Diffusion-Based Hierarchical Graph Neural Networks for Simulating Nonlinear Solid Mechanics

Tobias Würth[1*] **Niklas Freymuth**[2] **Gerhard Neumann**[2] **Luise Kärger**[1]
[1]Institute of Vehicle System Technology, Karlsruhe Institute of Technology, Karlsruhe
[2]Autonomous Learning Robots, Karlsruhe Institute of Technology, Karlsruhe

Abstract

Graph-based learned simulators have emerged as a promising approach for simulating physical systems on unstructured meshes, offering speed and generalization across diverse geometries. However, they often struggle with capturing global phenomena, such as bending or long-range correlations usually occurring in solid mechanics, and suffer from error accumulation over long rollouts due to their reliance on local message passing and direct next-step prediction. We address these limitations by introducing the Rolling Diffusion-Batched Inference Network (ROBIN), a novel learned simulator that integrates two key innovations: (i) Rolling Diffusion-Batched Inference (ROBI), a parallelized inference scheme that amortizes the cost of diffusion-based refinement across physical time steps by overlapping denoising steps across a temporal window. (ii) A Hierarchical Graph Neural Network built on algebraic multigrid coarsening, enabling multiscale message passing across different mesh resolutions. This architecture, implemented via Algebraic-hierarchical Message Passing Networks, captures both fine-scale local dynamics and global structural effects critical for phenomena like beam bending or multi-body contact. We validate ROBIN on challenging 2D and 3D solid mechanics benchmarks involving geometric, material, and contact nonlinearities. ROBIN achieves state-of-the-art accuracy on all tasks, substantially outperforming existing next-step learned simulators while reducing inference time by up to an order of magnitude compared to standard diffusion simulators.

1 Introduction

Physical simulations enable many engineering and scientific fields to gain quick insights into complex systems or to evaluate design decisions. Conventional simulations model the physical system using Partial Differential Equations (PDEs). Usually, the PDE is discretized by numerical methods, such as the Finite Element Method (FEM) [1], the Finite Volume Method (FVM) [2], or the Finite Difference Method (FDM) [3]. This process reduces the need for cumbersome, resource-intensive real-world experiments. Recent research aims to speed up simulation with Machine Learning (ML)-based models [4, 5]. These learned simulators promise to allow researchers and practitioners to evaluate large amounts of virtual, simulated experiments. These simulations in turn unlock applications in engineering design and manufacturing optimization [6–8].

This work aims to improve learned simulators, focusing on simulations of nonlinear solid mechanics as a representative class of examples. We combine recent image-based Denoising Diffusion Probabilistic Models (DDPMs) [9–11] with Hierarchical Graph Neural Networks (HGNNs) [12–14] (cf. Figure 1). While diffusion has shown promising results on images [15–17], audio [18, 19] and even policy learning for robotics [20, 21], it suffers from cost-intensive inference due to its iterative denoising procedure. We alleviate this high inference cost on time-dependent domains with Rolling Diffusion-

*correspondence to tobias.wuerth@kit.edu

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

Batched Inference (ROBI), a novel scheduling scheme that batches denoising steps of consecutive time steps. ROBI already starts denoising future prediction steps by using partially refined previous steps. This process reduces the number of model evaluations to the number of time steps and preserves the time-shift equivariance of Markovian systems. ROBI only affects the inference process, allowing us to utilize conventional, parallelized DDPM training.

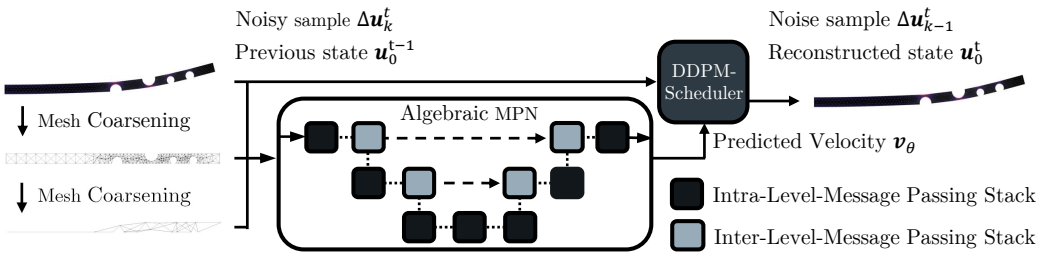


Figure 1: Overview of a Rolling Diffusion-Batched Inference Network (ROBIN) prediction. ROBIN coarsens the fine mesh multiple times with Algebraic multigrid (AMG) to create a graph hierarchy. ROBIN predicts the denoising velocity $\mathbf{v}_\theta(\Delta\mathbf{u}_k^t, k, \mathbf{u}_0^{t-1})$ at time step t using Algebraic-hierarchical Message Passing Networks (AMPNs), given a noisy residual sample $\Delta\mathbf{u}_k^t$, the diffusion step k, and a previous state $\mathbf{u}_0^{t-1}$. The prediction is used to draw a new noisy residual sample $\Delta\mathbf{u}_{k-1}^t$ and to update the state $\mathbf{u}_0^t$.

We combine ROBI with DDPMs and Algebraic-hierarchical Message Passing Networks (AMPNs) to form the Rolling Diffusion-Batched Inference Network (ROBIN), which significantly accelerates simulation while improving predictive accuracy. ROBIN constitutes the first diffusion-based refiner for simulating physical dynamics on unstructured meshes, surpassing the current accuracy ceiling of HGNNs. We train ROBINs on three challenging 2D and 3D solid mechanics datasets involving geometric, material, and contact nonlinearities. Across all datasets, ROBIN significantly improves over state-of-the-art mesh-based simulators [5, 22] in terms of predictive accuracy. Leveraging DDPMs, ROBIN accurately captures low-frequency global solution modes while resolving high-frequency components. We further find that our proposed inference method, ROBI, speeds up diffusion-based inference for learned simulations by up to an order of magnitude while maintaining accuracy. In addition, the AMPN architecture of ROBIN enables efficient transfer to much larger meshes, while maintaining near-FEM accuracy.[2]

To summarize, we i) propose ROBI, a novel inference scheduling scheme for diffusion-based simulators that amortizes denoising across time steps, reducing inference to a single model evaluation per step while preserving time-shift equivariance; ii) introduce ROBIN, a diffusion-based HGNN for nonlinear solid mechanics that combines multiscale message passing with ROBI to provide fast, accurate diffusion-based simulations; iii) demonstrate state-of-the-art performance on challenging 2D and 3D benchmarks, outperforming existing simulators in both accuracy and runtime.

2 Related Work

Simulating Complex Physics. Simulating complex physical systems often requires numerical solvers, such as the FEM [23, 24, 1], the FVM [2], or the FDM [3]. While accurate, numerical solvers scale poorly with problem complexity, often requiring multiple hours or even days for a single rollout on a modern workstation. Recent work shows that ML-based models are able to learn such numerical simulations from data [25, 26, 4, 5, 27, 9, 22, 10]. ML-based models provide speed-ups of one to two orders of magnitude while being fully differentiable, which accelerates downstream applications such as design [6] or manufacturing process optimization [7, 8]. Many learned simulators operate autoregressively. They mimic numerical solvers by using their own predictions to estimate the residual between successive time steps [4, 5, 28]. Similarly, Neural ODEs [29, 30] predict time derivatives and advance solutions via numerical integration. In contrast to these supervised approaches, Physics-Informed Neural Networks [31] directly operate on a PDE loss function to train Multilayer Perceptrons (MLPs) [32, 33, 8], Convolutional Neural Networks (CNNs) [34, 35] or Graph Neural Networks (GNNs) [36, 37]. Finally, Neural Operators aim to learn mesh-independent solution operators [38–40, 27, 41]. ROBIN is an autoregressive learned simulator that replaces direct

[2]Project page, code and datasets are available at https://tbswrth.github.io/ROBIN.

next-step prediction with multiple denoising diffusion steps, leveraging generative inference to improve prediction accuracy.

Learning Mesh-based Simulations with Graph Neural Network (GNN). Pfaff et al. [5] introduced MESHGRAPHNETS, a Message Passing Network (MPN)-based simulator that encodes simulation states as graphs using mesh connectivity and physical proximity. While accurate for small problems, MESHGRAPHNETS (MGNs) does not scale well, as its receptive field is limited by the number of message-passing steps in the MPN. To address this issue, recent work expands the receptive field via global attention [42–44] or hierarchical mesh representations using Graph Convolutional Networks [45] and MPNs [46, 13]. Extensions further improve efficiency and accuracy through rotation equivariance [14], hierarchical edge design [47], bi-stride pooling [48], and attention mechanisms acting on edges and across hierarchies [22]. [49] leverages Adaptive Mesh Refinement (AMR) to create mesh hierarchies for multi-scale GNNs. Complementary, Physics-Informed MESHGRAPHNETS [37] integrate FEM-based training to improve accuracy and robustness. Existing methods are generally trained using a next-step Mean Squared Error (MSE) loss, which favors learning lower solution frequencies at the cost of accuracy in higher frequency bands that have less impact on the loss [9]. However, autoregressive models trained with a MSE loss often overlook low-amplitude frequencies [9]. Our approach is orthogonal, coupling hierarchical GNN with denoising diffusion models. This approach takes advantage of the large receptive field of multi-scale GNNs while pushing accuracy toward diffusion limits.

Diffusion-based Simulations. Diffusion models have been applied to physics-informed image super-resolution [35], flow field reconstruction [50], and steady-state flow generation on grids using CNNs [51], and more recently to meshes with hierarchical GNNs [52]. These models, however, operate on isolated frames and do not capture time-dependent dynamics. In contrast, our model predicts deterministic physical evolution rather than equilibrium samples via autoregressive rollouts.

While next-step simulators trained with MSE loss capture high-amplitude, low-frequency components, they often miss low-amplitude components [9]. PDERefs (PDE-Refiners) address this via iterative refinement, improving long-horizon accuracy of grid-based CNN simulators. We extend this idea to unstructured meshes by combining algebraic mesh coarsening [53, 54] with hierarchical GNNs [52, 22] with shared layers. We further introduce a time-parallel denoising scheme at inference, removing the speed bottleneck while maintaining accuracy. Unlike diffusion-based CNN simulators that require K model evaluations per physical step [11, 55, 9], our method requires only a single hierarchical GNN call per step post warm-up, reducing inference costs over T time steps from $\mathcal{O}(KT)$ to $\mathcal{O}(T)$. Compared to video-based approaches [10, 56], which need to jointly process N steps and learn a time-dependent denoiser with high memory cost, our model, ROBIN, leverages time-translation invariance to train a time-independent denoiser with only one time step in memory. As such, ROBIN can be applied autoregressively and can freely interpolate between fully parallel denoising and memory-efficient one-step denoising at inference time. It also predicts state residuals instead of states, significantly improving long-horizon rollout fidelity.

