111.11426*, 2021.

Yang, G., Belongie, S., Hariharan, B., and Koltun, V. Geometry processing with neural fields. *Advances in Neural Information Processing Systems*, 34, 2021.

Yariv, L., Kasten, Y., Moran, D., Galun, M., Atzmon, M., Ronen, B., and Lipman, Y. Multiview neural surface reconstruction by disentangling geometry and appearance. *Advances in Neural Information Processing Systems*, 33: 2492–2502, 2020.

Zehnder, J., Li, Y., Coros, S., and Thomaszewski, B. Ntopo: Mesh-free topology optimization using implicit neural representations. *Advances in Neural Information Processing Systems*, 34:10368–10381, 2021.

# A. Implementation Details

## A.1. Optimization

We solve our time-integration optimization problem (Equation (3)) with the Adam optimizer (Kingma & Ba, 2014). For all examples in our experiments, we set an initial learning rate $\mathbf{lr}_0$ and reduce it by a factor of 0.1 if the loss value does not decrease for $\mathbf{iter}_\text{p}$ iterations. We stop the optimization process when the learning rate is lower than $\mathbf{lr}_\text{min}$ or until it reaches a maximum of $\mathbf{iter}_\text{max}$ iterations. Specific values of these hyper-parameters are described for each example below. We implemented our method using the PyTorch library and performed experiments on an NVIDIA GeForce RTX 3090 GPU.

## A.2. Advection Equation

For our advection example in Figure 2, the 1D spatial domain is $\Omega = [-2, 2]$. We consider the Dirichlet boundary condition, i.e., the advected quantity at boundaries equals zero. Hence we set the boundary constraint term in Equation (4) as

$$\mathcal{C} = ||u^{n+1}(\boldsymbol{x})||_2^2, \tag{15}$$

with the weighting factor $\lambda = 1$. The initial condition for this example is

$$\hat{u^0}(\boldsymbol{x}) = e{-}\frac{(\boldsymbol{x}-\mu)^2}{2\sigma^2}, \tag{16}$$

with $\mu = -1.5$ and $\sigma = 0.1$. We set the optimization hyper-parameters $\mathbf{lr}_0 = 1e{-}4$, $\mathbf{lr}_\text{min} = 1e{-}8$, $\mathbf{iter}_\text{p} = 500$ and $\mathbf{iter}_\text{max} = 20000$. For each gradient descent iteration, we randomly sample $|\mathcal{M}| = 5000$ points within the spatial domain $[-2, 2]$. For this example, our method takes $\sim 80s$ to compute per timestep, while the grid-based method (using the same memory) takes $\sim 4e{-}3s$.

## A.3. Incompressible Euler Equations

For our 2D fluid examples, the spatial domain is $\Omega = [-1, 1] \times [-1, 1]$. We consider solid boundary conditions, i.e., the fluid cannot go through the boundaries. Recall that we adopt the operator splitting scheme. Therefore, the boundary constraint terms for the three sequential steps are

$$\begin{aligned} \mathcal{C}_{adv} &= ||\boldsymbol{u}^{n+1}_{adv\perp}(\boldsymbol{x})||_2^2 \\ \mathcal{C}_{pro} &= ||\nabla_\perp p^{n+1}(\boldsymbol{x})||_2^2 \\ \mathcal{C}_{cor} &= ||\boldsymbol{u}^{n+1}_{\perp}(\boldsymbol{x})||_2^2 \end{aligned} \tag{17}$$

where $\perp$ indicates the perpendicular direction against the boundary. The weighting factor $\lambda = 1$.

**2D Taylor-Green vortex** Standard 2D Taylor-Green is originally defined in domain $[0, 2\pi] \times [0, 2\pi]$. We translate and scale the domain to $[-1, 1] \times [-1, 1]$ such that the input range fits our MLP with the SIREN activation (Sitzmann et al., 2020). Therefore, the initial condition for the velocity field becomes

$$\begin{aligned} \hat{\boldsymbol{u}^0}(\boldsymbol{x}) = (&\frac{1}{\pi}\sin[\pi(x+1)]\cos[\pi(y+1)], \\ &-\frac{1}{\pi}\cos[\pi(x+1)]\sin[\pi[y+1]]). \end{aligned} \tag{18}$$

After the simulation, we convert it back to domain $[0, 2\pi] \times [0, 2\pi]$ for evaluation and comparison. We set the optimization hyper-parameters $\mathbf{lr}_0 = 1e{-}5$, $\mathbf{lr}_\text{min} = 1e{-}8$, $\mathbf{iter}_\text{p} = 500$ and $\mathbf{iter}_\text{max} = 20000$. The size of the sample set is $|\mathcal{M}| = 256^2$. For this example, our method takes $\sim$ 10min to compute per timestep, while the grid-based method (using the same memory) takes $\sim 0.03s$.

