Neural Stress Fields for Reduced-order Elastoplasticity and Fracture

Zeshun Zong
UCLA
USA

Xuan Li
UCLA
USA

Minchen Li
UCLA
Carnegie Mellon University
USA

Maurizio M. Chiaramonte
Meta Reality Labs Research
USA

Wojciech Matusik
MIT CSAIL
USA

Eitan Grinspun
University of Toronto
Canada

Kevin Carlberg
Meta Reality Labs Research
USA

Chenfanfu Jiang
UCLA
USA

Peter Yichen Chen
MIT CSAIL
USA

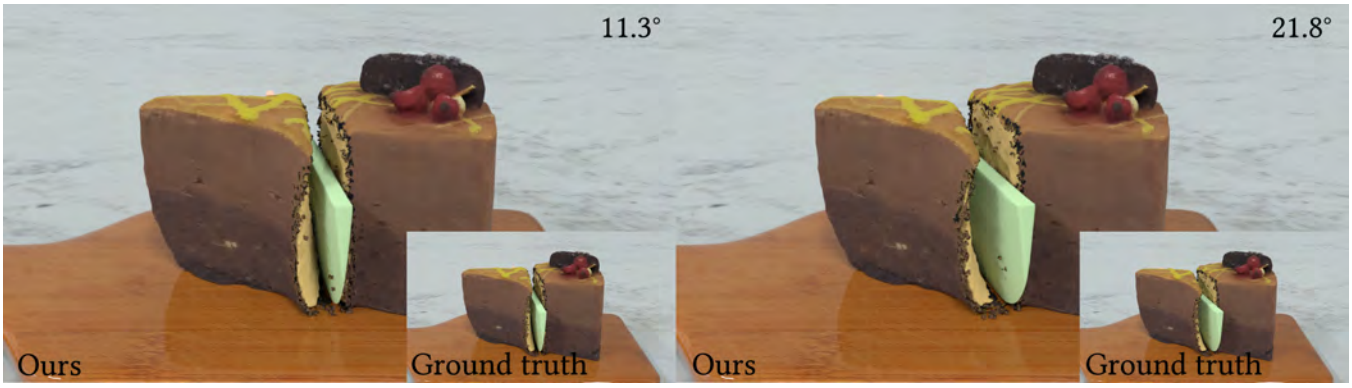

Figure 1: Our reduced-order model accurately simulates the cutting of a chocolate cake at various angles by time-stepping in a latent space of only dimension $r = 6$. The original full-order simulation employs $200,000$ particles. Courtesy of dimension reduction, our approach is 10.2× faster than the full-order simulation.

ABSTRACT

We propose a hybrid neural network and physics framework for reduced-order modeling of elastoplasticity and fracture. State-of-the-art scientific computing models like the Material Point Method (MPM) faithfully simulate large-deformation elastoplasticity and fracture mechanics. However, their long runtime and large memory consumption render them unsuitable for applications constrained by computation time and memory usage, e.g., virtual reality. To overcome these barriers, we propose a reduced-order framework. Our key innovation is training a low-dimensional manifold for the Kirchhoff stress field via an implicit neural representation. This low-dimensional neural stress field (NSF) enables efficient evaluations of stress values and, correspondingly, internal forces at arbitrary spatial locations. In addition, we also train neural deformation and affine fields to build low-dimensional manifolds for the deformation and affine momentum fields. These neural stress, deformation, and affine fields share the same low-dimensional latent space, which uniquely embeds the high-dimensional simulation state. After training, we run new simulations by evolving in this single latent space, which drastically reduces the computation time and memory consumption. Our general continuum-mechanics-based reduced-order framework is applicable to any phenomena governed by the elastodynamics equation. To showcase the versatility of our framework, we simulate a wide range of material behaviors, including elastica, sand, metal, non-Newtonian fluids, fracture, contact, and collision. We demonstrate dimension reduction by up to 100,000× and time savings by up to 10×.

SA Conference Papers '23, December 12–15, 2023, Sydney, NSW, Australia

ACM ISBN 979-8-4007-0315-7/23/12.
https://doi.org/10.1145/3610548.3618207

CCS CONCEPTS

• **Computing methodologies** → **Physical simulation**.

KEYWORDS

neural field, reduced-order model, model reduction, the material point method

ACM Reference Format:
Zeshun Zong, Xuan Li, Minchen Li, Maurizio M. Chiaramonte, Wojciech Matusik, Eitan Grinspun, Kevin Carlberg, Chenfanfu Jiang, and Peter Yichen Chen. 2023. Neural Stress Fields for Reduced-order Elastoplasticity and Fracture. In *SIGGRAPH Asia 2023 Conference Papers (SA Conference Papers '23), December 12–15, 2023, Sydney, NSW, Australia.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3610548.3618207

1 INTRODUCTION

Physical simulation plays a crucial role in computational mechanics, digital twins, computational design, robotics, animation, visual effects, and virtual reality. A crucial class of these physical simulations are those governed by the conservation of momentum equation [Gonzalez and Stuart 2008],

$$R\ddot{\boldsymbol{\phi}} = \nabla^{\boldsymbol{X}} \cdot \boldsymbol{P} + R\boldsymbol{b}, \tag{1}$$

where $\boldsymbol{P}$ is the first Piola-Kirchhoff stress, $\boldsymbol{\phi}$ is the deformation map, R is the initial density, $\boldsymbol{b}$ is the body force, and $\boldsymbol{X} \in \Omega_0$ is the reference position defined over domain Ω_0. This partial differential equation (PDE) governs a wide range of elastoplastic behaviors.

To numerically solve this PDE, one has to spatially and temporally discretize it, e.g., via finite difference, finite element, or finite volume methods. A particularly flexible discretization framework is the material point method (MPM) [Jiang et al. 2016; Sulsky et al. 1995]. MPM discretizes the spatial field via both Lagrangian particles and Eulerian grids. Thanks to this dual discretization paradigm, MPM thrives at handling large deformations, topology changes, and self-contact.

Nevertheless, MPM's versatility also comes at the cost of computation burden, in terms of both long runtime and excessive memory consumption. To obtain accurate results, MPM tracks a large number of state variables through the particles, often at the order of millions. Such a computation bottleneck significantly hinders the feasibility of deploying MPM in time-critical and memory-bound applications. Notably, MPM's high-dimension state variables also pose a challenge in applications where synchronization is required. For example, in virtual reality and cloud gaming, multiple users share the same simulated physical environment; each user's simulation state needs to be efficiently shared with others via internet streaming. Synchronizing millions of MPM particle data at frame rate is simply not possible.

We propose to solve these computational challenges via reduced-order modeling (ROM), also known as model reduction [Barbič and James 2005]. ROM reduces the computation cost by training a low-dimensional latent embedding of the original high-dimensional simulation data. After training, instead of evolving the original high-dimensional state variables over time, ROM only needs to time-step in the low-dimensional latent space, and synchronization between users only requires sharing the low-dimensional latent vector. The classic reduced-order, elasticity-only finite element method (FEM) [Sifakis and Barbic 2012] trains a low-dimensional embedding for the (discretized) deformation map $\boldsymbol{\phi}$ in eq. (1). However, the low-dimensional deformation embedding alone is not enough for MPM and elastoplasticity simulations in general.

History-dependent plasticity state variables. MPM simulations feature history-dependent effects, e.g., plastic deformations of sands or metals. The low-dimensional deformation embedding by itself is unable to determine the plasticity state variables that are crucial for MPM time-stepping.

Deformation gradients as independent state variables. MPM treats the deformation *gradient* as a separate state variable that evolves independently from deformation state variables. Again, the low-dimensional deformation embedding cannot capture these deformation gradients.

Our key observation is that the ultimate purpose of *all* these additional state variables is computing the stress field $\boldsymbol{P}$ in eq. (1). As such, we can bypass the need to capture these intermediate state variables by directly training a low-dimensional embedding for the stress field itself. The low-dimensional stress and deformation embedding together capture all the information necessary for MPM time-stepping. We construct the low-dimensional stress embeddings via implicit neural representations, also known as neural fields. Our neural stress field (NSF) approach enables stress evaluation and, in turn, force evaluation at arbitrary spatial locations. In a similar vein, we build low-dimensional neural deformation fields. To support MPM's affine particle-in-cell transferring scheme [Jiang et al. 2015], we also build low-dimensional neural affine fields for the affine momentum field. All these three neural fields share the same latent space.

After training, we solve new physical simulation problems via projection-based latent space dynamics [Benner et al. 2015; Carlberg et al. 2017]. During this PDE-constrained latent space dynamics stage, we obtain computation savings by evaluating the neural fields only at a small spatial subset, similar to the idea of cubature [An et al. 2008]. Our general, stress-based ROM approach works with any problem governed by the momentum equation eq. (1). To showcase the versatility of our approach, we validate NSF on a wide range of elastoplastic phenomena, including elastica, fracture, metal, sand, non-Newtonian fluids, contact, and collision. We demonstrate dimension reduction of 100,000× and computation savings of 10×.

2 RELATED WORK

The Material Point Method. Sulsky et al. [1995] introduced MPM by combining Lagrangian and Eulerian techniques for solid mechanics, drawing upon the earlier works by Brackbill and Ruppel [1986]; Harlow [1962] on PIC/FLIP. Since its introduction to the graphics community [Hegemann et al. 2013; Stomakhin et al. 2013], MPM has garnered considerable attention. Its primary advantage in modeling elastoplastic materials lies in its capability to handle extreme deformation and topological changes. MPM has been successfully applied to simulate various phenomena, including granular media [Chen et al. 2021; Daviet and Bertails-Descoubes 2016; Klár et al. 2016; Yue et al. 2018], non-Newtonian fluids [Fei et al. 2019; Yue et al. 2015], viscoelasticity [Fang et al. 2019], fracture [Wang et al. 2019; Wolper et al. 2020, 2019], and thermomechanics [Ding et al. 2019]. Efforts have been made to speed up MPM simulations through GPU [Fei et al. 2021; Gao et al. 2018; Wang et al. 2020b], multi-node [Qiu et al. 2023], and multigrid [Wang et al. 2020a] accelerations, as well as compiler optimization [Hu et al. 2019]. However, the substantial computational cost and memory consumption of MPM still present challenges that need to be addressed.