3 Rolling Diffusion-Batched Inference Network (ROBIN)

Graph Network Simulators (GNSs) for Mesh-based Simulations. We consider solving PDEs for physical quantities $\mathbf{u}(\mathbf{x}, t)$ that change over time $t \in [0, T]$ and inside a time-dependent domain $\mathbf{x}(t) \in \Omega(t)$. We focus on simulations on meshes, where $\mathcal{G} = (\mathcal{V}, \mathcal{E}^{\mathrm{M}})$ denotes the mesh graph and the graph nodes $\mathcal{V}$ and the graph edges $\mathcal{E}^{\mathrm{M}}$ correspond to mesh nodes and mesh edges. We seek solutions $\mathbf{u}_i(t) = \mathbf{u}(\mathbf{x}_i, t)$ at discrete node locations $\mathbf{x}_i(t) \in \Omega(t)$. To obtain discretized PDEs, usually numerical methods, such as the FEM, are applied that define the discretization of spatial operators, such as gradients $\partial \mathbf{u}(\mathbf{x}, t)/\partial \mathbf{x}$. Given the discretized operator $\mathcal{F}$, the PDE simplifies to a time-dependent Ordinary Differential Equation and requires solving $\partial \mathbf{u}_i/\partial t = \mathcal{F}(t, \mathbf{x}_i, \mathbf{u}_i)$. We can solve such systems using numerical time discretization schemes. In this work we use a simple *Euler* forward discretization $\mathbf{u}_i^{t+1} = \mathbf{u}_i^t + \Delta t\, \mathcal{F}(t, \mathbf{x}_i^t, \mathbf{u}_i^t)$, and set $\Delta t = 1$. We extend PDE-Refiner [9] to *Lagrangian* systems, where the domain Ω^t and node locations $\mathbf{x}_i^t \in \Omega^t$ evolve over time. Here, we predict the solution $\mathbf{u}_i^t$ at time step t by learning to reverse a probabilistic diffusion process [57] conditioned on the previous state of time step $t - 1$. The proposed methods also apply without any restriction to *Eulerian* systems, where the domains are time-independent.

3.1 Denoising Diffusion Probabilistic Models (DDPMs) for time-dependent simulations

Given a time-dependent solution $\mathbf{u}^t$ from a data distribution $q(\mathbf{u})$, the forward diffusion process is modeled by a Markov chain, where Gaussian noise is added gradually to the sample $\mathbf{u}_k^t$ at each diffusion step k

$$q(\mathbf{u}_{1:K}^t|\mathbf{u}_0^t) = \prod_{k=1}^{K} q(\mathbf{u}_k^t|\mathbf{u}_{k-1}^t) \,, \quad q(\mathbf{u}_k^t|\mathbf{u}_{k-1}^t) := \mathcal{N}(\mathbf{u}_k^t; \sqrt{1-\beta_k}\mathbf{u}_{k-1}^t, \beta_k \mathbf{I}) \,.$$

$\mathcal{N}$ denotes the normal distribution and β_k the noise specified by a variance scheduler. We learn to reverse the diffusion process, i.e.,

$$p_\theta(\mathbf{u}_{k-1}^t|\mathbf{u}_k^t) := \mathcal{N}(\mathbf{u}_{k-1}^t; \mu_\theta(\mathbf{u}_k^t, k), \Sigma(\mathbf{u}_k^t, k)) \,,$$

consisting of K iterative diffusion steps and starting from $k = K$. The mean μ_θ depends on the prediction of a learned model with parameters θ. The covariance Σ is assumed to be isotropic and given as $\Sigma = \sigma_k^2 \mathbf{I} = (\frac{1-\bar{\alpha}_{k-1}}{1-\bar{\alpha}_k}\beta_k)\mathbf{I}$ [57] with $\alpha_i = 1 - \beta_i$ and $\bar{\alpha}_k = \prod_{i=1}^{k} \alpha_i$. We train the model to predict the denoising velocity, i.e., the *v-prediction target*

$$\mathbf{v}_k^t = \sqrt{\bar{\alpha}_k}\epsilon^t - \sqrt{1-\bar{\alpha}_k}\mathbf{u}_0^t \,, \tag{1}$$

given gaussian noise ϵ^t [58]. Since this target smoothly interpolates between $-\mathbf{u}_0^t$ as $\bar{\alpha}_k \approx 0$ and $\boldsymbol{\epsilon}^t$ ($\bar{\alpha}_k \approx 1$), it emphasizes predicting the clean sample in early (high-noise) steps and the noise in later (high-signal) steps, simplifying learning. The model predicts the denoising velocity $\mathbf{v}_\theta(\mathbf{u}_k^t, k) = \mathbf{v}_\theta(\mathbf{u}_k^t, k, \mathbf{u}_0^{t-1})$ autoregressively, conditioned the model on the last time step solution $\mathbf{u}_0^{t-1}$. The training objective is then defined as $\mathbb{E}_{\mathbf{u}_0^t, \epsilon^t, k}\left[\left\|\mathbf{v}_\theta(\mathbf{u}_k^t, k, \mathbf{u}_0^{t-1}) - \mathbf{v}_k^t\right\|^2\right]$ [9]. To facilitate faster denoising, we follow the DDPM formulation of [9] and use an exponential β_k scheduler.

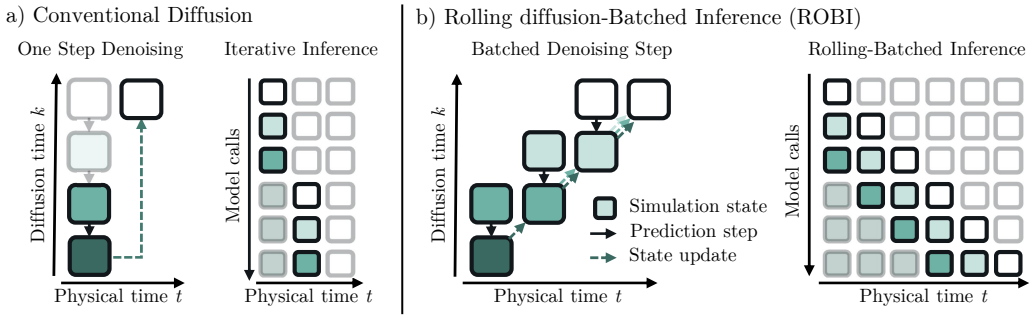


Figure 2: Overview of **a)** conventional autoregressive diffusion inference and **b)** ROBI. **a)** Conventional inference denoises the entire state of a physical time step at once before it shifts to the next time step (see *One Step Denoising*). The *Iterative Inference* requires K model calls per time step, where K denotes the number of diffusion steps. **b)** ROBI parallelizes denoising steps across physical time, processing up to K time steps batched, and reconstruct the physical states with the clean sample prediction subsequently (see *Batched Denoising Step*). This process allows *Rolling-Batched Inference* after the initial warm-up, reducing the number of model calls to one per time step.

The first denoising step is defined such that $\bar{\alpha}_K \approx 0$, which simplifies the v-prediction target to $\mathbf{v}_k^t \approx -\mathbf{u}_0^t$ (cf. Equation (1)). The noisy sample $\mathbf{u}_k^t \approx \epsilon^t$ corresponds to Gaussian Noise and is uninformative. Hence, for $k = K$ the model target converges to $\left\|\mathbf{v}_\theta(\mathbf{u}_K^t, K, \mathbf{u}_0^{t-1})) - \mathbf{v}_k^t\right\|^2 \approx \left\|\mathbf{v}_\theta(\epsilon^t, K, \mathbf{u}_0^{t-1})) + \mathbf{u}_0^t\right\|^2$. This MSE objective of the first denoising step $k = K$ mirrors the training objective of one-step models [9], i.e., auto-regressive models that predict the solution of the next time directly with a single prediction step. Hence, we expect similar accuracy as one-step models during the first diffusion steps. However, the DDPM-based model refines the initial prediction iteratively at each time step, improving the accuracy of the solution further. Note that, due to the noise scheduler,

each denoising step focuses on different amplitude and frequency levels of the solution [9], with later denoising steps increasingly paying attention to higher frequencies.

Rolling Diffusion-Batched Inference (ROBI). Conventional diffusion inference requires K model calls, each corresponding to a denoising step, per simulation time step [9]. Figure 2 a) shows an example. Thus, inference is roughly K times slower than one-step models [5]. We propose Rolling Diffusion-Batched Inference (ROBI) to accelerate inference in DDPM-based autoregressive simulators. Given a velocity prediction $\mathbf{v}_\theta(\mathbf{u}_k^t, k, \mathbf{u}_0^{t-1})$ at the denoising step k, we reconstruct a partially refined prediction $\tilde{\mathbf{u}}_{0|k}^t = \sqrt{\bar{\alpha}_k}\mathbf{u}_k^t - \sqrt{1-\bar{\alpha}_k}\mathbf{v}_\theta(\mathbf{u}_k^t, k, \mathbf{u}_0^{t-1})$, following [58].

As discussed in Section 3.1, early denoising steps behave similarly to one-step model predictions, while later steps progressively refine higher spatial frequencies (from coarse structures to fine details). After $m < K$ denoising steps at time t, the intermediate estimate $\tilde{\mathbf{u}}_{0|m}^t$ already captures low-to-mid frequency content that is sufficient to condition the next physical step $t{+}1$ to predict a solution within this already refined frequency band.