**Two vortices of different scale** For the example shown in Figure 4, the initial condition for the velocity field is

$$\hat{\boldsymbol{u}^0}(\boldsymbol{x}) = \begin{cases} \hat{\boldsymbol{u}^1}(\boldsymbol{x}), & \boldsymbol{x} \in [-1, 0]^2 \\ \hat{\boldsymbol{u}^2}(\boldsymbol{x}), & \boldsymbol{x} \in [\frac{7}{4}, 1]^2 \\ (0, 0), & \text{otherwise.} \end{cases} \tag{19}$$

where

$$\begin{aligned} \hat{\boldsymbol{u}^1}(\boldsymbol{x}) =&(\sin[2\pi(x+1)]\cos[2\pi(y+1)], \\ &-\cos[2\pi(x+1)]\sin[2\pi(y+1)]), \end{aligned} \tag{20}$$

$$\begin{aligned} \hat{\boldsymbol{u}^2}(\boldsymbol{x}) =&(\sin[8\pi(x-\frac{7}{4})]\cos[8\pi(y-\frac{7}{4})], \\ &-\cos[8\pi(x-\frac{7}{4})]\sin[8\pi(y-\frac{7}{4})]). \end{aligned} \tag{21}$$

The density field that we advect is initialized as

$$\hat{d^0}(\boldsymbol{x}) = \begin{cases} 1 & ||2\boldsymbol{x}+1|| \le 0.5 \text{ or } ||8\boldsymbol{x}+7|| \le 0.5 \\ 0 & \text{otherwise.} \end{cases} \tag{22}$$

Figure 10 visually illustrates the above initial conditions. We set the optimization hyper-parameters $\mathbf{lr}_0 = 1e{-}5$, $\mathbf{lr}_\text{min} = 1e{-}8$, $\mathbf{iter}_\text{p} = 500$ and $\mathbf{iter}_\text{max} = 20000$. The size of the sample set is $|\mathcal{M}| = 128^2$. For this example, our method takes $\sim$ 10min to compute per timestep.

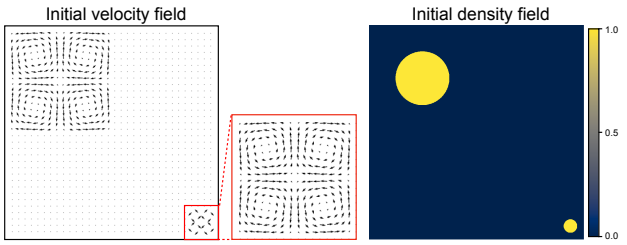


*Figure 10.* **Initial condition for the example in Figure 4.** Left: velocity field (Equation (19)). Right: density field (Equation (22)).

**Baseline setups** For PINN (Raissi et al., 2019) and PINN-sub (Krishnapriyan et al., 2021), we use the same MLP structure as ours ($\alpha = 3$ hidden layers of width $\beta = 32$,

*Table 6.* **Experiment setup for the elasticity examples.** $\rho_0$ is the density. $\lambda$ and $\mu$ are the first and second lame parameters. $\alpha$ and $\beta$ are the number of hidden layers and the dimension of the hidden features. $\mathbf{lr}_0$ and $\mathbf{iter}_{\text{max}}$ are the initial learning rate and the maximum number of iterations. $\mathbf{t}_{\text{avg}}$ is the average training time per time step. Note that the density $\rho_0$ and timestep size **dt** are reported as N/A for the quasistatic example Stretch (2D) (Figure 5).