Reduced-order Modeling. Classic reduced-order modeling methods employ linear subspaces [Barbič and James 2005; Sifakis and Barbic 2012]. These subspaces are often constructed via principal component analysis and, equivalently, proper orthogonal decomposition [Berkooz et al. 1993; Holmes et al. 2012]. These linear subspaces have been successively applied to solids [An et al. 2008; Barbič and Zhao 2011; Kim and James 2009; Xu et al. 2015; Yang et al. 2015] and fluids [Kim et al. 2019; Kim and Delaney 2013; Treuille et al. 2006; Wiewel et al. 2019]. Recently ROM methods have been exploring nonlinear low-dimensional manifolds, often leveraging autoencoder neural networks [Lee and Carlberg 2020]. These nonlinear approaches enable smaller latent space dimensions in comparison with the classic linear approaches [Fulton et al. 2019; Shen et al. 2021]. Our technique also falls into this nonlinear model reduction category.

Relatedly, there has been lots of progress in data-driven latent space dynamics [Lusch et al. 2018], and the entire latent space evolution is strictly learned via another neural network, e.g., recurrent neural networks [Wiewel et al. 2019]. By contrast, our method follows the classic, invasive ROM literature and evolves the latent space using the numerical methods and PDEs that were used to generate the training data. In our method, the latent space dynamics are entirely PDE-based without any data-driven component.

Neural Fields. A neural field [Xie et al. 2021] parameterizes a spatially dependent vector field via a neural network. The pioneering works by Chen and Zhang [2019]; Mescheder et al. [2019]; Park et al. [2019] employ this representation for signed distance fields, where different latent space vector corresponds to different geometries. Since then, it has been widely adopted for neural rendering [Mildenhall et al. 2020], topology optimization [Zehnder et al. 2021], geometry processing [Aigerman et al. 2022; Yang et al. 2021], and various PDE problems [Chen et al. 2022; Raissi et al. 2019]. Recently, Chen et al. [2023a,b]; Pan et al. [2023] have leveraged neural fields for ROM. Notably, Chen et al. [2023a] build a neural-field-based, reduced-order framework for MPM. Their approach constructs a low-dimensional embedding only for the deformation field. Consequently, their method is unable to handle history-dependent plasticity, and the deformation gradient computed from differentiating the learned deformation field is too inaccurate for large deformation phenomena such as a fracture. As a major point of departure, we train a low-dimensional manifold directly for the stress field and can therefore handle both plasticity and fracture. Furthermore, we achieve angular momentum conservation by training a low-dimensional neural affine field while [Chen et al. 2023a]'s formulation suffers from excessive dissipation.

3 BACKGROUND: FULL-ORDER MPM

This section will briefly recap the essential ingredients of the full-order MPM model. Sections 4 and 5 will introduce the corresponding reduced-order model. We refer to Jiang et al. [2016]; Sulsky et al. [1995] for additional MPM details.

3.1 Finite strain elasticity and elastoplasticity

Let $\Omega_0 \subset \mathbb{R}^3$ denote the material space and Ω_t the world space at time t. We are interested in the dynamics of a continuum in time $t \in [0, T]$. The deformation map $\boldsymbol{x} := \boldsymbol{\phi}(\boldsymbol{X}, t)$ maps $\boldsymbol{X} \in \Omega_0$ to world space coordinate $\boldsymbol{x} \in \Omega_t$. From the Lagrangian view, the dynamics of a continuum can be described by a density field $R(\boldsymbol{X}, t) : \Omega_0 \times [0, T] \to \mathbb{R}$ and a velocity field $\boldsymbol{V}(\boldsymbol{X}, t) = \frac{\partial \boldsymbol{\phi}(\boldsymbol{X}, t)}{\partial t} : \Omega_0 \times [0, T] \to \mathbb{R}^3$. They are governed by the conservation of mass

$$R(\boldsymbol{X}, t)J(\boldsymbol{X}, t) = R(\boldsymbol{X}, 0), \tag{2}$$

and the conservation of momentum

$$R(\boldsymbol{X}, 0)\frac{\partial \boldsymbol{V}}{\partial t}(\boldsymbol{X}, t) = \nabla^{\boldsymbol{X}} \cdot \boldsymbol{P} + R(\boldsymbol{X}, 0)\boldsymbol{g}. \tag{3}$$

Here $J = \det(\boldsymbol{F})$, $\boldsymbol{F} = \frac{\partial \boldsymbol{\phi}}{\partial \boldsymbol{X}}(\boldsymbol{X}, t)$ is the deformation gradient, $\boldsymbol{P}$ is the first Piola-Kirchhoff stress, and $\boldsymbol{g}$ is the gravity term. $\boldsymbol{P}$ can be related to the Kirchhoff stress $\boldsymbol{\tau}$ as $\boldsymbol{P} = \boldsymbol{\tau}\boldsymbol{F}^{-T}$.

For a hyperelastic solid, the Kirchhoff stress can be computed as $\boldsymbol{\tau} = \frac{\partial \psi}{\partial \boldsymbol{F}}(\boldsymbol{F})\boldsymbol{F}^T$, where ψ is the energy density function of the chosen constitutive model. For an elastoplastic continuum, the deformation gradient is multiplicatively decomposed into $\boldsymbol{F} = \boldsymbol{F}^E\boldsymbol{F}^P$, with the former being the elastic deformation that supplies elastic force, and the latter being the permanent plastic deformation gradient. The decomposition requires that $\boldsymbol{\tau}(\boldsymbol{F}^E)$ lies within an admissible region defined by some yield condition $y(\boldsymbol{\tau}) < 0$. Given $\boldsymbol{F}$, $\boldsymbol{F}^E$ evolves from $\boldsymbol{F}$, following some plastic flow until the yield condition is satisfied. The procedure is often called return mapping.

3.2 MPM discretization

MPM discretizes a continuum bulk into a set of Lagrangian particles p, and discretizes time t into a sequence of timesteps $t_0 = 0, t_1, t_2, \ldots$ Here we take a fixed stepsize Δt, so $t_n = n\Delta t$. The advection is performed on particles so eq. (2) is naturally satisfied. If we approximate $\boldsymbol{V}^n$ by $\frac{1}{\Delta t}(\boldsymbol{X}^{n+1} - \boldsymbol{X}^n)$, and assume no gravity and free surface for clarity, for an arbitrary test function Q, the weak form of eq. (3) is then given by

$$\int_{\Omega_0} R(\boldsymbol{X}, 0)\frac{1}{\Delta t}\left(\boldsymbol{V}^{n+1} - \boldsymbol{V}^n\right)Q d\boldsymbol{X} = -\int_{\Omega_0} \boldsymbol{P}\nabla^{\boldsymbol{X}} Q d\boldsymbol{X}. \tag{4}$$

Pushing forward the integral from Ω_0 to $\Omega_n = \Omega_{t_n}$, we obtain

$$\int_{\Omega_n} \rho\left(\boldsymbol{x}, t^n\right)\frac{1}{\Delta t}\left(\boldsymbol{v}^{n+1} - \boldsymbol{v}^n\right)q d\boldsymbol{x} = -\int_{\Omega_n} \frac{1}{J^n}\boldsymbol{P}\boldsymbol{F}^{nT}\nabla^{\boldsymbol{x}} q d\boldsymbol{x}, \tag{5}$$

where $\rho, \boldsymbol{v}^n, \boldsymbol{v}^{n+1}$ and q are the Eulerian counterparts of $R, \boldsymbol{V}^n, \boldsymbol{V}^{n+1}$ and Q, respectively [Jiang et al. 2016].

MPM adopts B-Spline-based interpolations and uses material particles p as quadratures to approximate the integration eq. (5). Let m_p denote the mass of particle p with initial position $\boldsymbol{X}_p$. Denote its position and velocity at time t_n by $\boldsymbol{x}_p^n$ and $\boldsymbol{v}_p^n$. Let m_i and $\boldsymbol{v}_i$ denote the mass and velocity on background grid node i at position $\boldsymbol{x}_i$. Let $N(\boldsymbol{x})$ denote the weight function, and $w_{ip}^n = N\left(\boldsymbol{x}_p^n - \boldsymbol{x}_i\right)$. Employing mass lumping, we can express the force equilibrium as

$$\frac{1}{\Delta t}m_i^n\left(\boldsymbol{v}_i^{n+1} - \boldsymbol{v}_i^n\right) = -\sum_p \boldsymbol{\tau}_p^n \nabla w_{ip}^n V_p^0, \tag{6}$$

thus providing a way to update the next stage grid velocities $\boldsymbol{v}_i^{n+1}$. Here V_p^0 and $\boldsymbol{\tau}_p^n$ are the initial volume and Kirchhoff stress at time t_n of material particle p.

3.3 MPM algorithm

At each step, particle mass and momentum are transferred to grid nodes. Grid velocities are updated and then transferred back to particles for advection. Let C_p^n denote the affine momentum of particle p at time t_n. The explicit MPM algorithm can therefore be summarized as the following:

(1) **P2G.** Transfer mass and momentum from particles to grid as $m_i^n = \sum_p w_{ip}^n m_p$ and $m_i^n v_i^n = \sum_p w_{ip}^n m_p \left(v_p^n + C_p^n \left(x_i - x_p^n\right)\right)$, if the APIC transfer scheme is adopted. If the conventional PIC scheme is adopted, the latter is simply replaced by $m_i^n v_i^n = \sum_p w_{ip}^n m_p v_p^n$.

(2) **Grid update.** Update grid velocities at next timestep by $v_i^{n+1} = v_i^n - \frac{\Delta t}{m_i} \sum_p \tau_p^n \nabla w_{ip}^n V_p^0 + \Delta t g$. Collision and Dirichlet boundary conditions are also handled at this stage.

(3) **G2P.** Transfer velocities back to particles and update particle states. $v_p^{n+1} = \sum v_i^{n+1} w_{ip}^n$, $x_p^{n+1} = x_p^n + \Delta t v_p^{n+1}$,

$$C_p^{n+1} = \frac{12}{\Delta x^2 (b+1)} \sum_i w_{ip}^n v_i^{n+1} \left(x_i^n - x_p^n\right)^T,$$

$$F_p^{\text{trial},n+1} = \left(\mathbf{I} + \Delta t \sum_i v_i^{n+1} \left(\nabla w_{ip}^n\right)^T\right) F_p^{E,n},$$

$F_p^{E,n+1} = \text{returnMap}(F_p^{\text{trial},n+1})$ and $\tau_p^{n+1} = \tau(F_p^{E,n+1})$.

Here b is the B-spline degree, and Δx is the Eulerian grid spacing. If additional damping is desired, RPIC can be added in the computation of C_p as in [Fang et al. 2019].

4 REDUCED-ORDER MODEL: KINEMATICS

To reduce the full-order MPM model, we will construct a nonlinear approximation to the solution of eq. (3) over a low-dimensional manifold. A schematic illustration is shown in fig. 2.