ROBI exploits this property by starting the denoising of step $t{+}1$ as soon as step t has progressed by m steps. We initialize $\mathbf{u}_K^{t+1} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and denoise the batch of both time steps $t_w{\in}\{t, t+1\}$. More generally, after jm denoising steps, we initialize the time step $t{+}j$ and denoise a rolling time window with $t_w{\in}\{t, ..., t+j\}$ in parallel. After a short warm-up, fully denoised samples (those with $k{=}0$) drop out and the window size becomes constant with $w{=}K/m$ and $t_w{\in}\{t, ..., t{+}w{-}1\}$. Notably, the denoising index k and thus the noise level increase along the window toward later physical times, aligning with the natural growth of forecast uncertainty. This reduces the number of model calls from KT (conventional inference) to $K{-}m{+}mT$ steps. For $m{=}1$, the number of model calls reduces to $K{+}T$, which is effectively the same complexity $\mathcal{O}(T)$ as for one-step models, when $K \ll T$. We refer to m as the *denoising stride*. Figure 2 b) visualizes the special case $m{=}1$ and $K{=}3$ diffusion steps. Each model call advances the simulation by one physical step and applies one denoising step to each of the K partially denoised predictions in the window.

A single *Batched Denoising Step* (cf. Figure 2 b) left for $m{=}1$ and $K{=}3$) of ROBI consists of two consecutive steps. In the *Prediction step*, the model outputs the velocity $\mathbf{v}_\theta(\mathbf{u}_{k_w}^{t_w}, k_w, \mathbf{u}_0^{t_w-1})$ for each element in the prediction window $t_w{\in}\{t, ..., t{+}w{-}1\}$, with the diffusion indices $k_w = {\in}\{m, ..., K\}$. Subsequently, we perform a *State Update* step. Using the scheduler, we reconstruct $\tilde{\mathbf{u}}_{0|k_w}^{t_w}$ to update the conditioning states $\mathbf{u}_0^{t_w}$, and sample the reverse transition $\mathbf{u}_{k_w-1}^{t_w} \sim p_\theta(\mathbf{u}_{k_w-1}^{t_w} | \mathbf{u}_{k_w}^{t_w})$. Both are used in the next *Prediction Step* as inputs. After m such *Batched Denoising Steps*, we advance in time and drop the fully denoised states ($k{=}0$) and initialize a new Gaussian sample ($k{=}K$) for the next time index $t{+}w$, so the subsequent call evaluates $\mathbf{v}_\theta(\mathbf{u}_{k_w}^{t_w+1}, k_w, \mathbf{u}_0^{t_w})$.

For $m{=}K$, ROBI reduces to conventional autoregressive diffusion inference (cf. Figure 2 a)). Thus, the denoising stride m can be considered a hyperparameter that trades off prediction accuracy and memory usage with inference speed. Most notably, in *State Update*, ROBI reconstructs the physical states, while the rolling time window is treated purely as a batch dimension in the prediction model. Consequently, the model always predicts $\mathbf{v}_\theta(\mathbf{u}_k^t, k) = \mathbf{v}_\theta(\mathbf{u}_k^t, k, \mathbf{u}_0^{t-1})$, i.e., conditioned only on the previous reconstructed state during both training and inference, which proves to be more stable and accurate for autoregressive ML-based simulations [5, 9]. Furthermore, this preserves the time-shift equivariance of Markovian systems, fast training convergence and small GPU memory utilization.

In practice, we find predicting residuals $\Delta\mathbf{u}_k^t$ improves accuracy. Let the model denoise a batch of residual states $\Delta\tilde{\mathbf{u}}_0^{t_w} \approx \Delta\tilde{\mathbf{u}}_{0|k_w}^{t_w}$ of the time window $t_w{\in}\{t, ..., t{+}w{-}1\}$ and denote the last fully denoised state as $\mathbf{u}_0^{t-1}$. We then recover clean states inside the window via a cumulative sum $\tilde{\mathbf{u}}_0^{t+j} = \mathbf{u}_0^{t-1} + \sum_{i=0}^{j} \Delta\tilde{\mathbf{u}}_0^{t+i},\ j \in \{0, ..., w-1\}$. To further accelerate inference, we optionally stop denoising early at a *truncation step* k_{tr} and use the partially denoised state $\tilde{\mathbf{u}}_{0|k_{\text{tr}}}^t$ as the final prediction of the time step. This remains effective because early denoising steps already approximate one-step prediction.

3.2 Denoising Diffusion Probabilistic Models (DDPMs) for mesh-based simulations

To fully utilize Denoising Diffusion Probabilistic Models (DDPMs)'s potential for generating rich, multi-frequency solutions, prediction models must handle multi-scale information. Hierarchical Graph Neural Networks (HGNNs) are particularly well-suited for this, as their architecture inherently

learns representations at varying levels of granularity, mirroring the diverse frequency content present in DDPM outputs. Leveraging this idea, we train HGNN to predict the discrete denoising velocity $\mathbf{v}_{i,\theta}(\mathbf{u}_{i,k}^t, k, \mathbf{u}_{i,0}^{t-1})$ for mesh-based simulations on the mesh graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}^{\mathrm{M}})$. It takes the current noisy sample $\mathbf{u}_{i,k}^t$ and is conditioned on the previous clean sample $\mathbf{u}_{i,0}^{t-1}$.

Root-node AMG-based Mesh Coarsening. We construct a hierarchical mesh graph $\mathcal{G}^{\mathrm{H}} = \mathcal{G}^{0:L} = (\mathcal{V}^{0:L}, \mathcal{E}^{0:L,\mathrm{M}})$ consisting of $L+1$ mesh graphs with nodes $\mathcal{V}^{0:L}$ and mesh edges $\mathcal{E}^{0:L,\mathrm{M}}$ by coarsening the fine graph $\mathcal{G}^0 := \mathcal{G}$ L times. Coarsening is performed with root-node smoothed aggregation [53] implemented in [59]. The solver takes the fine-mesh adjacency (with self-loops) A^0 as its system matrix and creates a hierarchy of adjacency $A^{0:L}$, upsampling (prolongation) $U^{0:L-1}$ and downsampling (restriction) $D^{0:L-1}$ matrices. In practice, our implementation only requires the fine adjacency A^0, the root-node solver [59], and constructing the graphs of each level from the non-zeros of the returned matrices $A^{1:L}$. The resulting nodes $j \in \mathcal{V}^{0:L}$ of each level are always a subset of the fine mesh nodes. Unlike bi-stride pooling with Delaunay remeshing [22], this AMG-based approach better preserves mesh geometry by leveraging the strength of connections in the adjacency matrix. Figure 10 in Section E visualizes this difference.

We additionally define $L-1$ downsampling edges $\mathcal{E}^{l,\mathrm{D}}$ and upsampling edges $\mathcal{E}^{l,\mathrm{U}}$, which connect nodes between successive levels $\mathcal{G}^l$ and $\mathcal{G}^{l+1}$. We define these as the connections (non-zero values) of the given up- $U^{0:L-1}$ and downsampling $D^{0:L-1}$ matrices, resulting in an extended hierarchy graph $\mathcal{G}^{\mathrm{H}} = (\mathcal{V}^{0:L}, \mathcal{E}^{0:L,\mathrm{M}} \cup \mathcal{E}^{0:L-1,\mathrm{D}} \cup \mathcal{E}^{0:L-1,\mathrm{U}})$. As before, we found that the obtained pooling mappings respect the mesh geometry well, as shown in Figure 10 d) in Section E.

For contact experiments, we add contact edges $\mathcal{E}^{0,\mathrm{C}}$ [22] to the graph hierarchy $\mathcal{G}^{\mathrm{H}}$. In a simulation involving two colliding components, we define a bidirectional edge between the nodes of the first part and the nodes of the second part if their distance is less than the specified contact radius R. The resulting hierarchy is given by $\mathcal{G}^{\mathrm{H}} = (\mathcal{V}^{0:L}, \mathcal{E}^{0:L,\mathrm{M}} \cup \mathcal{E}^{0:L,\mathrm{C}} \cup \mathcal{E}^{0:L-1,\mathrm{D}} \cup \mathcal{E}^{0:L-1,\mathrm{U}})$.

3.3 Algebraic-hierarchical Message Passing Networks (AMPNs).

Encoder. Let $\mathcal{G}^{\mathrm{H}}$ be a hierarchical graph as defined above and $\mathbf{u}_{i,k}^t$ the noisy sample at denoising step k of simulation step t. We define node embeddings $\mathbf{k}_i \in \mathcal{V}^{0:L}$, mesh edge embedding $\mathbf{e}_{ij}^{\mathrm{M}} \in \mathcal{E}^{0:L,\mathrm{M}}$, contact edge embeddings $\mathbf{e}_{ij}^{\mathrm{C}} \in \mathcal{E}^{\mathrm{C}}$, downsampling edge embeddings $\mathbf{e}_{ij}^{\mathrm{D}} \in \mathcal{E}^{0:L-1,\mathrm{D}}$ and upsampling edge embeddings $\mathbf{e}_{ij}^{\mathrm{U}} \in \mathcal{E}^{0:L-1,\mathrm{U}}$. We add relative node distances $\mathbf{x}_{ij}^t = \mathbf{x}_i^t - \mathbf{x}_j^t$ and their norm $|\mathbf{x}_{ij}^t|$ to all edge embeddings and the initial distance $\mathbf{x}_{ij}^0 = \mathbf{x}_i^0 - \mathbf{x}_j^0$ and their norm $|\mathbf{x}_{ij}^0|$ to mesh edges, down- and upsampling embeddings. Node embeddings include a one-hot encoding n_i of the node type. Node embeddings at level $l = 0$ additionally include $\mathbf{u}_{i,k}^t$ and task-specific features. All embeddings are projected to the latent dimension d via linear layers. We add a Fourier encoding [57] for the denoising step k and the normalized level $l^* = l/L$ to inform the AMPN of relative graph depth.