| Example | Dim | $\lvert\mathcal{M}\rvert$ | dt | $\rho_0$ | $\lambda$ | $\mu$ | $\alpha$ | $\beta$ | $\mathbf{lr}_0$ | $\mathbf{iter}_{\text{max}}$ | $\mathbf{t}_{\text{avg}}$(s) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Collision (2D) (Figure 7)** | 2 | $100^2$ | 0.1 | $1e1$ | $2e1$ | $1e3$ | 3 | 68 | $1e{-}5$ | $1e4$ | $1.38e2$ |
| **Stretch (2D) (Figure 5)** | 2 | $100^2$ | N/A | N/A | $1e0$ | $1e3$ | 3 | 68 | $1e{-}4$ | $5e4$ | $2.30e3$ |
| **Bunny (Figure 8)** | 3 | $20^3$ | 0.1 | $1e0$ | $1e2$ | $1e3$ | 3 | 66 | $1e{-}5$ | $2e4$ | $1.70e3$ |
| **Spot (Figure 1)** | 3 | $20^3$ | 0.1 | $1e0$ | $1e2$ | $1e3$ | 3 | 66 | $1e{-}3$ | $5e3$ | $1.74e3$ |
| **Lucy (Figure 13)** | 3 | $20^3$ | 0.1 | $1e0$ | $1e3$ | $1e3$ | 3 | 128 | $1e{-}4$ | $2e4$ | $1.16e3$ |

with SIREN activation). For piDeepONet (Wang et al., 2021b), we use $\alpha = 3$ hidden layers of width $\beta = 32$ with Tanh activation for both the branch net and trunk net. Since they do not support the operator-splitting scheme, we follow the previous literature (Chuang & Barba, 2022) and use the residual of incompressible Euler equation as the physics-informed training objective,

$$\mathcal{L}_{\theta_{\boldsymbol{u}},\theta_p} = \underbrace{\sum_{\substack{\boldsymbol{x}\in\mathcal{M}\subset\Omega\\ t\in\mathcal{T}}} ||\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u}\cdot\nabla\boldsymbol{u} + \nabla p||_2^2 + ||\nabla\cdot\boldsymbol{u}||_2^2}_{\text{PDE residual}} + \underbrace{\sum_{\substack{\boldsymbol{x}\in\Omega\\ t=0}} ||\boldsymbol{u}-\hat{\boldsymbol{u}}^0||_2^2}_{\text{initial condition}} + \underbrace{\sum_{\substack{\boldsymbol{x}\in\partial\Omega\\ t\in\mathcal{T}}} ||\boldsymbol{u}_\perp(\boldsymbol{x})||_2^2}_{\text{boundary condition}} \tag{23}$$

where $\boldsymbol{u} = \boldsymbol{u}_{\theta_{\boldsymbol{u}}}(\boldsymbol{x}, t)$ and $p = \boldsymbol{u}_{\theta_p}(\boldsymbol{x}, t)$ are parameterized by MLPs. The number of spatial samples used for each training iteration is the same as ours. We train their models until convergence.

Another baseline (`Ours-residual` in Table 3) uses an implicit-time-discretized version of this objective function for time integration, i.e.,

$$\boldsymbol{\mathcal{I}} = \sum_{\substack{\boldsymbol{x}\in\mathcal{M}\subset\Omega\\ t\in\mathcal{T}}} ||\frac{\boldsymbol{u}^{n+1}-\boldsymbol{u}^n}{t} + \boldsymbol{u}^{n+1}\cdot\nabla\boldsymbol{u}^{n+1} + \nabla p^{n+1}||_2^2 + \sum_{\substack{\boldsymbol{x}\in\mathcal{M}\subset\Omega\\ t\in\mathcal{T}}} ||\nabla\cdot\boldsymbol{u}^{n+1}||_2^2 + \sum_{\substack{\boldsymbol{x}\in\partial\Omega\\ t\in\mathcal{T}}} ||\boldsymbol{u}_\perp^{n+1}(\boldsymbol{x})||_2^2. \tag{24}$$

### A.4. Elastodynamic Equation

**Initial and Boundary Conditions** For our 2D elasticity examples in Figure 5 and Figure 7, the 2D spatial domain is $\Omega = [-1, 1] \times [-1, 1]$. For our 3D elasticity examples in Figure 12, the 3D spatial domain is $\Omega = [-1, 1] \times [-1, 1] \times [-1, 1]$. For our 2D and 3D examples involving nonregular geometry (Figure 8, Figure 1 and Figure 13), the spatial domain is the interior of the shape, including the boundary. The initial condition for all the elasticity examples is

$$\hat{\boldsymbol{\phi}}^0(\boldsymbol{x}) = (0, 0) \text{ (2D)}, \quad \hat{\boldsymbol{\phi}}^0(\boldsymbol{x}) = (0, 0, 0) \text{ (3D)} \tag{25}$$