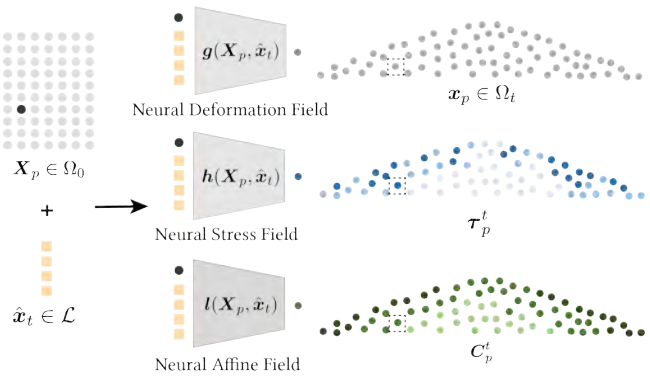

Figure 2: Latent space kinematics. Given a latent space vector $\hat{x}_t \in \mathcal{L}$, evaluating the neural deformation, stress, and affine fields at *any* reference position $X_p \in \Omega_0$ (e.g., the black dot in Ω_0) results in the corresponding deformation, stress, and affine momentum at time t at the current position (the boxed dot in Ω_t).

4.1 Low-dimensional Manifold Construction

Let the continuous field $f(X, t; \mu) : \Omega_0 \times [0, T] \to \mathbb{R}^m$ denote any relevant state variable in the solution to eq. (3) for $X \in \Omega_0$ at time t. Example state variables include the deformation map, stress, etc. Here, μ is the generalized problem parameter, including but not limited to material parameters, initial conditions, and boundary conditions. Choice of μ for each experiment will be detailed in section 6. We seek a continuous field $\hat{f}(\cdot; \hat{x})$ defined over Ω_0 and parameterized by $\hat{x} \in \mathcal{L}$, a low-dimensional latent space, such that

$$\hat{f}(X; \hat{x}(t, \mu)) \approx f(X, t; \mu), \forall X \in \Omega_0 \text{ and } \forall t \in [0, T]. \tag{7}$$

The dimension r of $\mathcal{L} \subset \mathbb{R}^r$ is taken to be a small number so that the dynamics of a continuum becomes the evolution of the latent space vector $\hat{x}$ in a low-dimensional latent space $\mathcal{L}$. For notational simplicity, we will omit explicit dependence on μ. To computationally construct any of these low-dimensional manifolds, we will employ a neural field, also known as implicit neural representation [Xie et al. 2021]. Next, we will discuss specific MPM state variables for which we will build neural fields.

4.2 Neural Deformation Fields

Similar to classic elastic-only FEM, one must build a low-dimensional manifold for the deformation field $\phi(X, t)$ [Barbič and James 2005]. We achieve this by constructing a manifold $g(X, \hat{x})$ [Chen et al. 2023a] such that

$$\forall X \in \Omega_0, \forall t \in [0, T], g(X, \hat{x}_t) \approx \phi(X, t) = x_t. \tag{8}$$

4.3 Neural Stress Fields

Unlike elasticity-only FEM, MPM features additional history-based plastic effects and state variables. Moreover, the deformation gradient F is treated as an evolving state variable independent of x. To address these various state variables, we observe that representing the stress field is a neat yet effective approach. Since the eventual goal of *all* these state variables is computing the stress tensor, by directly building a low-dimensional manifold for the stress tensor, we avoid cumbersome treatment of numerous plasticity state variables as well as inaccurate calculation of deformation gradient. We approximate the Kirchhoff stress field $\tau(x, t)$ by a manifold $h(X, \hat{x})$ such that

$$\forall X \in \Omega_0, \forall t \in [0, T], h(X, \hat{x}_t) \approx \tau(x, t) = \tau(\phi(X, t), t). \tag{9}$$

The right hand side of eq. (5) can thus be approximated as

$$-\int_{\Omega_n} \frac{1}{J^n} \tau(x) \nabla^x q dx = -\int_{\Omega_0} \tau(X) \nabla^x q dX \approx -\int_{\Omega_0} h(X, \hat{x}_n) \nabla^x q dX,$$

which naturally fits within the spatial discretization of MPM.

Remarks. (1) An alternative approach is to use the deformation gradient to compute the stress. The deformation gradient can be computed by differentiating the neural deformation field [Chen et al. 2023a]. However, the numerical deformation gradient F_{MPM} in the full-order MPM is not computed from $\frac{\partial \phi}{\partial X}$, but rather numerically integrated. Consequently, this approach will cease to provide accurate grid forces when $\frac{\partial \phi}{\partial X}$ does not resemble F_{MPM}, e.g., in numerical fracture. A well-trained neural stress field, on the other hand, directly supplies the correct grid forces for MPM grid update. (2) Since stress is computed from the elastic part of

the deformation gradient $F^E = \text{returnMap}(F^{\text{trial}})$, the plastic flow is implicitly stored. Evaluation of the return map can be avoided in deployment, thus reducing the computational cost. Overall, our neural-stress-field approach is a general approach that allows for reduced-order solutions for all the standard plasticity models.

4.4 Neural Affine Fields

Additionally, to accommodate for the affine momentum term C used in APIC and RPIC transfer scheme (section 3.3), we construct another manifold $\boldsymbol{l}(X, \hat{\boldsymbol{x}})$ such that $\boldsymbol{l}(X, \hat{\boldsymbol{x}}_n) \approx \boldsymbol{C}(\boldsymbol{\phi}(X, t), t)$. This field enables angular momemtum conservation [Jiang et al. 2015].

4.5 Network training

Let $\mathcal{T} = \{t_0, t_1, ..., t_N = T\}$, $\mathcal{P}$ denote the set of all material particles p, $\mathcal{U}$ denote the set of problem parameters μ that we are interested in, and $\mathcal{U}_{\text{train}} \subset \mathcal{U}$ a subset for training. Let the training set be $\{(\boldsymbol{x}_p^n, \boldsymbol{\tau}_p^n, \boldsymbol{C}_p^n) : p \in \mathcal{P}, \mu \in \mathcal{U}_{\text{train}}\}$. Define $\boldsymbol{x}^n = [\boldsymbol{x}_1^n, \boldsymbol{x}_2^n, ..., \boldsymbol{x}_{|\mathcal{P}|}^n]^T$. The implementation of the three manifolds is summarized below:

(1) Train displacement decoder network $\boldsymbol{g}_{\theta_g}(X, \hat{\boldsymbol{x}})$ and encoder network $\boldsymbol{e}_{\theta_e}(\boldsymbol{x}^n)$ by

$$\min_{\theta_g, \theta_e} \sum_{\text{training set}} ||\boldsymbol{g}_{\theta_g}(\boldsymbol{X}_p, \boldsymbol{e}_{\theta_e}(\boldsymbol{x}^n)) - \boldsymbol{x}_p^n||_2^2.$$

(2) Denote the latent space vectors obtained from the encoder above as $\hat{\boldsymbol{x}}_n = \boldsymbol{e}_{\theta_e}(\boldsymbol{x}^n)$. Train stress decoder network $\boldsymbol{h}_{\theta_h}(X, \hat{\boldsymbol{x}}_n)$ by

$$\min_{\theta_h} \sum_{\text{training set}} ||\boldsymbol{h}_{\theta_h}(\boldsymbol{X}_p, \hat{\boldsymbol{x}}_n) - \boldsymbol{\tau}_p^n||_2^2.$$

(3) Train an affine momentum network $\boldsymbol{l}_{\theta_l}(X, \hat{\boldsymbol{x}}_n)$ by

$$\min_{\theta_l} \sum_{\text{training set}} ||\boldsymbol{l}_{\theta_l}(\boldsymbol{X}_p, \hat{\boldsymbol{x}}_n) - \boldsymbol{C}_p^n||_2^2.$$

Here θ_* denotes network weights. Additional training details and network architecture are listed in the supplementary material.

If the problem parameter μ contains information about return mapping, we can make the stress decoder explicitly depend on μ, i.e., $\boldsymbol{h}(X, \hat{\boldsymbol{x}}, \mu)$.

Together with the three neural networks, we have equipped ourselves with all ingredients needed to perform one step of MPM algorithm.

5 REDUCED-ORDER MODEL: DYNAMICS

After training, we can run new simulations by time-stepping in the latent space $\mathcal{L}$, from $\hat{\boldsymbol{x}}_n$ to $\hat{\boldsymbol{x}}_{n+1}$. For this, we follow the projection-based ROM approach by Chen et al. [2023a]. Our projection-based ROM approach takes three steps: (1) network inference, (2) MPM time-stepping, and (3) network inversion. The pipeline is shown in fig. 3. As we will see, since the dimension of the manifold r is much much smaller than that of the full order problem $|\mathcal{P}|$, only a small subset $\mathcal{S} \subset \mathcal{P}$ of particles, which are named sample particles, are needed to determine $\hat{\boldsymbol{x}}$ dynamics. Nevertheless, due to the non-local nature of MPM, time integration of this subset will involve a larger subset $\mathcal{N} \subset \mathcal{P}$, which we refer to as integration particles. Note that $\mathcal{S} \subset \mathcal{N}$ and $r \leq 3|\mathcal{S}| < 3|\mathcal{N}| \ll |\mathcal{P}|$. These sample and integration particles bear similarities to the cubature points often employed in reduced-order FEM [An et al. 2008]. Their exact choice will be deferred to section 5.4.

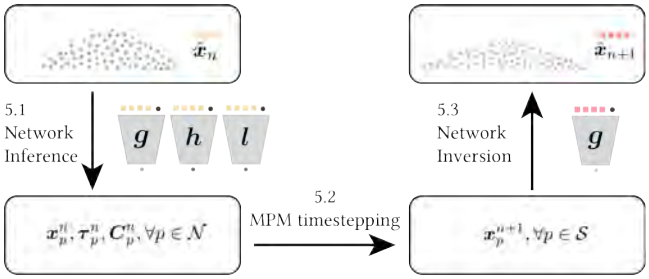

Figure 3: Latent space dynamics. We time-step the latent space via three steps. Each step involves a small spatial subset $\mathcal{S} \subset \mathcal{N} \subset \mathcal{P}$ of the original full-order MPM particles.

5.1 Network inference

At timestep t_n, given $\hat{\boldsymbol{x}}_n$, the states for all initial location $\boldsymbol{X} \in \Omega_0$, and in particular for the integration particles $p \in \mathcal{N}$ with initial position $\boldsymbol{X}_p$ can be obtained by inferencing the neural networks

$$\boldsymbol{x}_p^n = \boldsymbol{g}(\boldsymbol{X}_p, \hat{\boldsymbol{x}}_n), \quad \boldsymbol{v}_p^n = \frac{1}{\Delta t}(\boldsymbol{g}(\boldsymbol{X}_p, \hat{\boldsymbol{x}}_n) - \boldsymbol{g}(\boldsymbol{X}_p, \hat{\boldsymbol{x}}_{n-1})),$$
$$\boldsymbol{\tau}_p^n = \boldsymbol{h}(\boldsymbol{X}_p, \hat{\boldsymbol{x}}_n), \quad \boldsymbol{C}_p^n = \boldsymbol{l}(\boldsymbol{X}_p, \hat{\boldsymbol{x}}_n).$$

Note that the particle velocity here is obtained by backward differencing the position field, consistent with the explicit MPM framework.