Processor and Decoder. Similar to AMG solvers [54, 53] and UNets [60], Algebraic-hierarchical Message Passing Networks (AMPNs) use a *V-cycle* to propagate information between levels. They consist of five core message passing modules: Pre-Processing, Downsampling, Solving, Upsampling and Post-Processing, as shown in Figure 1. Pre-Processing, Solving and Post-Processing modules use an Intra-Level-Message Passing Stack (Intra-MP-Stack) consisting of N message passing steps to update the heterogeneous subgraph $\mathcal{G}^l = (\mathcal{V}^l, \mathcal{E}^{l,\mathrm{C}} \cup \mathcal{E}^{l,\mathrm{M}})$ of level l. Given node embeddings $\mathbf{k}_i \in \mathcal{V}^l$, contact edge embeddings $\mathbf{e}_{ij}^{\mathrm{C}} \in \mathcal{E}^{l,\mathrm{C}}$ and mesh edge embeddings $\mathbf{e}_{ij}^{\mathrm{M}} \in \mathcal{E}^{l,\mathrm{M}}$, the message passing update of the level graph at step n is defined by

$$
\begin{aligned}
\mathbf{e}_{ij}^{\mathrm{C},n+1} &= W_{\theta,\mathcal{E}^{\mathrm{C}}}^n \, \mathbf{e}_{ij}^{\mathrm{C},n} + f_{\theta,\mathcal{E}^{\mathrm{C}}}^n(\mathbf{k}_i^n, \mathbf{k}_j^n, \mathbf{e}_{ij}^{\mathrm{C},n}) \,, \\
\mathbf{e}_{ij}^{\mathrm{M},n+1} &= W_{\theta,\mathcal{E}^{\mathrm{M}}}^n \, \mathbf{e}_{ij}^{\mathrm{C},n} + f_{\theta,\mathcal{E}^{\mathrm{M}}}^n(\mathbf{k}_i^n, \mathbf{k}_j^n, \mathbf{e}_{ij}^{\mathrm{M},n}) \,, \\
\mathbf{k}_i^{n+1} &= W_{\theta,\mathcal{V}}^n \, \mathbf{k}_i^n + f_{\theta,\mathcal{V}}^n(\mathbf{k}_i^n, \bigoplus_j \mathbf{e}_{ij}^{\mathrm{C},n+1}, \bigoplus_j \mathbf{e}_{ij}^{\mathrm{M},n+1}) \quad .
\end{aligned}
\tag{2}
$$

The operator $\bigoplus$ denotes a permutation-invariant aggregation, $f_{\theta,\cdot}^n$ MLPs and $W_{\theta,\cdot}^n$ weight matrices [46, 5, 52]. Downsampling modules update the subgraph $\mathcal{G}^{l,\mathrm{D}} = (\mathcal{V}^{l+1} \cup \mathcal{V}^l, \mathcal{E}^{l,\mathrm{D}})$ with an Inter-Level-Message Passing Stack (Inter-MP-Stack) of N message passing steps. The receiver embeddings are

$\mathbf{k}_i^{\text{rec}} \in \mathcal{V}^{l+1}$, the sender embeddings $\mathbf{k}_j^{\text{send}} \in \mathcal{V}^l$, and the edge embeddings $\mathbf{e}_{ij} \in \mathcal{E}^{l,\text{D}}$. Similarly, the upsampling layers update the embeddings of the subgraph $\mathcal{G}_l^{\text{U}} = (\mathcal{V}^{l+1} \cup \mathcal{V}^l, \mathcal{E}^{l,\text{U}})$ of level l. Here, the receiver embeddings are $\mathbf{k}_i^{\text{rec}} \in \mathcal{V}^l$, the sender embeddings $\mathbf{k}_j^{\text{send}} \in \mathcal{V}^{l+1}$, and the edge embeddings $\mathbf{e}_{ij} \in \mathcal{E}^{l,\text{U}}$. A message passing step of an Intra-MP-Stack is defined as

$$\begin{aligned} \mathbf{e}_{ij}^{n+1} &= W_{\theta,\mathcal{E}}^n \, \mathbf{e}_{ij}^n + f_{\theta,\mathcal{E}}^n(\mathbf{k}_i^{\text{rec},n}, \mathbf{k}_j^{\text{send},n}, \mathbf{e}_{ij}^n) \,, \\ \mathbf{k}_i^{\text{rec},n+1} &= W_{\theta,\mathcal{V}}^n \, \mathbf{k}_i^{\text{rec},n} + f_{\theta,\mathcal{V}}^n(\mathbf{k}_i^{\text{rec},n}, \bigoplus_j \mathbf{e}_{ij}^{n+1}) \quad . \end{aligned} \tag{3}$$

Our *V-cycle* starts at level $l = 0$ with pre-processing and downsampling at each level, repeated until the coarsest level $l = L$ is reached. We then apply multiple message passing steps at level L, which has the largest receptive field. Next, we upsample and post-process each level back up to $l = 0$. All Pre-Processing, Downsampling, Upsampling, and Post-Processing modules share weights across levels. A final linear layer decodes the fine-level node embeddings $\mathbf{k}_i \in \mathcal{V}^0$ to produce the velocity prediction $\mathbf{v}_{i,\theta}(\mathbf{u}_{i,k}^t, k, \mathbf{u}_{i,0}^{t-1})$.

4 Experiments

Datasets. We evaluate our model on the three different datasets, namely BENDINGBEAM, IMPACTPLATE [22] and DEFORMINGPLATE [5]. We introduce the BENDINGBEAM dataset (Figure 3a)), featuring quasi-static, geometrically non-linear deformations of beams with high aspect ratios. The setup challenges models to capture global deformations via broad receptive fields and resolve high spatial frequencies due to locally thin, low-stiffness regions. In IMPACTPLATE (Figure 3 b)), the models must learn flexible dynamics with varying material parameters and accurately resolve very localized deformation at the contact point. DEFORMINGPLATE (Figure 3 c)) considers quasi-static contact simulations induced by scripted actuators that deform 3D plates consisting of nonlinear, hyperelastic material.

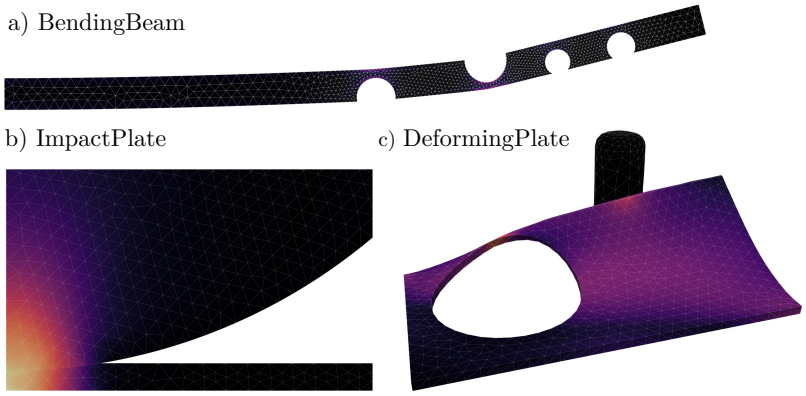


Figure 3: Example predictions of ROBIN on the considered datasets. ROBIN predicts the part deformations as well as the von Mises stress (color, yellow is high) on all experiments. **a)** BENDINGBEAM considers global part deformations of beams induced by local forces. **b)** In IMPACTPLATE, the models have to predict locally large deformations, caused by a collision with an accelerated ball. **c)** The hyperelastic plates in DEFORMINGPLATE are deformed by a scripted actuator.

Experimental setup. For all tasks, we target the displacement residual of the node positions with respect to the next time step $\Delta\mathbf{u}_{i,0}^t = \mathbf{x}_i^{t+1} - \mathbf{x}_i^t$ during denoising. We additionally denoise the von Mises stress $\sigma_{\text{vMises},i,0}^t$ directly without residuals to gain further insight into the dynamics of the three experiments. ROBIN uses $K = 20$ denoising steps and a denoising stride of $m = 1$ by default. The task-specific features are specified in Section C, listed in Table 2. We measure the prediction error using an *RMSE*, as specified in Section C. Section C also provides information about additional settings of ROBIN, including training details and hyperparameters.

Baselines. We compare our model with three prominent baselines for nonlinear deformation simulations, namely MGNs, Hierarchical Contact Mesh Transformers (HCMTs), and Bi-Stride Multi-Scale GNNs (BSMSs). Detailed setups are provided in Section D.

Variants. We demonstrate that a *single trained* ROBIN can easily switch between different rollout modes by varying the denoising stride m and the truncation step k_{tr}. We therefore evaluate multiple rollout settings (m/k_{tr}) on the same trained model ROBIN model: the default (1/20), conventional autoregressive diffusion inference (20/20) (i.e., m=20 denoising steps per physical step), and an intermediate variant (5/20) with $m = 5$. To study early stopping, we further reduce the truncation step k_{tr} with (1/10), (1/5), (1/3), (1/2) and (1/1).

Ablations. We ablate key components of ROBIN to assess their impact. *10 diffusion steps* and *5 diffusion steps* train with reduced diffusion length K. *W/o diffusion* trains the AMPN as a one-step autoregressive model with MSE loss. *W/o hierarchy* disables hierarchical message passing, operating only on the fine mesh G^0. *State prediction* replaces residual-based prediction with direct denoising of $\mathbf{u}_{i,k}^t$ instead of $\Delta\mathbf{u}_{i,k}^t$. *W/o shared layer* uses 15 unshared message passing layers. *HCMT model* replaces AMPNs by HCMTs as the hierarchical model for ROBIN. Section D provides additional implementation and training details.

Generalization to large meshes. To assess upscaling, we introduce BENDINGBEAMLARGE with meshes on average over ten times larger than in BENDINGBEAM. We compare fine-tuning a pre-trained ROBIN to training from scratch, showing rapid transfer to substantially larger meshes without architectural changes, enabled by shared blocks in the AMPNs that accommodate varying hierarchy depths.

5 Results

Baselines. Figure 4 a) compares the rollout RMSE of ROBIN against HCMT, MGN, and BSMS. ROBIN yields substantial error reductions for the prominent IMPACTPLATE and DEFORMING-PLATE datasets, and yields an even larger improvement on BENDINGBEAM. Figure 5 demonstrates that ROBIN is able to propagate local boundary conditions across the part for accurate predictions of the global part deformation. In addition, ROBIN resolves the non-linear bending curve of the FEM. This requires fusing geometric features across scales, e.g., global part dimensions together with local thin walls, for an accurate prediction of the global part stiffness. All baselines struggle to reproduce the FEM results, particularly the global deformation modes.