The boundary constraint for elasticity examples involves positional constraints or collision constraints. Positional constraints, or *Dirichlet boundary conditions*, can be realized by defining the position of the constraint set $\partial\Omega$ as the desired goal positions $\overline{\boldsymbol{\phi}}_{\partial\Omega}$:

$$\boldsymbol{\mathcal{I}}_{\text{pos}} = \|\boldsymbol{\phi}_{\partial\Omega}^{n+1} - \overline{\boldsymbol{\phi}}_{\partial\Omega}\|_2^2. \tag{26}$$

Collision constraints can be handled by adding unilateral constraints dynamically and viewing the collision penalty force as an external force. Specifically, for a colliding point $\boldsymbol{q}_c$, we first find the closest surface point $\boldsymbol{b}_c$ with normal $\boldsymbol{n}_c$, and define our spring-like collision penalty force as:

$$\boldsymbol{f}_{\text{col}} = k_{\text{col}}((\boldsymbol{b}_c - \boldsymbol{q}_c)^\top \boldsymbol{n}_c)\boldsymbol{n}_c. \tag{27}$$

where $k_{\text{col}}$ is the ratio for the collision penalty force.

The corresponding collision energy can be defined as the work exerted by the collision force:

$$\boldsymbol{\mathcal{I}}_{\text{col}} = \rho_0 \boldsymbol{f}_{\text{col}}^T \boldsymbol{\phi}^{n+1}. \tag{28}$$

**Experiment Setup** For all the 2D comparisons under the same memory usage, we use $\alpha = 3$ hidden layers of width $\beta = 68$ with SIREN activation function (Sitzmann et al., 2020) for our MLP, which takes the same memory (57 KB) as the FEM mesh (0.8K vertices, 1.5K faces) and MPM point cloud (1.7K points) in use. We initialize the 2D deformation field of the network to be zero using $|\mathcal{M}| = 1000^2$ uniform and random samples. Then we train the network using $|\mathcal{M}| = 100^2$ uniform and random samples at each training iteration. We use Bartels (Levin, 2020) and Taichi (Hu et al., 2019) to perform the FEM and MPM simulation, respectively. We run our FEM and MPM comparison on CPU using a MacBook Pro with the Apple M2 processor and 24GB of RAM.

For the 3D comparison under the same memory usage, for the bunny example (Figure 8), we use $\alpha = 3$ hidden layers of width $\beta = 66$ with SIREN activation function for our MLP, which takes the same memory (53 KB) as the FEM mesh (0.5K vertices, 1.5K tetrahedra) in use. For the statue example (Figure 13), we use $\alpha = 3$ hidden layers of width $\beta = 128$ with SIREN activation function for our

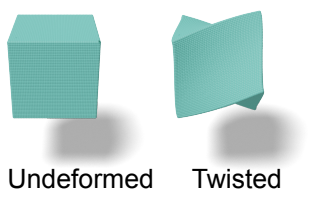


*Figure 11.* **Twisting test.** Quasistatic simulation in 3D. The right end is twisted 45 degrees.

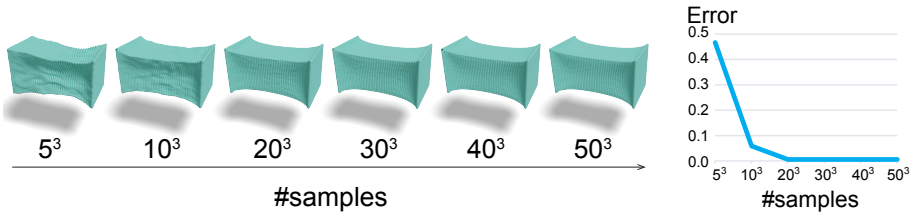


*Figure 12.* **Sampling convergence test**. We use a different number of spatial samples $|\mathcal{M}|$ and optimize for the same number of gradient descent iterations. On the right, we further report the error with respect to the reference result (using $|\mathcal{M}| = 50^3$). As we increase the number of spatial samples, the simulation result converges.