5.2 MPM time-stepping

One step of the MPM algorithm (section 3.3) is performed on the integration particles $\mathcal{N}$ to advance to t_{n+1}. Integrating all the particles belonging to $\mathcal{N}$ guarantees that the states on sample particles $\boldsymbol{x}_p^{n+1}|_{p \in \mathcal{S}}$ are the same as if we perform the full-order MPM on all particles $p \in \mathcal{P}$. There is no approximation in this step.

5.3 Network Inversion

With the new particle positions at t_{n+1} in hand, we are able to find the corresponding $\hat{\boldsymbol{x}}_{n+1}$ by inverting the neural deformation field,

$$\hat{\boldsymbol{x}}_{n+1} = \arg\min_{\hat{\boldsymbol{x}} \in \mathbb{R}^r} \sum_{p \in \mathcal{S}} ||\boldsymbol{g}(\boldsymbol{X}_p, \hat{\boldsymbol{x}}) - \boldsymbol{x}_p^{n+1}||_2^2. \tag{10}$$

In this optimization problem, both the unknown $\hat{\boldsymbol{x}}_{n+1}$ and the number of summands $|\mathcal{S}|$ are significantly reduced. As the latent space trajectory generally evolves smoothly, with $\hat{\boldsymbol{x}}_n$ as an initial guess, eq. (10) can be rapidly solved via the Gauss-Newton method [Nocedal and Wright 1999], converging in 2-3 iterations. We can optionally further speed up this nonlinear solver via a first-order Taylor approximation [Chen et al. 2023a].

5.4 Construction of Sample and Integration Particles

The least-squares problem is well-posed provided $|\mathcal{S}| \geq r/3$. The projection will be more accurate if a decent number of sample particles can reflect the deformation of the geometry. For example, there should not be a group of sample particles that stand still

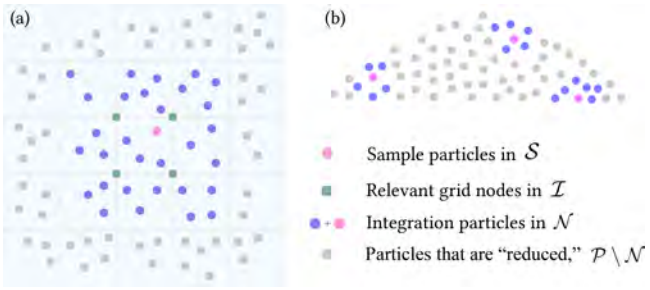

Figure 4: (a) Construction of sample particles and integration particles; (b) Sample and integration particles across the domain.

in a corner. Moreover, the sample particles can be different (in terms of both quantities and spatial distributions) at different time steps. For simplicity, we fix a set of sample particles throughout $[0, T]$. Currently, we choose sample particles via either user-defined heuristics (section 6.1 cake cutting) or random sampling (see all other experiments). Future work may consider further optimizing the sample particle choices [An et al. 2008].

Once $\mathcal{S}$ is chosen, we assemble a group of integration particles containing just enough information to evolve sample particles to the next time step. This is done by the following: a. identify the set of grid nodes relevant to $\mathcal{S}$ as $\mathcal{I} = \{\text{all grid nodes } i : \exists p \in \mathcal{S} \text{ s.t. } N_i(\boldsymbol{x}_p) \neq 0\}$, b. identify the set of integration particles relevant to $\mathcal{I}$ as $\mathcal{N} = \{\text{all particles } p \in \mathcal{P} : \exists i \in \mathcal{I} \text{ s.t. } N_i\left(\boldsymbol{g}(\boldsymbol{X}_p; \hat{\boldsymbol{x}}_n)\right) \neq 0\}$. An illustration is shown in fig. 4.

6 EXPERIMENTS

We validate the proposed reduced-order framework on a wide range of elastoplastic examples. The choice of the problem parameter μ is stated in each experiment. The Experimental statistics are summarized in table 1.

In addition to visual results, we will also report the total relative deformation error across space and time,

$$\delta = \sqrt{\frac{\sum_{n=1,2,..,N, p\in\mathcal{P}} ||\boldsymbol{g}(\boldsymbol{X}_p, \hat{\boldsymbol{x}}_n) - \boldsymbol{\phi}(\boldsymbol{X}_p, t_n)||^2}{\sum_{n=1,2,..,N, p\in\mathcal{P}} ||\boldsymbol{\phi}(\boldsymbol{X}_p, t_n)||^2}}. \tag{11}$$

Throughout this section, dataset $\mathcal{D}$ is always split as non-overlapping $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$. Neural fields are constructed with $\mathcal{D}_{\text{train}}$ and validated on $\mathcal{D}_{\text{test}}$. Furthermore, we will report the dimension reduction ratio defined by $\gamma = 3|\mathcal{P}|/r$, i.e., the dimension of the full-order model divided by the latent space dimension. See the supplementary material for additional details regarding experiments, the training dataset, generalizability, extrapolation, and elastoplastic models.

6.1 Fracture

One remarkable feature of our neural stress field is its ability to capture fracture. We first simulate the tearing of a piece of bread with $|\mathcal{P}| = 4 \times 10^4$ particles governed by pure elasticity under different Young's moduli. The problem parameter μ is the Young's modulus of the material. Weak elements are inserted in the middle region to seed the fracture. Figure 5(a) shows that our method is able to accurately generate the fracture pattern under various unseen

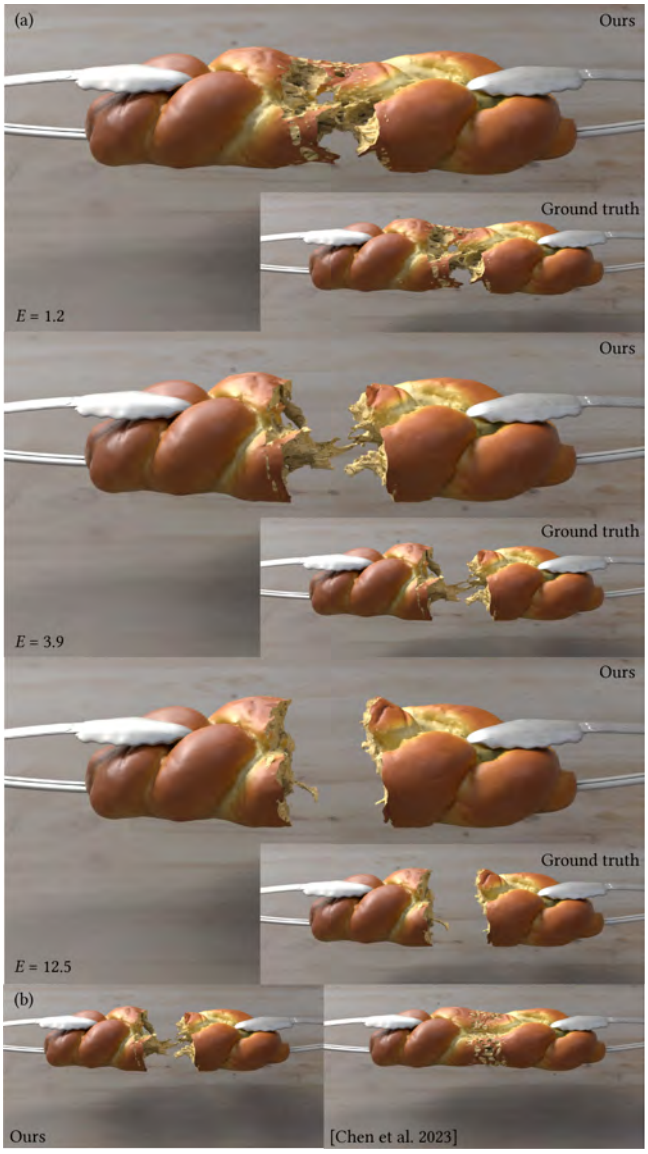

Figure 5: Tear a piece of bread. Our method accurately captures the tearing behavior at different elastic moduli. Due to a lack of accurate stress representation and the inaccurate deformation gradients computed from neural fields, the baseline approach by [Chen et al. 2023a] fails to capture the fracturing behavior.

Young's moduli. In MPM, numerical fracture happens when two (or more) sets of particles cannot see each other via the grid, after which the deformation gradient $\boldsymbol{F}$ for the two (or more) fractured pieces evolve independently. Thus, in this scenario, $\boldsymbol{F}$ computed from $\frac{\partial \boldsymbol{\phi}}{\partial \boldsymbol{X}}$ or its approximation $\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{X}}$ would provide a misleading stress $\boldsymbol{\tau}$ that is *not* what is being used in the MPM setting. As is shown in fig. 5(b), the baseline method by Chen et al. [2023a], which uses $\boldsymbol{\tau} \approx \boldsymbol{\tau}(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{X}})$, fails to reconstruct a clean fracture. Our neural stress field, on the

Figure 6: After training on low-res simulation (left), our method can directly infer high-resolution results (right) by querying the continuous neural deformation field. No additional post-processing is needed.

other hand, explicitly equips the reduced-order model with the *bona fide* stress $\boldsymbol{\tau}$ that *is* used in the ground truth MPM simulation. Here $\mathcal{S}$ consists of 450 randomly chosen particles $p \in \mathcal{P}$, and the total relative deformation error is $\delta = 1.2\%$, as is defined in eq. (11). The error for the baseline method is $\delta = 6.5\%$.

Our neural stress field is also applicable to fracture with plastic models, such as von Mises plasticity, as is shown in cake cutting in fig. 1. Here we adopt the plasticity model in [Wang 2020]. The cake is simulated with $|\mathcal{P}| = 2 \times 10^5$ particles. A spatula is slicing the cake at different angles, represented by the problem parameter μ. We select 700 particles clustered toward the middle and then reduce the sample size to 400 after $\frac{T}{2}$. The number of integration particles is 1.35×10^4 on average. The full-order and reduced-order MPM simulators are both implemented in WARP [Macklin 2022] under double precision. The neural networks are implemented in PyTorch. The total wall clock time of the full-order simulation is 14.495s, while the wall time of our reduced method is 1.417s. We achieve an overall speedup of 10.23× with an error of 1.3%. In general, since the dynamics are constrained to the low-dimensional manifold, we are also able to take a larger time step ($1.5\Delta t$.) at deployment time. In both fracture examples we choose $r = 6$, and $\gamma = 2\times10^4$ and 1×10^6, respectively. With our reduced model, we are also able to achieve considerable memory saving. In this scenario, the average memory consumption of the full-order MPM model is 1.61G, while ours is 0.79G, including both latent space physics and neural networks. The computing setup is detailed in the supplementary material.