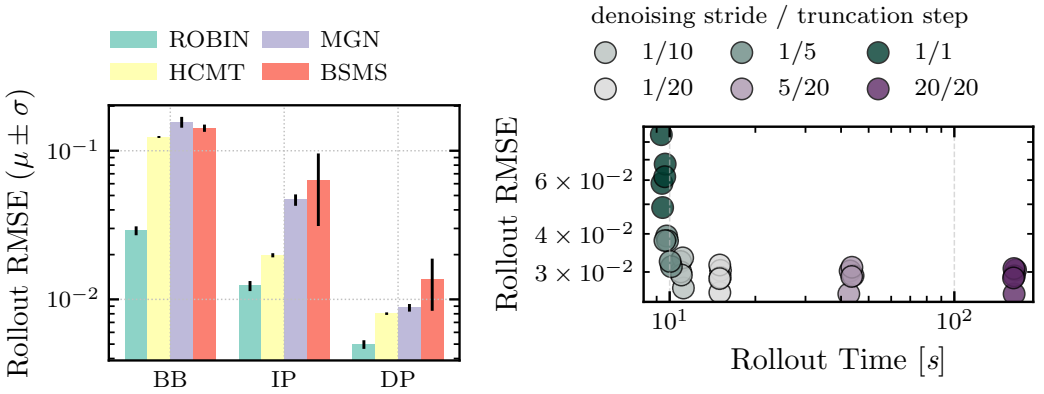


Figure 4: **Left:** Rollout error measured by the RMSE of the predicted nodes positions. ROBIN surpasses the accuracy of the baselines HCMT, MGN, and BSMS on all three datasets BENDING-BEAM, IMPACTPLATE and DEFORMINGPLATE. **Right:** Comparison of inference time and error of ROBIN and its variants on BENDINGBEAM. The default variant of ROBIN (1/20) achieves the same accuracy as conventional diffusion inference (20/20), while the inference speed is close to the one step variant (1/1). Reducing the truncation step k_{tr} trades accuracy for speed.

Inference speed. Figure 4 b) plots rollout RMSE versus inference time on BENDINGBEAM. The default variant (1/20) of ROBIN is about an order of magnitude faster than conventional autoregressive diffusion inference (20/20), without compromising accuracy. Decreasing the truncation step k_{tr} (i.e., fewer diffusion steps per physical step at inference) trades rollout accuracy for speed. Nevertheless, the fastest variant (1/1), which can be seen as a one-step model variant of ROBIN, still significantly outperforms the accuracy of the baselines (cf. Figure 4 a)). Most notably, ROBIN (1/20) requires only $\approx 58\%$ more time than the one-step variant (1/1), despite performing K=20 denoising steps per time step, highlighting the efficiency of the parallel denoising scheme ROBI. Similar trends hold for IMPACTPLATE and DEFORMINGPLATE (cf. Figure 11 in Section E). Figure 14 in Section E

Figure 5: Comparison of the predicted rollout deformations and von Mises stresses (color, yellow is high) on BENDINGBEAM between **a)** the FEM, **b)** ROBIN, **c)** HCMT, **d)** MGN, and **e)** BSMS. ROBIN is able to accurately reproduce the FEM results. Neither HCMT, MGN nor BSMS are able to resolve global deformation modes, illustrating the importance of the AMPN for global message propagation.

visualizes how high frequency errors accumulate over the rollout if we skip the later denosing steps, demonstrating the importance of reducing the high frequency errors for long rollouts despite low displacement RMSE.

Diffusion truncation. Figure 6 visualizes displacement RMSE and fine-mesh edge-wise gradient RMSE $||(\mathbf{u}_i - \mathbf{u}_j)||_2 / ||(\mathbf{x}_i^0 - \mathbf{x}_j^0)||_2$, against ground truth. Early diffusion steps primarily remove low-frequency error (global RMSE drops), whereas later steps reduce high-frequency error (gradient RMSE). Skipping the later denoising steps results in high frequency error accumulation and mesh degradation throughout the rollout (cf. Figure 14 in Section E), as observed in one step models [9]. With m=1, ROBIN does not increase the gradient error on BENDINGBEAM. We observe similar results on DEFORMINGPLATE and IMPACTPLATE, as illustrated in Figure 15 in Section E.

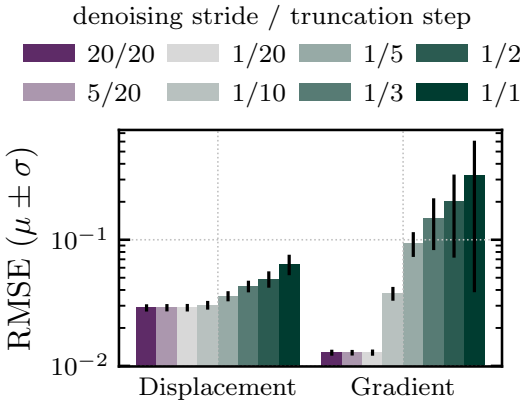


Figure 6: Comparison of the predicted rollout displacements and displacement gradients on BENDINGBEAM for different truncation steps k_{tr}. While the first diffusion steps strongly decrease the displacement RMSE, the later steps are important to reduce the local displacement gradient RMSE. A fast inference with a low denoising stride neither increases the displacement RMSE nor the gradient RMSE during a rollout.

Ablations. As shown in Figure 7, ablations confirm the importance of each ROBIN component. Reducing the diffusion length K slightly decreases accuracy across all datasets. However, even with K=5 diffusion steps, ROBIN remains significantly more accurate than all baselines. Using non-shared layers significantly degrades performance on BENDINGBEAM, likely due to a reduced effective receptive field. The large error increase for *State prediction* indicates that residual prediction is crucial for LAGRANGIAN simulations. Across all datasets, ROBIN outperforms the non-hierarchical variant, the non-diffusion variant, and the combination of DDPMs with the strongest hierarchical baseline (HCMT), demonstrating the synergy between DDPMs and AMPNs.

Generalization to large meshes. Fine-tuning the pre-trained ROBIN converges markedly faster and attains substantially lower RMSE than training from scratch, demonstrating mesh-size independence and efficient transfer to deeper AMG hierarchies with twelve times more nodes, without architectural changes. In Figure 8, pretrained predictions are visually indistinguishable from FEM even in thin, high-stress regions with large bending. The model trained from scratch exhibits negligible deformation due to much slower convergence within the same training budget. ROBIN consistently requires 31 s per case, while the numerical solver averages 108 s and can require up to 4248 s, due to problem-dependent nonlinear convergence costs. Table 3 in Section E lists the quantitative results.

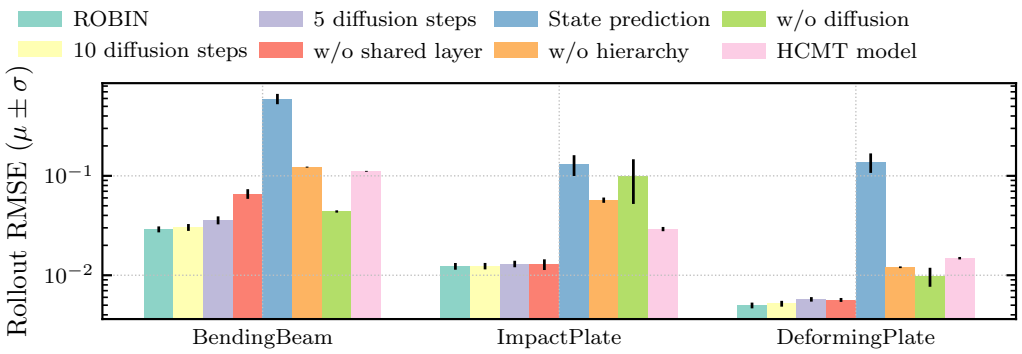


Figure 7: Rollout RMSE of the displacement predictions. ROBIN remains accurate, even for a very small number of diffusion steps. Shared layers are crucial for BENDINGBEAM to increase the receptive field. Replacing residual predictions with state predictions, hierarchical architectures with non-hierarchical architectures or with HCMT architectures, and diffusion with non-diffusion architectures significantly decrease the accuracy across all datasets.

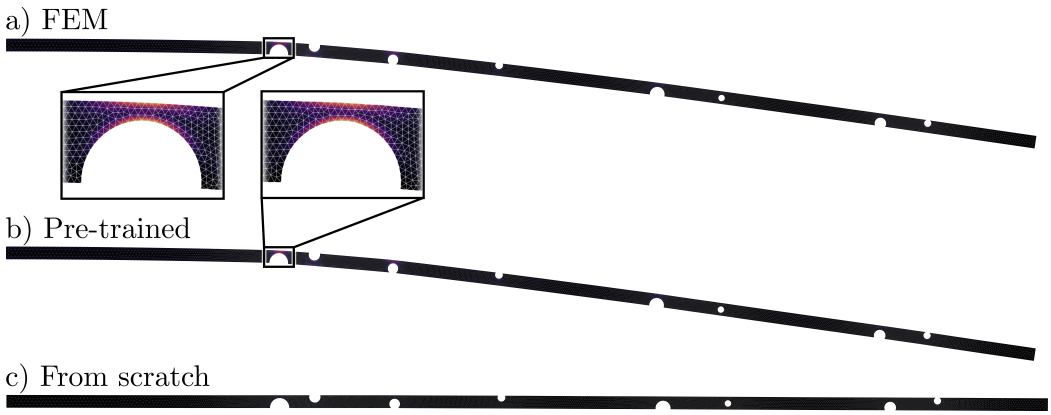


Figure 8: Comparison of the predicted rollout deformations and von Mises stresses (color, yellow is large) on BENDINGBEAMLARGE between **a)** the FEM, **b)** a fine-tuned ROBIN model, which was pre-trained on the small BENDINGBEAM dataset, and, **c)** a ROBIN model trained from scratch for the same number of training iterations. The fine-tuned ROBIN closely matches the FEM deformation and stress distribution, capturing even local high-stress hotspots, whereas the from-scratch model underestimates deformation due to much slower convergence within the same training budget.

6 Conclusion

We introduced Rolling Diffusion-Batched Inference Network (ROBIN), a diffusion-based HGNN that utilizes AMPNs to refine mesh-based predictions across scales. Leveraging the expressiveness of multiscale message passing and the accuracy of diffusion, ROBIN outperforms state-of-the-art simulators on varied nonlinear solid mechanics tasks in terms of predictive accuracy. These tasks include a novel BENDINGBEAM dataset that reveals limitations of current learned simulators. ROBI, ROBIN's inference scheme, parallelizes diffusion across time steps, reducing inference runtime by up to an order of magnitude without sacrificing accuracy. We validated ROBIN on three challenging datasets, including the new BENDINGBEAM benchmark, and demonstrated significant gains in accuracy and efficiency. We discuss the broader impact of this work and our method in Appendix A.

Limitations and Future Work. ROBIN currently does not possess SO(3) equivariance. Adding this property could improve the accuracy of orientation-sensitive predictions. We focus on DDPM, while other diffusion formulations or denoising schedules could provide valuable insights and further enhance performance. Similarly, our experiments cover nonlinear solid mechanics, and extensions to other domains, such as fluid dynamics, are a promising direction. Lastly, ROBIN's combination of fast inference and high accuracy opens opportunities for accelerating multi-stage design and optimization workflows.