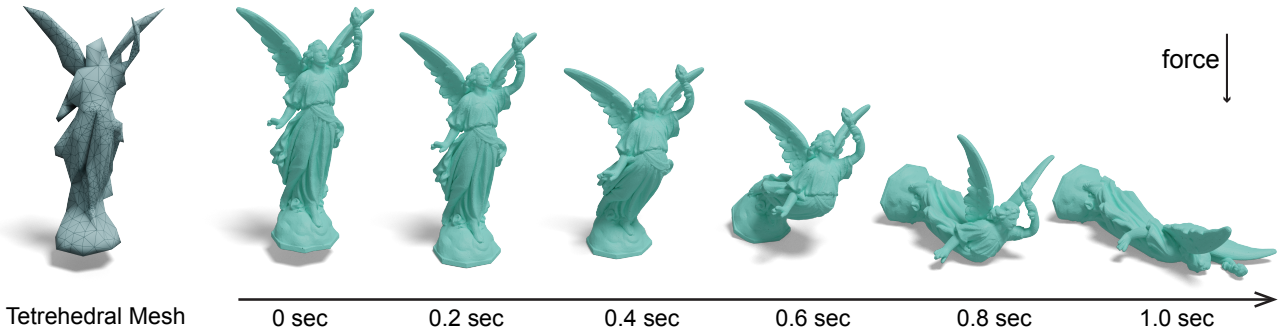


*Figure 13.* **The statue collides with the ground and deforms.** Our implicit neural representation (green, right) is capable of capturing more fine geometry details compared to the traditional tetrahedral mesh representation (blue, left) under the same memory footprint.

MLP, which takes the same memory (197 KB) as the FEM mesh (2.0K vertices, 7.0K tetrahedra) in use. We initialize the 3D deformation field of the network to be zero using $|\mathcal{M}| = 100^3$ uniform and random samples. Then we train the network using $|\mathcal{M}| = 20^3$ uniform and random samples at each training iteration. Here for simplicity, we use the mesh vertices as the uniform samples. We further report all the parameters and experiment setup in Table 6. In addition, we set the hyper-parameters $\textbf{iter}_{\text{p}} = 800$ and $\textbf{lr}_{\text{min}} = 1e{-}8$ for all elasticity examples and assume a constant density in the reference space.

The geometry (i.e., the undeformed shape) can be any representation (e.g., analytical, mesh, or level set), as long as it allows sampling within the volume. For our examples in Figure 5 and Figure 7, the geometry (a square) is represented analytically. For examples in Figure 8 and Figure 13, the geometry is represented using the original high-resolution mesh.

For sampling of the shapes involving nonregular geometry, for simplicity, we choose to use a triangle or tetrahedral mesh and perform sampling within the volume *during the training*. An ideal alternative would be adopting the implicit representation of the surface and performing rejection sampling based on it.

For rendering, we sample a sufficient number of points from the undeformed shape and evaluate the trained model at time $t$ on the sample positions to predict their deformation. Thus the deformed shape $x^t$ at each time step $t$ can be obtained by applying the deformation field $\phi^t$ on the sample points of undeformed shape $x^0$, i.e., $x^t = x^0 + \phi^t(x^0)$. For visualization, we only sample the surface of the shape in 3D cases. Then we render the shape as a dense point cloud.

## B. Additional Results

### B.1. Elastodynamic Equation

In Figure 11, we demonstrate that our method exhibits volume-preserving property on a 3D twisting example. In Figure 13, we provide another example involving complex contact-induced deformations.

The L2 distance errors in Figure 6 and Table 4 are computed using the vertices of the reference FEM mesh. Specifically, the L2 distance is defined between the deformed positions of those vertices in the reference solution and in our/compared FEM solution. In our solution, the deformed positions of those vertices are queried via direct network inference. In the compared low-resolution FEM solution, the deformed positions of those vertices are calculated using the barycentric interpolation.

Finally, we demonstrate our method's qualitative and quantitative convergence when increasing the number of spatial samples for training. In Figure 12, we compare the quasistatic stretching results when using a different number of spatial samples (recall Section 3.2 Spatial Sampling), and report the error with respect to the reference result (using $|\mathcal{M}| = 50^3$).