6.2 Sand plasticity

MPM is particularly suitable for simulating granular media. We simulate a column of sand falling onto the ground under gravity. Here μ represents different friction angles. Our neural stress field can perfectly capture such a noisy stress distribution and yields excellent results on $\mathcal{D}_{\text{test}}$, with an average error of $\delta = 0.4\%$. (See fig. 7) The ground truth is simulated with $|\mathcal{P}| = 72,000$ particles, while we set $r = 6, \gamma = 3.6 \times 10^4$ and $|\mathcal{S}| = 150$. The memory consumption of full-order MPM for this scenario is 0.91G, and that of our reduced model is 0.65G.

Once trained using a low-resolution simulation, our approach can arbitrarily boost the resolution with no cost by simply evaluating the neural deformation fields at more $\boldsymbol{X}_p \in \Omega_0$. In fig. 6, we boost the resolution by 100× when running latent space dynamics.

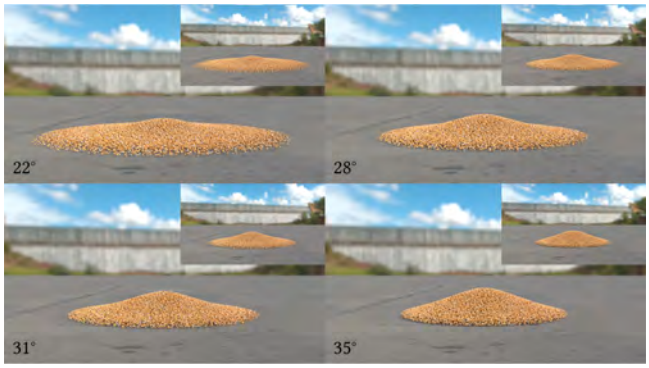

Figure 7: Simulate column collapse for sand under varying friction angles.

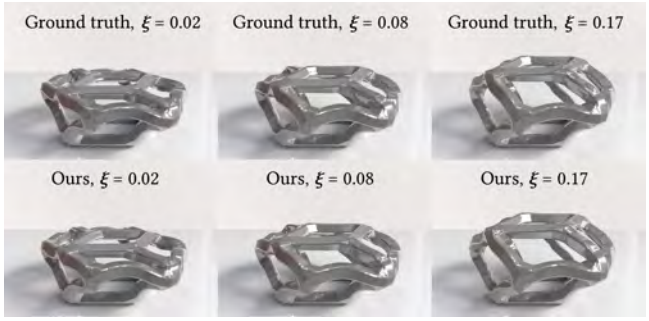

Figure 8: Our neural stress field can capture the hardening effect under different hardening coefficients.

6.3 Metal plasticity

Our neural stress field can also handle history-based plasticity models, such as the effect of hardening [Wang et al. 2019]. The squeezing and bouncing back of a metal frame is simulated with von Mises return mapping under different hardening coefficients $\mu = \tau_Y$. In the ground truth simulation, the yield condition is $y(\boldsymbol{\tau}) < 0$, and thus the return mapping is constantly updated to account for hardening, de facto making the yield condition another path-based state $y(\cdot) = y(\cdot, t)$. Since our neural stress field directly approximates the stress computed *after* the return mapping, such complexity is circumvented. In other words, the hardening state is implicitly learned by our neural stress field $\boldsymbol{h}$. In fig. 8, we compare our deployment results and ground truth under different hardening coefficients.A sampling of $|\mathcal{S}| = 50$ particles out of $|\mathcal{P}| = 49,978$ yields a remarkably small error of $\delta = 0.2\%$ averaging over all testing data, where we choose $r = 5$ and $\gamma = 29.987$. While end-to-end ML frameworks [Sanchez-Gonzalez et al. 2020] can only predict particle positions at rollout time, our PDE-based reduced-order model captures various physical quantities beyond positions. Indeed, our neural stress field can also accurately predict the stress distribution, as is shown in fig. 9 Furthermore, we can sample even fewer points to still obtain reasonably good results. Sampling only 20 particles results in an error of $\delta = 0.5\%$, while sampling merely 30 particles results in an error of $\delta = 0.3\%$, and the results are almost indistinguishable visually compared with the ground truth.

Table 1: Simulation and reduction statistics.

Scene	Figure	Model	Δt	Δx	# of particles	Elasticity/Plasticity	dim($\mathcal{L}$)	MLP size for $\boldsymbol{g}$, $\boldsymbol{h}$, and $\boldsymbol{l}$, respectively	Error
Bread	5	Fixed corotated elasticity	0.001	0.0063	40,000	$E \in [1.0, 13.0]$	$r = 6$	$(5, 48 \cdot 3), (5, 64 \cdot 6), (5, 64 \cdot 9)$	1.2%
Cake	1	von Mises with softening	0.0016	0.0063	200,000	$\tau_y = 0.1, \theta = 0.03$	$r = 6$	$(5, 48 \cdot 3), (5, 64 \cdot 6), (5, 64 \cdot 9)$	1.3%
Sand	7 6	Drucker-Prager	0.002	0.0067	71,363	$\phi_f \in [20°, 40°]$	$r = 6$	$(5, 48 \cdot 3), (5, 72 \cdot 6), (5, 72 \cdot 9)$	0.4%
Metal	8 9	von Mises with hardening	0.0015	0.01	49,978	$\tau_Y = 0.05, \xi \in [0.0, 0.2]$	$r = 5$	$(5, 32 \cdot 3), (5, 48 \cdot 6), (5, 48 \cdot 9)$	0.2-0.5%
Toothpaste	10 11	Herschel-Bulkley	0.001	0.0063	21,811	$\tau_Y = 0.05, \eta = 0.17$	$r = 6$	$(5, 32 \cdot 3), (5, 48 \cdot 6), (5, 48 \cdot 9)$	0.6-1.8%
Jelly cube	12	Fixed corotated elasticity	0.01	0.02	10,000	$E = 1.0$	$r = 5$	$(5, 32 \cdot 3), (5, 64 \cdot 6), (5, 64 \cdot 9)$	0.2%
Squishy ball	13	Fixed corotated elasticity	0.002	0.0067	97,857	$E = 40.0$	$r = 6$	$(5, 48 \cdot 3), (5, 64 \cdot 6), (5, 72 \cdot 9)$	0.2%

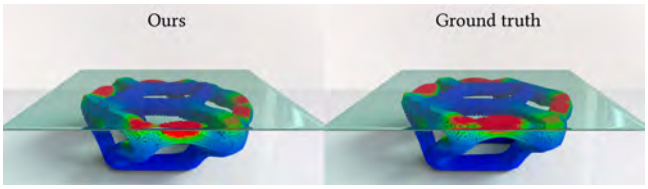

Figure 9: Unlike end-to-end ML frameworks that can only predict particle positions, our first-principal-based reduced-order approach also matches stress quantitatively.

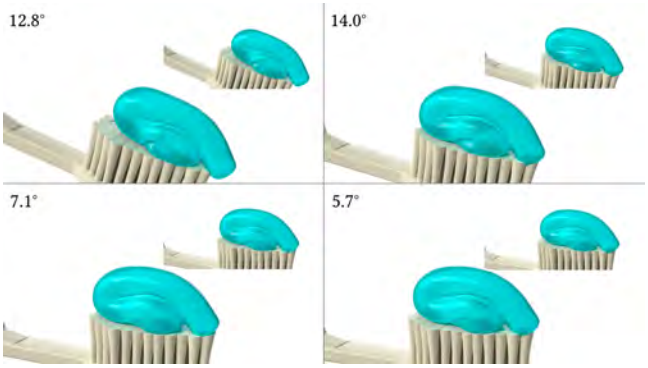

Figure 10: A ribbon of toothpaste is smeared onto a toothbrush held at different angles. The four subplots show our deployment results with 50 sample points. The corresponding ground truth is shown in the top right corner of each subplot.

In addition, with a randomly chosen 30 sample particles, and with the timestep in deployment set to $1.5\Delta t$, we are able to speed up the total wall clock time from 7.83s in the full-order MPM to 1.55s in the reduced model, achieving a speedup more than $5\times$. In this setup, the full-order MPM memory consumption is 0.82G, while ours is 0.51G.

6.4 Non-Newtonian fluids

We simulate a ribbon of toothpaste smeared onto a toothbrush holding at different angles with $|\mathcal{P}| = 45,412$ particles (See fig. 10). Here, the problem parameter represents different boundary conditions, i.e., toothbrush inclination. We choose $r = 6$, and thus $\gamma = 22,706$. We follow the Herschel-Bulkley model in [Yue et al. 2015]. With

Figure 11: Results for different numbers of sampling points are shown. For this example where the toothpaste is held at 7.1°, sampling 15 randomly chosen particles yields an error of $\delta = 1.8\%$, while sampling 30 yields an error of $\delta = 0.9\%$.

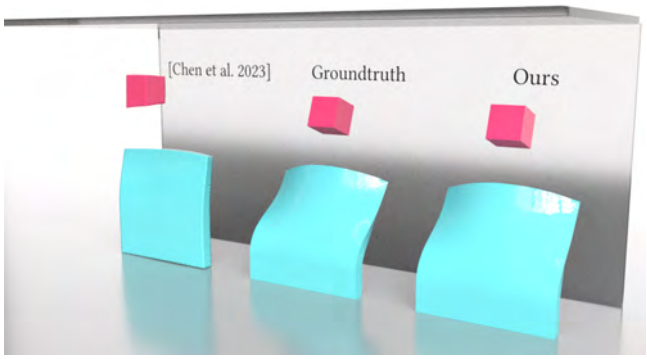

Figure 12: A jelly cube hits onto a jelly wall. Our approach accurately reflects the rotation of the cube. The baseline approach [Chen et al. 2023b] is much more dissipative since it does not support angular momentum. The error δ for our approach is 0.20%, and for the baseline approach is 16.6%. The problem parameter μ represents different initial velocities of the jelly cube.

just 50 sampling points, we can predict the dynamics of toothpaste with an averaging total relative deformation error $\delta = 0.6\%$. Further, the sample size can be even reduced without too much discount on the overall visual quality. As is shown in fig. 11, with only 30 points, the deployment result still looks reasonably good, with an error of $\delta = 0.9\%$.