Acknowledgements

This work is part of the DFG AI Research Unit 5339 regarding the combination of physics-based simulation with AI-based methodologies for the fast maturation of manufacturing processes. The financial support by German Research Foundation (DFG, Deutsche Forschungsgemeinschaft) is gratefully acknowledged. The authors acknowledge support by the state of Baden-Württemberg through bwHPC, as well as the HoreKa supercomputer funded by the Ministry of Science, Research and the Arts Baden-Württemberg and by the German Federal Ministry of Education and Research. This work is supported by the Helmholtz Association Initiative and Networking Fund on the HAICORE@KIT partition.

References

[1] Mats G. Larson and Fredrik Bengzon. *The Finite Element Method: Theory, Implementation, and Applications*, volume 10 of *Texts in Computational Science and Engineering*. Springer, Berlin, Heidelberg, 2013. ISBN 978-3-642-33286-9 978-3-642-33287-6. doi: 10.1007/978-3-642-33287-6.

[2] F. Moukalled, L. Mangani, and M. Darwish. *The Finite Volume Method in Computational Fluid Dynamics: An Advanced Introduction with OpenFOAM® and Matlab*, volume 113 of *Fluid Mechanics and Its Applications*. Springer International Publishing, Cham, 2016. ISBN 978-3-319-16873-9 978-3-319-16874-6. doi: 10.1007/978-3-319-16874-6.

[3] M. Necati Özişik, Helcio R. B. Orlande, Marcelo J. Colaço, and Renato M. Cotta. *Finite Difference Methods in Heat Transfer*. CRC Press, July 2017. ISBN 978-1-351-34991-8.

[4] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to Simulate Complex Physics with Graph Networks. In *Proceedings of the 37th International Conference on Machine Learning*, pages 8459–8468. PMLR, November 2020.

[5] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter Battaglia. Learning Mesh-Based Simulation with Graph Networks. In *International Conference on Learning Representations*, October 2020.

[6] Kelsey Allen, Tatiana Lopez-Guevara, Kimberly L. Stachenfeld, Alvaro Sanchez Gonzalez, Peter Battaglia, Jessica B. Hamrick, and Tobias Pfaff. Inverse Design for Fluid-Structure Interactions using Graph Network Simulators. *Advances in Neural Information Processing Systems*, 35:13759–13774, December 2022.

[7] Clemens Zimmerling, Christian Poppe, Oliver Stein, and Luise Kärger. Optimisation of manufacturing process parameters for variable component geometries using reinforcement learning. *Materials & Design*, 214:110423, February 2022. ISSN 02641275. doi: 10.1016/j.matdes.2022.110423.

[8] Tobias Würth, Constantin Krauß, Clemens Zimmerling, and Luise Kärger. Physics-informed neural networks for data-free surrogate modelling and engineering optimization – An example from composite manufacturing. *Materials & Design*, 231:112034, July 2023. ISSN 02641275. doi: 10.1016/j.matdes.2023.112034.

[9] Phillip Lippe, Bas Veeling, Paris Perdikaris, Richard Turner, and Johannes Brandstetter. PDE-Refiner: Achieving Accurate Long Rollouts with Neural PDE Solvers. *Advances in Neural Information Processing Systems*, 36:67398–67433, December 2023.

[10] David Ruhe, Jonathan Heek, Tim Salimans, and Emiel Hoogeboom. Rolling Diffusion Models. In *Forty-First International Conference on Machine Learning*, June 2024.

[11] Georg Kohl, Liwei Chen, and Nils Thuerey. Benchmarking Autoregressive Conditional Diffusion Models for Turbulent Flow Simulation. In *ICML 2024 AI for Science Workshop*, June 2024.

[12] Wenzhuo Liu, Mouadh Yagoubi, and Marc Schoenauer. Multi-resolution Graph Neural Networks for PDE Approximation. In Igor Farkaš, Paolo Masulli, Sebastian Otte, and Stefan Wermter, editors, *Artificial Neural Networks and Machine Learning – ICANN 2021*, pages 151–163, Cham, 2021. Springer International Publishing. ISBN 978-3-030-86365-4. doi: 10.1007/978-3-030-86365-4_13.

[13] Meire Fortunato, Tobias Pfaff, Peter Wirnsberger, Alexander Pritzel, and Peter Battaglia. MultiScale MeshGraphNets, October 2022.

[14] Mario Lino, Stathi Fotiadis, Anil A. Bharath, and Chris D. Cantwell. Multi-scale rotation-equivariant graph neural networks for unsteady Eulerian fluid dynamics. *Physics of Fluids*, 34 (8):087110, August 2022. ISSN 1070-6631. doi: 10.1063/5.0097679.

[15] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in neural information processing systems*, 34:8780–8794, 2021.

[16] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

[17] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022.

[18] Zhifeng Kong, Wei Ping, Jiaji Huang, Kexin Zhao, and Bryan Catanzaro. Diffwave: A versatile diffusion model for audio synthesis. In *International Conference on Learning Representations*, 2020.

[19] Nanxin Chen, Yu Zhang, Heiga Zen, Ron J Weiss, Mohammad Norouzi, and William Chan. Wavegrad: Estimating gradients for waveform generation. In *International Conference on Learning Representations*, 2020.

[20] Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, page 02783649241273668, 2023.

[21] Paul Maria Scheikl, Nicolas Schreiber, Christoph Haas, Niklas Freymuth, Gerhard Neumann, Rudolf Lioutikov, and Franziska Mathis-Ullrich. Movement primitive diffusion: Learning gentle robotic manipulation of deformable objects. *IEEE Robotics and Automation Letters*, 2024.

[22] Youn-Yeol Yu, Jeongwhan Choi, Woojin Cho, Kookjin Lee, Nayong Kim, Kiseok Chang, ChangSeung Woo, Ilho Kim, SeokWoo Lee, Joon Young Yang, Sooyoung Yoon, and Noseong Park. Learning Flexible Body Collision Dynamics with Hierarchical Contact Mesh Transformer. In *The Twelfth International Conference on Learning Representations*, October 2023.

[23] Junuthula Narasimha Reddy. *An Introduction to the Finite Element Method*, volume 3. McGraw-Hill New York, 2005.

[24] Susanne C. Brenner and L. Ridgway Scott. *The Mathematical Theory of Finite Element Methods*, volume 15 of *Texts in Applied Mathematics*. Springer New York, New York, NY, 2008. ISBN 978-0-387-75933-3 978-0-387-75934-0. doi: 10.1007/978-0-387-75934-0.

[25] Xiaoxiao Guo, Wei Li, and Francesco Iorio. Convolutional Neural Networks for Steady Flow Approximation. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 481–490, San Francisco California USA, August 2016. ACM. ISBN 978-1-4503-4232-2. doi: 10.1145/2939672.2939738.

[26] Zichao Long, Yiping Lu, Xianzhong Ma, and Bin Dong. PDE-Net: Learning PDEs from Data. In *Proceedings of the 35th International Conference on Machine Learning*, pages 3208–3216. PMLR, July 2018.

[27] Johannes Brandstetter, Daniel E. Worrall, and Max Welling. Message Passing Neural PDE Solvers. In *International Conference on Learning Representations*, October 2021.

[28] Jonas Linkerhägner, Niklas Freymuth, Paul Maria Scheikl, Franziska Mathis-Ullrich, and Gerhard Neumann. Grounding graph network simulators using physical sensor observations. *arXiv preprint arXiv:2302.11864*, 2023.

[29] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural Ordinary Differential Equations. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[30] Valerii Iakovlev, Markus Heinonen, and Harri Lähdesmäki. Learning continuous-time PDEs from sparse data with graph neural networks. In *International Conference on Learning Representations*, October 2020.

[31] M. Raissi, P. Perdikaris, and G.E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, February 2019. ISSN 00219991. doi: 10.1016/j.jcp.2018.10.045.

[32] Sina Amini Niaki, Ehsan Haghighat, Trevor Campbell, Anoush Poursartip, and Reza Vaziri. Physics-informed neural network for modelling the thermochemical curing process of composite-tool systems during manufacture. *Computer Methods in Applied Mechanics and Engineering*, 384:113959, October 2021. ISSN 00457825. doi: 10.1016/j.cma.2021.113959.

[33] Luning Sun, Han Gao, Shaowu Pan, and Jian-Xun Wang. Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data. *Computer Methods in Applied Mechanics and Engineering*, 361:112732, April 2020. ISSN 00457825. doi: 10.1016/j.cma.2019.112732.

[34] Han Gao, Luning Sun, and Jian-Xun Wang. PhyGeoNet: Physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state PDEs on irregular domain. *Journal of Computational Physics*, 428:110079, March 2021. ISSN 00219991. doi: 10.1016/j.jcp.2020.110079.

[35] Dule Shu, Zijie Li, and Amir Barati Farimani. A physics-informed diffusion model for high-fidelity flow field reconstruction. *Journal of Computational Physics*, 478:111972, April 2023. ISSN 0021-9991. doi: 10.1016/j.jcp.2023.111972.

[36] Han Gao, Matthew J. Zahr, and Jian-Xun Wang. Physics-informed graph neural Galerkin networks: A unified framework for solving PDE-governed forward and inverse problems. *Computer Methods in Applied Mechanics and Engineering*, 390:114502, February 2022. ISSN 0045-7825. doi: 10.1016/j.cma.2021.114502.

[37] Tobias Würth, Niklas Freymuth, Clemens Zimmerling, Gerhard Neumann, and Luise Kärger. Physics-informed MeshGraphNets (PI-MGNs): Neural finite element solvers for non-stationary and nonlinear simulations on arbitrary meshes. *Computer Methods in Applied Mechanics and Engineering*, 429:117102, September 2024. ISSN 0045-7825. doi: 10.1016/j.cma.2024.117102.

[38] Zongyi Li, Nikola Borislavov Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier Neural Operator for Parametric Partial Differential Equations. In *International Conference on Learning Representations*, October 2020.

[39] Gege Wen, Zongyi Li, Kamyar Azizzadenesheli, Anima Anandkumar, and Sally M. Benson. U-FNO—An enhanced Fourier neural operator-based deep-learning model for multiphase flow. *Advances in Water Resources*, 163:104180, May 2022. ISSN 0309-1708. doi: 10.1016/j.advwatres.2022.104180.

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

[41] Zhongkai Hao, Zhengyi Wang, Hang Su, Chengyang Ying, Yinpeng Dong, Songming Liu, Ze Cheng, Jian Song, and Jun Zhu. GNOT: A General Neural Operator Transformer for Operator Learning.

[42] Steeven Janny, Aurélien Bénéteau, Madiha Nadri, Julie Digne, Nicolas Thome, and Christian Wolf. EAGLE: Large-scale Learning of Turbulent Fluid Dynamics with Mesh Transformers. In *The Eleventh International Conference on Learning Representations*, September 2022.

[43] Xu Han, Han Gao, Tobias Pfaff, Jian-Xun Wang, and Li-Ping Liu. Predicting Physics in Mesh-reduced Space with Temporal Attention, May 2022.

[44] Benedikt Alkin, Andreas Fürst, Simon Schmid, Lukas Gruber, Markus Holzleitner, and Johannes Brandstetter. Universal Physics Transformers. *CoRR*, January 2024.

[45] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M. Bronstein. Geometric Deep Learning on Graphs and Manifolds Using Mixture Model CNNs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5115–5124, 2017.

[46] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks, October 2018.

[47] Artur Grigorev, Michael J. Black, and Otmar Hilliges. HOOD: Hierarchical Graphs for Generalized Modelling of Clothing Dynamics. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16965–16974, 2023.

[48] Yadi Cao, Menglei Chai, Minchen Li, and Chenfanfu Jiang. Efficient Learning of Mesh-Based Physical Simulation with Bi-Stride Multi-Scale Graph Neural Network. In *Proceedings of the 40th International Conference on Machine Learning*, pages 3541–3558. PMLR, July 2023.

[49] Roberto Perera and Vinamra Agrawal. Multiscale graph neural networks with adaptive mesh refinement for accelerating mesh-based simulations. *Computer Methods in Applied Mechanics and Engineering*, 429:117152, September 2024. ISSN 0045-7825. doi: 10.1016/j.cma.2024.117152.

[50] Tianyi Li, Alessandra S. Lanotte, Michele Buzzicotti, Fabio Bonaccorso, and Luca Biferale. Multi-Scale Reconstruction of Turbulent Rotating Flows with Generative Diffusion Models. *Atmosphere*, 15:60, December 2023. doi: 10.3390/atmos15010060.

[51] Marten Lienen, David Lüdke, Jan Hansen-Palmus, and Stephan Günnemann. From Zero to Turbulence: Generative Modeling for 3D Flow Simulation. In *The Twelfth International Conference on Learning Representations*, October 2023.

[52] Mario Lino Valencia, Tobias Pfaff, and Nils Thuerey. Learning Distributions of Complex Fluid Simulations with Diffusion Graph Networks. In *The Thirteenth International Conference on Learning Representations*, October 2024.

[53] Petr Vanek, Jan Mandel, and Marian Brezina. Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems. *Computing*, 56(3):179–196, 1996.

[54] J. W. Ruge and K. Stüben. 4. Algebraic Multigrid. In Stephen F. McCormick, editor, *Multigrid Methods*, pages 73–130. Society for Industrial and Applied Mathematics, January 1987. ISBN 978-1-61197-188-0 978-1-61197-105-7. doi: 10.1137/1.9781611971057.ch4.

[55] Salva Rühling Cachay, Bo Zhao, Hailey Joren, and Rose Yu. DYffusion: A Dynamics-informed Diffusion Model for Spatiotemporal Forecasting.

[56] Han Gao, Xu Han, Xiantao Fan, Luning Sun, Li-Ping Liu, Lian Duan, and Jian-Xun Wang. Bayesian conditional diffusion models for versatile spatiotemporal turbulence generation. *Computer Methods in Applied Mechanics and Engineering*, 427:117023, July 2024. ISSN 0045-7825. doi: 10.1016/j.cma.2024.117023.

[57] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising Diffusion Probabilistic Models. In *Advances in Neural Information Processing Systems*, volume 33, pages 6840–6851. Curran Associates, Inc., 2020.

[58] Tim Salimans and Jonathan Ho. Progressive Distillation for Fast Sampling of Diffusion Models. In *International Conference on Learning Representations*, October 2021.

[59] Nathan Bell, Luke N. Olson, and Jacob Schroder. PyAMG: Algebraic multigrid solvers in python. *Journal of Open Source Software*, 7(LA-UR-23-26551), 2022.

[60] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham, 2015. Springer International Publishing. ISBN 978-3-319-24574-4. doi: 10.1007/978-3-319-24574-4_28.

[61] Tom Gustafsson and Geordie Drummond Mcbain. scikit-fem: A python package for finite element assembly. *Journal of Open Source Software*, 5(52):2369, 2020.

[62] A. Paszke. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

[63] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, January 2017.

[64] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 107:3–11, November 2018. ISSN 08936080. doi: 10.1016/j.neunet.2017.12.012.

[65] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, and Michael Isard. {TensorFlow}: A system for {Large-Scale} machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, 2016.

[66] D. T. Lee and B. J. Schachter. Two algorithms for constructing a Delaunay triangulation. *International Journal of Computer & Information Sciences*, 9(3):219–242, June 1980. ISSN 0091-7036, 1573-7640. doi: 10.1007/BF00977785.

A Broader Impact

The ML-based simulator, Rolling Diffusion-Batched Inference Network (ROBIN), offers significant advantages for computational modeling and simulation. This is achieved by reducing computational costs while maintaining accuracy. This enables engineers to iterate through significantly more design variations or to quickly evaluate numerous scenarios using the fast model. However, like all powerful computational tools, there is a risk of misuse, for instance, in weapons development or unsustainable resource exploitation.

B Datasets

Table 1 provides an overview of the considered datasets in this work.

Table 1: Comparison of the datasets BENDINGBEAM, BENDINGBEAMLARGE, IMPACTPLATE [22] and DEFORMINGPLATE [5] considered in this work. The column *Nonlinearity* distinguishes three different types: geometry (Geo), material (Mat), and boundary conditions (BC).

Datasets	Dynamic	Nonlinearity	avg. # Nodes	Mesh Type	Steps T	Dim
BENDINGBEAM	Quasi-Static	Geo	744	Triangles	400	2D
BENDINGBEAMLARGE	Quasi-Static	Geo	8897	Triangles	100	2D
IMPACTPLATE	Dynamic	Geo, BC	2208	Triangles	52	2D
DEFORMINGPLATE	Quasi-Static	Geo, BC, Mat	1271	Tetrahedrons	400	3D

BENDINGBEAM. This dataset considers the bending of beam parts due to external forces. Bending is one of the most basic deformation modes of parts in structural mechanics. The dataset is designed as a diagnostic benchmark for neural PDE solvers, addressing various potential bottlenecks. The force and handle boundary conditions are very local, only being defined on a small subset of mesh nodes (cf. Figure 9). However, the resulting deformations affect all nodes.

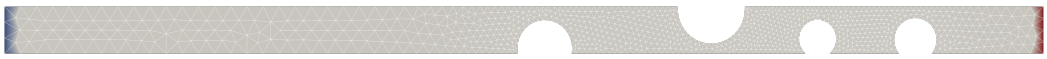


Figure 9: Visualization of the different node types on the BENDINGBEAM experiment. Blue nodes mark handle boundaries with fixed node positions. The red color indicates nodes where the force boundary condition is applied. Those locally applied forces in combination with the global, geometry-dependent part stiffness mainly determine the global deformation.

Hence, the neural PDE must effectively propagate local information to all nodes of the mesh. Next, the dataset considers beams with large aspect ratios. This results in large graph diameters, which represent the shortest path between the most distant nodes. The mesh resolution is increased at thin walls, which additionally increases the graph diameter. The local boundary conditions have to be transmitted across a large number of nodes, which challenges the ability to propagate messages globally. The geometry and especially the thin locations of the geometry strongly influence the global bending stiffness and deformation of the part. Overall, the model has to output accurate solutions across various spatial frequencies.

The solutions are created with scikit-fem [61], iteratively solved using Newton-Raphson until the residual fell below a tolerance of 10^{-8}. Each simulation is solved for a total number of 400 time steps. We create a total number of 1000 simulations for training, 100 for validation and 100 for testing.

To evaluate the generalization capability of ROBIN, we create an additional dataset variant BENDINGBEAMLARGE, containing meshes more than ten times larger than those in BENDINGBEAM, with an average of 8897 nodes. One simulation in BENDINGBEAMLARGE contains 100 time steps.

C Setup

Hardware and Compute. We train all models on a single NVIDIA A100 GPU with a maximum training time of 48 hours, while most models required approximately 40 hours. In total, we trained

11 models across 3 datasets, each on 5 seeds: ROBIN, MGN, HCMT, BSMS, as well as 7 ablations of ROBIN. That amounts to 40 hours$\times 5 \times 3 \times 11 = 6600$ hours training time. Furthermore, we trained two models on BENDINGBEAMLARGE: a pre-trained ROBIN model and a model from scratch, each on 5 seeds for approximately 40 hours, yielding a total duration of 40 hours $\times$ 5 $\times$ 2 = 400 hours. For each training, we required a comparable amount of time for development and hyperparameter tuning. Additionally, we conducted inference experiments to measure the inference speed for 4 models (ROBIN, MGN, HCMT, and BSMS) and 7 ROBIN variants across 5 different seeds. We run inference experiments on 3 datasets, with each experiment taking about 1 hour on average. In total we obtain a runtime of 1 hour $\times$ 5 $\times$ 3 $\times$ 11 = 165 hours for the inference experiments.

Training. We implement ROBIN in *PyTorch* [62] and train it with ADAM [63]. We use an exponential learning rate decay, which decreases the learning rate from $1e-4$ to $1e-6$ over the training time, including 1000 linearly increasing warm-up steps. We clip gradients such that their L_2-norm doesn't exceed 1. We train ROBIN in BENDINGBEAM with 9M samples and in IMPACTPLATE with 6M samples both with a batch size of 16, resulting in 562,500 and 375,000 training iterations. In DEFORMINGPLATE we reduce the batch size to 12 and train for 300,000 iterations with 3.6M samples.