6.5 Rotation and Collision

We simulate a collision scenario that yields salient rotation (fig. 12) with $|\mathcal{P}| = 10^4$ particles. With a manifold dimension of $r = 5$ and

Figure 13: An elastic squishy ball falls onto an inclined plane. Compared with the baseline [Chen et al. 2023b], ours accurately captures both the self-contact and the rotation. The error δ for our approach is 0.19%, and for the baseline approach is 4.7%. μ represents different inclinations of the plane.

$|\mathcal{S}| = 50$ sample particles, our approach is able to accurately capture the rotational dynamics. The baseline approach, nevertheless, suffers from noticeable artifacts due to its flawed representation of stress and affine fields. Our approach can also phonograph complex contact scenarios (fig. 13). We simulate an elastic squishy ball falling onto an inclined plane with $|\mathcal{P}| = 10^5$ particles. The manifold dimension is set to $r = 6$. A randomly selected set of $|\mathcal{S}| = 300$ sample particles suffices to delineate the contact of tentacles. The baseline method performs poorly as the affine momentum is missing, and the representation of stress is inaccurate in extreme contact. Notice that we do not need to sample all tentacles to capture their motion; rather, a small $\mathcal{S}$ is used to determine $\hat{\boldsymbol{x}}$ in the latent space so that our neural deformation and neural stress field can generate their motion. The error δ for either of the above experiments is less than 0.2%. The dimension reduction ratios are $6,000$ and 5×10^4.

7 DISCUSSIONS AND FUTURE WORK

We proposed Neural Stress Fields (NSF), a novel, reduced-order framework for elastoplastic and fracture simulations. NSF significantly alleviates the computational burden of simulating complex elastoplasticity and fracture effects by training a unified, low-dimensional latent space for the neural deformation, stress, and affine fields. Following the training phase, we efficiently conserve computational resources by leveraging these low-dimensional latent variables for evolution. Our approach sets a compelling precedent for multiple potential research trajectories.

Generalization. Our work supports both interpolation and extrapolation of the training data (see experiments on sand friction angles and bread weak elements). Nevertheless, our approach cannot handle extremely out-of-distribution extrapolation. We trade aggressive generalizability for massive compression and speedup. Future work may consider exploring alternative balancing between generalizability and performance. In addition, for each experiment, we train a network using data from this particular scenario [Sifakis and Barbic 2012]. An exciting future direction is training on one scenario but generalizing to multiple materials and objects.

Training time. Currently, training time is long, between 2hrs and 20hrs. Our target applications are cases where the model would be re-used multiple times. For example, after training, our model can be deployed in VR and gaming applications, where millions of users will interact with it. In these cases, training time is not the main bottleneck. That said, improving training time will help capture larger scenes and accelerate development cycles.

High-frequency neural fields. MPM simulations often involve stress fields with high-frequency details and large spatial variations. In practice, we find it challenging to train neural fields that correctly capture these distributions, preventing us from capturing larger scenes. Future work may consider developing more advanced neural architectures [Sitzmann et al. 2020; Tancik et al. 2022] to improve performance when high-frequency details are presented.

Path-dependent plasticity. Our latent space vector $\hat{\boldsymbol{x}}_t$ is only determined by the position $\boldsymbol{x}_t$. Nevertheless, since plasticity is path-dependent [Borja 2013], the same position field does not imply the same stress field. A potential fix to this issue would be to, instead of training two distinct networks, concatenate $\boldsymbol{x}$ and $\boldsymbol{\tau}$ and train $\boldsymbol{g}(\boldsymbol{X}_p, \boldsymbol{e}([\boldsymbol{x}_n, \boldsymbol{\tau}_n])) \approx [\boldsymbol{x}_p^n, \boldsymbol{\tau}_p^n]$. In addition, to more explicitly enforce history dependency, future work may consider evolving the latent space according to both stress updates and deformation updates.

Data-free training. Sharp et al. [2023] introduces a data-free reduced-order modeling framework by incorporating a physics-informed loss term. Extending it to include MPM's plasticity and fracture phenomena is another exciting direction.

ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation (2153851, 2153863, 2023780), Meta, and Natural Sciences and Engineering Research Council of Canada (Discovery RGPIN-2021-03733). We also express our gratitude to the developers and open-source communities behind PyTorch and NVIDIA Warp.

REFERENCES

Noam Aigerman, Kunal Gupta, Vladimir G Kim, Siddhartha Chaudhuri, Jun Saito, and Thibault Groueix. 2022. Neural Jacobian Fields: Learning Intrinsic Mappings of Arbitrary Meshes. *arXiv preprint arXiv:2205.02904* (2022).

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

Jernej Barbič and Doug L James. 2005. Real-time subspace integration for St. Venant-Kirchhoff deformable models. *ACM transactions on graphics (TOG)* 24, 3 (2005), 982–990.

Jernej Barbič and Yili Zhao. 2011. Real-time large-deformation substructuring. *ACM transactions on graphics (TOG)* 30, 4 (2011), 1–8.

Peter Benner, Serkan Gugercin, and Karen Willcox. 2015. A survey of projection-based model reduction methods for parametric dynamical systems. *SIAM review* 57, 4 (2015), 483–531.

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

Ronaldo I Borja. 2013. *Plasticity*. Vol. 2. Springer.

Jeremiah U Brackbill and Hans M Ruppel. 1986. FLIP: A method for adaptively zoned, particle-in-cell calculations of fluid flows in two dimensions. *Journal of Computational physics* 65, 2 (1986), 314–343.

Kevin Carlberg, Matthew Barone, and Harbir Antil. 2017. Galerkin v. least-csquares Petrov–Galerkin projection in nonlinear model reduction. *J. Comput. Phys.* 330 (2017), 693–734.

Honglin Chen, Rundi Wu, Eitan Grinspun, Changxi Zheng, and Peter Yichen Chen. 2022. Implicit Neural Spatial Representations for Time-dependent PDEs. *arXiv preprint arXiv:2210.00124* (2022).

Peter Yichen Chen, Maytee Chantharayukhonthorn, Yonghao Yue, Eitan Grinspun, and Ken Kamrin. 2021. Hybrid discrete-continuum modeling of shear localization in granular media. *Journal of the Mechanics and Physics of Solids* 153 (2021), 104404.

Peter Yichen Chen, Maurizio M Chiaramonte, Eitan Grinspun, and Kevin Carlberg. 2023a. Model reduction for the material point method via an implicit neural representation of the deformation map. *J. Comput. Phys.* 478 (2023), 111908.

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. 2023b. CROM: Continuous Reduced-Order Modeling of PDEs Using Implicit Neural Representations. In *The Eleventh International Conference on Learning Representations*. https://openreview.net/forum?id=FUORz1tG8Og

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

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. 2015. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289* (2015).

Gilles Daviet and Florence Bertails-Descoubes. 2016. A semi-implicit material point method for the continuum simulation of granular materials. *ACM Transactions on Graphics (TOG)* 35, 4 (2016), 1–13.

Mengyuan Ding, Xuchen Han, Stephanie Wang, Theodore F Gast, and Joseph M Teran. 2019. A thermomechanical material point method for baking and cooking. *ACM Transactions on Graphics (TOG)* 38, 6 (2019), 1–14.

William Falcon. 2019. PyTorch Lightning.

Yu Fang, Minchen Li, Ming Gao, and Chenfanfu Jiang. 2019. Silly rubber: an implicit material point method for simulating non-equilibrated viscoelastic and elastoplastic solids. *ACM Transactions on Graphics (TOG)* 38, 4 (2019), 1–13.

Yun Fei, Christopher Batty, Eitan Grinspun, and Changxi Zheng. 2019. A multi-scale model for coupling strands with shear-dependent liquid. *ACM Transactions on Graphics (TOG)* 38, 6 (2019), 1–20.

Yun Fei, Yuhan Huang, and Ming Gao. 2021. Principles towards Real-Time Simulation of Material Point Method on Modern GPUs. *arXiv preprint arXiv:2111.00699* (2021).

Lawson Fulton, Vismay Modi, David Duvenaud, David IW Levin, and Alec Jacobson. 2019. Latent-space Dynamics for Reduced Deformable Simulation. In *Computer graphics forum*, Vol. 38. Wiley Online Library, 379–391.

Ming Gao, Xinlei Wang, Kui Wu, Andre Pradhana, Eftychios Sifakis, Cem Yuksel, and Chenfanfu Jiang. 2018. GPU optimization of material point methods. *ACM Transactions on Graphics (TOG)* 37, 6 (2018), 1–12.

Oscar Gonzalez and Andrew M Stuart. 2008. *A first course in continuum mechanics*. Vol. 42. Cambridge University Press.

Francis H Harlow. 1962. *The particle-in-cell method for numerical solution of problems in fluid dynamics*. Technical Report. Los Alamos National Lab.(LANL), Los Alamos, NM (United States).

Jan Hegemann, Chenfanfu Jiang, Craig Schroeder, and Joseph M Teran. 2013. A level set method for ductile fracture. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 193–201.

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

Yuanming Hu, Tzu-Mao Li, Luke Anderson, Jonathan Ragan-Kelley, and Frédo Durand. 2019. Taichi: a language for high-performance computation on spatially sparse data structures. *ACM Transactions on Graphics (TOG)* 38, 6 (2019), 1–16.

Chenfanfu Jiang, Craig Schroeder, Andrew Selle, Joseph Teran, and Alexey Stomakhin. 2015. The affine particle-in-cell method. *ACM Transactions on Graphics (TOG)* 34, 4 (2015), 1–10.

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

Byungsoo Kim, Vinicius C Azevedo, Nils Thuerey, Theodore Kim, Markus Gross, and Barbara Solenthaler. 2019. Deep fluids: A generative network for parameterized fluid simulations. In *Computer Graphics Forum*, Vol. 38. Wiley Online Library, 59–70.

Theodore Kim and John Delaney. 2013. Subspace fluid re-simulation. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 1–9.

Theodore Kim and Doug L James. 2009. Skipping steps in deformable simulation with online model reduction. In *ACM SIGGRAPH Asia 2009 papers*. 1–9.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

Gergely Klár, Theodore Gast, Andre Pradhana, Chuyuan Fu, Craig Schroeder, Chenfanfu Jiang, and Joseph Teran. 2016. Drucker-prager elastoplasticity for sand animation. *ACM Transactions on Graphics (TOG)* 35, 4 (2016), 1–12.

Kookjin Lee and Kevin T Carlberg. 2020. Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders. *J. Comput. Phys.* 404 (2020), 108973.

Bethany Lusch, J Nathan Kutz, and Steven L Brunton. 2018. Deep learning for universal linear embeddings of nonlinear dynamics. *Nature communications* 9, 1 (2018), 4950.

Miles Macklin. 2022. Warp: A High-performance Python Framework for GPU Simulation and Graphics. https://github.com/nvidia/warp. NVIDIA GPU Technology Conference (GTC).

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

Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. 2020. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European conference on computer vision*. Springer, 405–421.

Jorge Nocedal and Stephen J Wright. 1999. *Numerical optimization*. Springer.

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

Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. 2019. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 165–174.

Yuxing Qiu, Samuel Temple Reeve, Minchen Li, Yin Yang, Stuart Ryan Slattery, and Chenfanfu Jiang. 2023. A Sparse Distributed Gigascale Resolution Material Point Method. *ACM Transactions on Graphics* 42, 2 (2023), 1–21.

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

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

Nicholas Sharp, Cristian Romero, Alec Jacobson, Etienne Vouga, Paul G Kry, David IW Levin, and Justin Solomon. 2023. Data-Free Learning of Reduced-Order Kinematics. *arXiv preprint arXiv:2305.03846* (2023).

Siyuan Shen, Yin Yang, Tianjia Shao, He Wang, Chenfanfu Jiang, Lei Lan, and Kun Zhou. 2021. High-Order Differentiable Autoencoder for Nonlinear Model Reduction. *ACM Trans. Graph.* 40, 4, Article 68 (jul 2021), 15 pages. https://doi.org/10.1145/3450626.3459754

Eftychios Sifakis and Jernej Barbic. 2012. FEM Simulation of 3D Deformable Solids: A Practitioner's Guide to Theory, Discretization and Model Reduction. In *ACM SIGGRAPH 2012 Courses* (Los Angeles, California) *(SIGGRAPH '12)*. Association for Computing Machinery, New York, NY, USA, Article 20, 50 pages. https://doi.org/10.1145/2343483.2343501

Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. 2020. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems* 33 (2020), 7462–7473.

Alexey Stomakhin, Craig Schroeder, Lawrence Chai, Joseph Teran, and Andrew Selle. 2013. A material point method for snow simulation. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 1–10.

Deborah Sulsky, Shi-Jian Zhou, and Howard L Schreyer. 1995. Application of a particle-in-cell method to solid mechanics. *Computer physics communications* 87, 1-2 (1995), 236–252.

Matthew Tancik, Vincent Casser, Xinchen Yan, Sabeek Pradhan, Ben Mildenhall, Pratul P Srinivasan, Jonathan T Barron, and Henrik Kretzschmar. 2022. Block-nerf: Scalable large scene neural view synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 8248–8258.

Adrien Treuille, Andrew Lewis, and Zoran Popović. 2006. Model reduction for real-time fluids. *ACM Transactions on Graphics (TOG)* 25, 3 (2006), 826–834.

Stephanie Wang. 2020. *A Material Point Method for Elastoplasticity with Ductile Fracture and Frictional Contact*. University of California, Los Angeles.

Stephanie Wang, Mengyuan Ding, Theodore F Gast, Leyi Zhu, Steven Gagniere, Chenfanfu Jiang, and Joseph M Teran. 2019. Simulation and visualization of ductile fracture with the material point method. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 2, 2 (2019), 1–20.

Xinlei Wang, Minchen Li, Yu Fang, Xinxin Zhang, Ming Gao, Min Tang, Danny M Kaufman, and Chenfanfu Jiang. 2020a. Hierarchical optimization time integration for cfl-rate mpm stepping. *ACM Transactions on Graphics (TOG)* 39, 3 (2020), 1–16.

Xinlei Wang, Yuxing Qiu, Stuart R Slattery, Yu Fang, Minchen Li, Song-Chun Zhu, Yixin Zhu, Min Tang, Dinesh Manocha, and Chenfanfu Jiang. 2020b. A massively parallel and scalable multi-GPU material point method. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 30–1.

Steffen Wiewel, Moritz Becher, and Nils Thuerey. 2019. Latent space physics: Towards learning the temporal evolution of fluid flow. In *Computer graphics forum*, Vol. 38. Wiley Online Library, 71–82.

Joshuah Wolper, Yunuo Chen, Minchen Li, Yu Fang, Ziyin Qu, Jiecong Lu, Meggie Cheng, and Chenfanfu Jiang. 2020. AnisoMPM: Animating Anisotropic Damage Mechanics. *ACM Trans. Graph.* 39, 4, Article 37 (2020).

Joshuah Wolper, Yu Fang, Minchen Li, Jiecong Lu, Ming Gao, and Chenfanfu Jiang. 2019. CD-MPM: continuum damage material point methods for dynamic fracture animation. *ACM Transactions on Graphics (TOG)* 38, 4 (2019), 1–15.

Yiheng Xie, Towaki Takikawa, Shunsuke Saito, Or Litany, Shiqin Yan, Numair Khan, Federico Tombari, James Tompkin, Vincent Sitzmann, and Srinath Sridhar. 2021. Neural Fields in Visual Computing and Beyond. *arXiv preprint arXiv:2111.11426* (2021).

Hongyi Xu, Yijing Li, Yong Chen, and Jernej Barbič. 2015. Interactive material design using model reduction. *ACM Transactions on Graphics (TOG)* 34, 2 (2015), 1–14.

Guandao Yang, Serge Belongie, Bharath Hariharan, and Vladlen Koltun. 2021. Geometry Processing with Neural Fields. *Advances in Neural Information Processing Systems* 34 (2021).

Yin Yang, Dingzeyu Li, Weiwei Xu, Yuan Tian, and Changxi Zheng. 2015. Expediting precomputation for reduced deformable simulation. *ACM Trans. Graph* 34, 6 (2015).

Yonghao Yue, Breannan Smith, Christopher Batty, Changxi Zheng, and Eitan Grinspun. 2015. Continuum foam: A material point method for shear-dependent flows. *ACM Transactions on Graphics (TOG)* 34, 5 (2015), 1–20.

Yonghao Yue, Breannan Smith, Peter Yichen Chen, Maytee Chantharayukhonthorn, Ken Kamrin, and Eitan Grinspun. 2018. Hybrid grains: Adaptive coupling of discrete and continuum simulations of granular media. *ACM Transactions on Graphics (TOG)* 37, 6 (2018), 1–19.

Jonas Zehnder, Yue Li, Stelian Coros, and Bernhard Thomaszewski. 2021. Ntopo: Mesh-free topology optimization using implicit neural representations. *Advances in Neural Information Processing Systems* 34 (2021), 10368–10381.

A EXPERIMENT DETAILS

In the bread tearing example, a total of 24 simulations were generated with varying Young's moduli. A random choice of 4 simulations were reserved for testing, while the rest 20 simulations were for training. In the cake-cutting example, the spatula is slicing the cake at different angles. Here $\mathcal{D}_{\text{train}}$ contains 12 simulations and $\mathcal{D}_{\text{test}}$ contains 3 simulations. In the example of sand, 20 simulations for different friction angles, four of which are reserved for testing. In the example of metal, $\mathcal{D}_{\text{train}}$ contains 12 simulations, and $\mathcal{D}_{\text{test}}$ contains 4 simulations. The hardening coefficient ξ is distinct in different simulations. In the example of toothpaste, we also have 12 training simulations and 4 testing simulations. The toothbrush is held at different angles. Finally, the last two examples (jelly cube and squishy ball) both contain 3 training simulations and 1 testing simulation.

B EXTRAPOLATION, GENERALIZATION, AND TRAINING DATA

In this section, we provide more aggressive extrapolation and generalization experiments. To demonstrate the differences between testing data (visualized in the main text) and training data, we also visualize the training data in this section.

In the bread tearing example, the problem parameter is the Young's modulus of the material, and weak elements are inserted to help with fracture. One generalization test is employing unseen weak elements. The results were shown in fig. 14. A mild perturbation on the weak elements (a random perturbation with a scale of 8%) would yield a decent result. The total position error is $\delta = 2.7\%$. On the other hand, if we aggressively perturb the weak elements (a random perturbation with a scale of 30%), the resulting deployment will suffer from significant errors. We observe a 'partially unsuccessful' fracture. The total position error surges to $\delta = 11\%$. In summary, our method cannot capture cases where the fracture pattern is drastically different from the ones shown in the training data.

We also list the training data (fig. 15) in the bread-tearing experiment, all of which have fracture patterns different from the testing data.

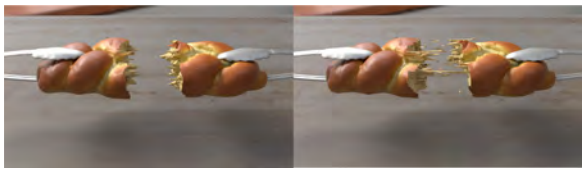

Figure 14: Deployment results with unseen weak elements. Left: A mild perturbation on the weak elements. The simulation remains fairly accurate. Right: A more aggressive perturbation on the weak elements. The simulation suffers from significantly larger errors.

We also performed several extrapolation tests for the sand experiment. In the training data set $\mathcal{D}_{\text{train}}$, the smallest friction angle is 21° and the largest is 38.5°. See c16. The testing data shown in the main text are interpolations where the friction angle lies between

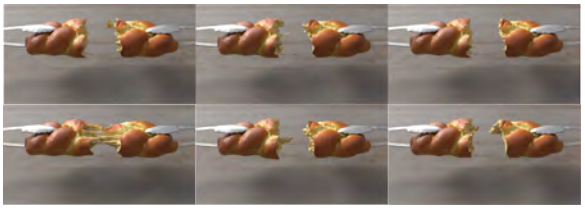

Figure 15: Sample training data of the bread tearing experiment.

21° and 38.5°. Our approach is also robust under reasonable extrapolations. For instance, under a friction angle of 44°, our approach generates accurate results, with a total position error of $\delta = 1.8\%$. However, our method will not work under extreme extrapolation. When we set the friction angle to a significantly larger 55°, our latent space dynamics suffer from very larger errors. The sand column neither keeps its symmetry nor obeys its boundary condition (it penetrates the ground). See 17. Thus, our current pipeline does not support extreme extrapolation that is significantly different from the training data.

Figure 16: Sample training data for the sand example. Left: simulation with the smallest friction angle 21° in the training data set $\mathcal{D}_{\text{train}}$. Right: simulation with the largest friction angle 38.5° in $\mathcal{D}_{\text{train}}$.

Figure 17: Extrapolation for the sand example. Left: the problem parameter θ is set to 44°, where $\mathcal{D}_{\text{train}}$ contains several simulations with $\theta \in [21^\circ, 38.5^\circ]$. The extrapolation yields accurate results, with a total position error of merely $\delta = 1.8\%$. Right: an extreme extrapolation of $\theta = 55^\circ$ is performed. Our approach suffers from larger errors.

C RUNTIME ANALYSIS

The timing experiments are performed on a machine with Intel Core i7-8700K and NVIDIA Quadro P6000. Both the full-order and the reduced MPM simulator are implemented in WARP [Macklin 2022]. The neural networks were implemented in PyTorch. The runtime reported is an average of ten repeated trials.

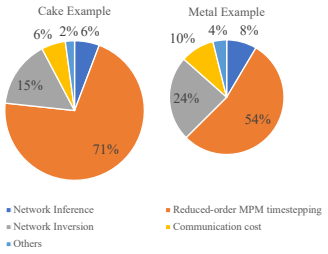

Figure 18: Pie-chart breakdown for the runtime of each component in our reduced-order pipeline for two examples: the cake and the metal. The pie chart for the metal example is plotted smaller to show that this is a smaller problem and thus has a shorter runtime. Overall, the main bottleneck of the reduced-order algorithm remains to be MPM timestepping while the network operations introduce little overhead.

In the example of cake, the runtime of full-order MPM is $4.53s/100$ frames. The runtime of the reduced-order MPM is $0.469s/100$ frames. The cost in network inference, projection, as well as communication between WARP and PyTorch is $0.195/100$ frames.

In the example of metal, the runtime of full-order MPM is $1.305s/100$ frames. The runtime of the reduced-order MPM is $0.209s/100$ frames. The cost in network inference, projection, as well as communication between WARP and PyTorch is $0.178/100$ frames.

We also provide two pie charts for the breakdown of the runtime of each component in our reduced-order scheme for the two examples. The runtime for network-related operations grows relatively mildly when problem size increases, whereas the runtime for MPM timestepping in theory grows linearly in problem size (see next paragraph for a discussion). Thus, problems with larger scales tend to enjoy more time savings from our algorithm.

For a paralleled explicit MPM simulator, one can usually observe that the runtime is roughly linearly proportional to the total number of particles. The two majority of costs come from the SVD computation for stress and the atomic addition in P2G. In our experiments, reduced MPM with about 9% of original particles costs about 16% of original runtime, and with 6.7% of original particles costs about 10.3% of original runtime. In theory, this reduced MPM runtime should be even smaller. In addition, SVD is not required in the reduced MPM. We reason that this is because, during deployment, PyTorch has already occupied some memory. Thus, the computing power allocated for the reduced MPM solver in WARP would be less than that allocated for a full-order MPM solver alone.

D TRAINING DETAILS

The training data is generated from an MPM solver written in WARP [Macklin 2022] under double precision. The Adam optimizer [Kingma and Ba 2014] for stochastic gradient descent is used for training. The Xavier initialization is used for the ELU layers. We fix the learning rates to be $(10^{-3}, 5\times10^{-4}, 2\times10^{-4}, 10^{-4}, 5\times10^{-5})$. For the Neural Deformation Field, 300 epochs are trained for the learning rate above. For the Neural Stress Field and Neural Affine Field, 600 epochs are trained for the learning rate above. The batch size for the three manifolds is $k\cdot|\mathcal{P}|$ where we choose k to be the largest value such that the training data fits within memory or 32, whichever is smaller. The training data is normalized to have mean zero and variance one. The whole training pipeline is implemented in Pytorch Lightning [Falcon 2019].

E NETWORK ARCHITECTURE

The (input, output) dimension pairs for $\boldsymbol{g}, \boldsymbol{h}$ and $\boldsymbol{l}$ are $(3+r, 3)$, $(3+r, 6)$, and $(3+r, 9)$, respectively. The Kirchhoff stress $\boldsymbol{\tau}$ has 6 degrees of freedom since it is symmetric. Each MLP network contains 5 hidden layers, each of which has a width of $\beta\cdot d$, where $d=3$ for $\boldsymbol{g}$, $d=6$ for $\boldsymbol{h}$, and $d=9$ for $\boldsymbol{l}$. β is a hyperparameter, the exact value of which for each experiment is listed in Table 1 in the main text. We adopt the ELU activation function [Clevert et al. 2015]. The encoder network is devised as the following: several 1D convolution layers of kernel size 6, stride size 4, and output channel size 3 are applied until the length of the 1D output vector reaches or below 12. The vector is then reshaped to 1 channel. One MLP layer transforms its dimension to 32, followed by the last MLP layer that outputs a vector with dimension r.

F ELASTICITY AND PLASTICITY DETAILS

We first list all parameters that shall be needed in discussing the models below.

Notation	Meaning	Relation to (E,ν)
E	Young's modulus	/
ν	Poisson's ratio	/
$\hat{\mu}$	Shear modulus	$\hat{\mu}=\frac{E}{2(1+\nu)}$
λ	Lamé modulus	$\lambda=\frac{E\nu}{(1+\nu)(1-2\nu)}$
κ	Bulk modulus	$\kappa=\frac{E}{3(1-2\nu)}$

In all plasticity models used in our work, the deformation gradient is multiplicatively decomposed into $\boldsymbol{F}=\boldsymbol{F}^E\boldsymbol{F}^P$ following some yield stress condition. A hyperelastic constitutive model is applied to $\boldsymbol{F}^E$ to compute the Kirchhoff stress $\boldsymbol{\tau}$. For a pure elastic continuum, one simply takes $\boldsymbol{F}^E=\boldsymbol{F}$.

F.1 Fixed corotated elasticity

The Kirchhoff stress $\boldsymbol{\tau}$ is defined as

$$\boldsymbol{\tau}=2\hat{\mu}(\boldsymbol{F}^E-\boldsymbol{R})\boldsymbol{F}^{E^T}+\lambda(J-1)J, \tag{12}$$

where $\boldsymbol{R}=\boldsymbol{U}\boldsymbol{V}^T$ and $\boldsymbol{F}^E=\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$ is the singular value decomposition of elastic deformation gradient. [Jiang et al. 2015]

F.2 StVK elasticity

The Kirchhoff stress $\boldsymbol{\tau}$ is defined as

$$\boldsymbol{\tau}=\boldsymbol{U}\left(2\hat{\mu}\boldsymbol{\epsilon}+\lambda\,\mathrm{sum}(\boldsymbol{\epsilon})\mathbf{1}\right)\boldsymbol{V}^T, \tag{13}$$

where $\boldsymbol{\epsilon}=\log(\Sigma)$ and $\boldsymbol{F}^E=\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$. [Klár et al. 2016]

F.3 Drucker-Prager plasticity

The return mapping of Drucker-Prager plasticity for sand [Klár et al. 2016] is, given $\boldsymbol{F}=\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$ and $\boldsymbol{\epsilon}=\log(\Sigma)$,

$$\boldsymbol{F}^E=\boldsymbol{U}\mathcal{Z}(\Sigma)\boldsymbol{V}^T.$$

$$\mathcal{Z}(\Sigma) = \begin{cases} \mathbf{1} & \mathrm{sum}(\boldsymbol{\epsilon}) > 0 \\ \Sigma & \delta\gamma \le 0, \text{ and } \mathrm{sum}(\boldsymbol{\epsilon}) \le 0 \\ \exp\left(\boldsymbol{\epsilon} - \delta\gamma \frac{\hat{\epsilon}}{\|\hat{\epsilon}\|}\right) & \text{otherwise} \end{cases}$$

Here $\delta\gamma = \|\hat{\boldsymbol{\epsilon}}\| + \alpha \frac{(d\lambda + 2\hat{\mu})\,\mathrm{sum}(\boldsymbol{\epsilon})}{2\hat{\mu}}$, $\alpha = \sqrt{\frac{2}{3}} \frac{2\sin\phi_f}{3-\sin\phi_f}$, and ϕ_f is the friction angle. $\hat{\epsilon} = \mathrm{dev}(\epsilon)$

F.4 von Mises plasticity

Given $\boldsymbol{F} = \boldsymbol{U}\Sigma\boldsymbol{V}^T$ and $\boldsymbol{\epsilon} = \log(\Sigma)$,

$$\boldsymbol{F}^E = \boldsymbol{U}\mathcal{Z}(\Sigma)\boldsymbol{V}^T,$$

where

$$\mathcal{Z}(\Sigma) = \begin{cases} \Sigma, & \left\|\boldsymbol{\tau} - \frac{1}{d}\,\mathrm{sum}(\boldsymbol{\tau})\right\| - \tau_y \le 0 \\ \exp\left(\boldsymbol{\epsilon} - \delta\gamma \frac{\hat{\epsilon}}{\|\boldsymbol{\epsilon}\|}\right), & \text{Otherwise} \end{cases},$$

and $\delta\gamma = \|\hat{\boldsymbol{\epsilon}}\|_F - \frac{\tau_y}{2\hat{\mu}}$. Here τ_Y is the yield stress. If hardening is included, the yield stress is updated as $\tau_Y^{n+1} = \tau_Y^n + 2\hat{\mu}\xi\delta\gamma$, where ξ is the hardening coefficient. If softening is included, yield stress is updated as $\tau_Y^{n+1} = \tau_Y^n - \theta\|\boldsymbol{\epsilon} - \mathrm{proj}(\boldsymbol{\epsilon})\|_F$. When τ_Y reaches zero, the material is considered damage and its Lamé parameters are set to zero. [Wang et al. 2019]

F.5 Herschel-Bulkley plasticity

We follow [Yue et al. 2015] and take the simple case where $h = 1$. Denote $\boldsymbol{s}^{\mathrm{trial}} = \mathrm{dev}(\boldsymbol{\tau}^{\mathrm{trial}})$, and $s^{\mathrm{trial}} = ||\boldsymbol{s}^{\mathrm{trial}}||$. The yield condition is $\Phi(s) = s - \sqrt{\frac{2}{3}}\sigma_Y \le 0$. If it is violated, we modify s^{trial} by

$$s = s^{\mathrm{trial}} - \left(s^{\mathrm{trial}} - \sqrt{\frac{2}{3}}\sigma_Y\right) \Big/ \left(1 + \frac{\eta}{2\hat{\mu}\Delta t}\right).$$

$\boldsymbol{s}$ can then be recovered as $\boldsymbol{s} = s \cdot \frac{\boldsymbol{s}^{\mathrm{trial}}}{||\boldsymbol{s}^{\mathrm{trial}}||}$. Define $\boldsymbol{b}^E = \boldsymbol{F}^E\boldsymbol{F}^{E^T}$. The Kirchhoff stress $\boldsymbol{\tau}$ is computed as

$$\boldsymbol{\tau} = \frac{\kappa}{2}\left(J^2 - 1\right)\boldsymbol{I} + \hat{\mu}\,\mathrm{dev}\left[\det(\boldsymbol{b}^E)^{-\frac{1}{3}}\boldsymbol{b}^E\right].$$