Features. Table 2 provides an overview of the used input and output features for ROBIN. In addition to the default features, we extend the node embeddings of BENDINGBEAM with the force residual Δf_{BC}, which is defined by the boundary condition. In IMPACTPLATE, we add the density ρ_i and the Young's modulus Y_i as node features. In DEFORMINGPLATE, we add the scripted displacement residual $\Delta \mathbf{u}_{\text{BC}}$ of the actuator. We normalize all input features based on the training dataset, setting them to have a zero mean and unit variance. We add a small amount of training noise [4, 5] of 10^{-5} $\sigma_{\mathbf{x}}$ to the node positions $\mathbf{x}_i^t$, where we scale the noise level with the standard deviation of the features $\sigma_{\mathbf{x}}$. For IMPACTPLATE we noise the input history $\Delta \mathbf{u}_{i,0}^{t-1} = \mathbf{x}_i^t - \mathbf{x}_i^{t-1}$ with 10^{-3} $\sigma_{\mathbf{x}}$ to prevent overfitting on the history.

Table 2: Node $\mathbf{k}_i$ and edge embeddings $\mathbf{e}_{ij}$ for the different datasets, depending on the node $\mathcal{V}$ and edge sets $\mathcal{E}$.

Datasets	**Inputs** $\mathcal{V}^0$	**Inputs** $\mathcal{V}^{1:L}$	**Inputs** $\mathcal{E}^{0:L,\text{M}}$	**Inputs** $\mathcal{E}^{0:L,\text{C}}$	**Inputs** $\mathcal{E}^{0:L,\text{U/D}}$	**Outputs** $\mathcal{V}^{0,\text{M}}$
BENDINGBEAM	$\mathbf{n}_i$, $\Delta \mathbf{u}_{i,k}^t$, Δf_{BC}	$\mathbf{n}_i$	$\mathbf{x}_{ij}^t$,$\lvert\mathbf{x}_{ij}^t\rvert$, $\mathbf{x}_{ij}^0$,$\lvert\mathbf{x}_{ij}^0\rvert$	$\mathbf{x}_{ij}^t$,$\lvert\mathbf{x}_{ij}^t\rvert$	$\mathbf{x}_{ij}^t$,$\lvert\mathbf{x}_{ij}^t\rvert$, $\mathbf{x}_{ij}^0$,$\lvert\mathbf{x}_{ij}^0\rvert$	$\mathbf{v}_{i,\theta}(\Delta \mathbf{u}_{i,k}^t)$
IMPACTPLATE	$\mathbf{n}_i$, $\Delta \mathbf{u}_{i,k}^t$, $\Delta \mathbf{u}_{i,0}^{t-1}$, ρ_i, Y_i	$\mathbf{n}_i$	$\mathbf{x}_{ij}^t$,$\lvert\mathbf{x}_{ij}^t\rvert$, $\mathbf{x}_{ij}^0$,$\lvert\mathbf{x}_{ij}^0\rvert$	$\mathbf{x}_{ij}^t$,$\lvert\mathbf{x}_{ij}^t\rvert$	$\mathbf{x}_{ij}^t$,$\lvert\mathbf{x}_{ij}^t\rvert$, $\mathbf{x}_{ij}^0$,$\lvert\mathbf{x}_{ij}^0\rvert$	$\mathbf{v}_{i,\theta}(\Delta \mathbf{u}_{i,k}^t)$
DEFORMINGPLATE	$\mathbf{n}_i$, $\Delta \mathbf{u}_{i,k}^t$, $\Delta \mathbf{u}_{\text{BC}}$	$\mathbf{n}_i$	$\mathbf{x}_{ij}^t$,$\lvert\mathbf{x}_{ij}^t\rvert$, $\mathbf{x}_{ij}^0$,$\lvert\mathbf{x}_{ij}^0\rvert$	$\mathbf{x}_{ij}^t$,$\lvert\mathbf{x}_{ij}^t\rvert$	$\mathbf{x}_{ij}^t$,$\lvert\mathbf{x}_{ij}^t\rvert$, $\mathbf{x}_{ij}^0$,$\lvert\mathbf{x}_{ij}^0\rvert$	$\mathbf{v}_{i,\theta}(\Delta \mathbf{u}_{i,k}^t)$

Hierarchical Graph. Since the relative motion of the components in the considered experiments is not too large, we define the contact edges based on the initial mesh configuration and keep them fixed to maintain a constant graph. In DEFORMINGPLATE we set the contact radius to $R = 0.1$, connecting actuator nodes with plate nodes. In IMPACTPLATE we connect ball nodes and plate nodes with a radius of $R = 1.2$. In all three experiments, we create $L = 2$ coarse layers to obtain 3 mesh levels.

Algebraic-hierarchical Message Passing Networks. We use 3 Pre- and 3 Post-processing layers, 2 Up- and 2 Downsampling layers and 5 Solving layers, which yields a total number of 15 learnable layers. We add a layer norm before each MLP and use two linear layers, a hidden size of 128 and a Sigmoid Linear Unit (SiLU) [64] activation function. A max aggregation is used in all message passing layers.

Denoising Diffusion Probabilistic Models. We use $K = 20$ denoising steps and a denoising stride of $m = 5$ for ROBIN by default. The β variances of the DDPM scheduler are geometrically spaced for training and inference, starting from a minimum noise variance of $1e-4$ (for β_1) and going up to 1.0 (for β_K).

Metrics. To compare the rollout accuracy, we follow [22] and define the Root Mean Squared Euclidean distance error RMSE $= \sqrt{1/(N_i N_j) \sum_{i=1}^{N_i} \sum_{j=1}^{N_j} (\tilde{u}_{ij} - u_{ij})^2}$, where the prediction $\tilde{u}_{ij}$

and the ground truth u_{ij} have N_i nodes and N_j features. We then calculate the mean over all time steps, the mean over the dataset and finally the mean μ and standard deviation σ over the 5 seeds.

D Baselines, Ablations and Variants

Baselines. We use the official *TensorFlow* [65] implementation of the authors for the baselines HCMT[3] [22] and MGN[4] [5]. We use ADAM [63] for training HCMT and MGN with an exponential learning rate decay from $1e{-}4$ to $1e{-}6$, a batch size of 1 and a hidden size of 128. We use 15 message-passing steps for MGN on all datasets, as well as a total of 15 Hierarchical Mesh Transformer (HMT) and Contact Mesh Transformer (CMT) layers for HCMT.

On BENDINGBEAM, we train HCMT for 4M training iterations with a training noise [4, 5] of 0.001. We maximize the receptive field and set the number of mesh levels to 5, the maximum at which at least five nodes remain available across all meshes in the dataset, required for Delaunay remeshing [66]. This results in 9 HMTs. Since BENDINGBEAM is a contact-free task, we replaced the dual-branch CMT by 6 single-branch Mesh Transformer layers that only attend to mesh edges instead of mesh and contact edges. We use the same architecture and hyperparameter for HCMT on IMPACTPLATE and DEFORMINGPLATE as proposed by the authors [22], and train it for 3M steps and 2M steps, respectively.

We follow the authors' implementation and add world edges to the mesh graph of MGN instead of contact edges to increase the receptive field of the non-hierarchical architecture. MGN is trained for 3M iterations on BENDINGBEAM and uses a training noise of 0.001 with a world edge radius of $R = 0.13$. On IMPACTPLATE we train MGN for 3M steps and use a world edge radius of $R = 0.03$ and a training noise of 0.003. We train MGN on DEFORMINGPLATE for 1.5M steps and use the authors' proposed settings [5]. To prevent out-of-memory errors in edge cases on DEFORMINGPLATE, we restrict the number of world edges to 200,000 by selecting those with the smallest node distances.

For BSMS [48], we use the official *PyTorch* [62] implementation[5] of the authors. We follow the authors and use the maximum number of hierarchy levels possible to maximize the receptive field. We train BSMS on BENDINGBEAM with a batch size of 12 and 6M training samples, i.e., 500K training iterations. We use a training noise of 0.001 and 5 mesh levels. On IMPACTPLATE we train BSMS with a batch size of 8 and 5M samples (625K training iterations), a training noise of 0.003, a contact radius of $R = 0.4$, and 7 hierarchy levels. We train BSMS on DEFORMINGPLATE with a batch size of 8 and 6M samples (750K training iterations), a training noise of 0.003, a contact radius of $R = 0.03$, and 6 hierarchy levels.

Ablations. The *10 diffusion steps* and *5 diffusion steps* ablation use the same settings as ROBIN, despite the reduced number of diffusion steps K. For the *w/o hierarchy* ablation we use the fine mesh graph $\mathcal{G}^0$ instead of the hierarchical graph $\mathcal{G}^{0:L}$, replace our AMPN by a single Intra-MP-Stack with 15 learnable message passing steps and remove the positional level encoding. In addition, we follow MGN and replace contact edges with world edges to increase the receptive field. To stay within the training budget, we reduce the number of training samples to 1.2M for BENDINGBEAM and to 8M for IMPACTPLATE. For DEFORMINGPLATE we reduce the batch size to 1 and the training samples to 0.6M and also restrict the number of world edges to 200,000 as for MGN. The *w/o diffusion* ablation trains the AMPN with an MSE loss to predict directly the displacement residual $\Delta \mathbf{u}^t_{i,0}$ and uses a one-step autoregressive rollout, such as HCMT, MGN, and BSMS. We use the same training noise settings as the baselines to stabilize the rollouts. The *w/o shared layer* ablation uses a total number of 15 non-shared learnable message passing layers distributed as follows: 1 Pre-Processing and 1 Post-Processing layer per level, 1 Up- and 1 Downsampling layer between each level, and 5 Solving layers. The faster predictions allow an increase in the number of training samples to 11M for BENDINGBEAM, to 8M for IMPACTPLATE, and to 4.6M for DEFORMINGPLATE. For the *HCMT model* ablation, we replace the AMPNs with HCMTs. More specifically, we use the same mesh hierarchy and the same model architecture as HCMTs. Everything else remains the same in ROBIN, including DDPMs and ROBI.

[3] `https://github.com/yuyudeep/hcmt/tree/main`

[4] `https://github.com/google-deepmind/deepmind-research/tree/master/meshgraphnets`

[5] `https://github.com/Eydcao/BSMS-GNN/tree/main`

E Results

AMG-based mesh coarsening. Root-node AMG coarsening [53] preserves mesh geometry and connectivity (cf. Figure 10). The coarse mesh remains well-aligned with thin geometrical features, and smoothed transfer operators yield wider, algebraically informed receptive fields compared to bi-stride pooling. This fidelity is crucial for predicting geometrically nonlinear deformations.

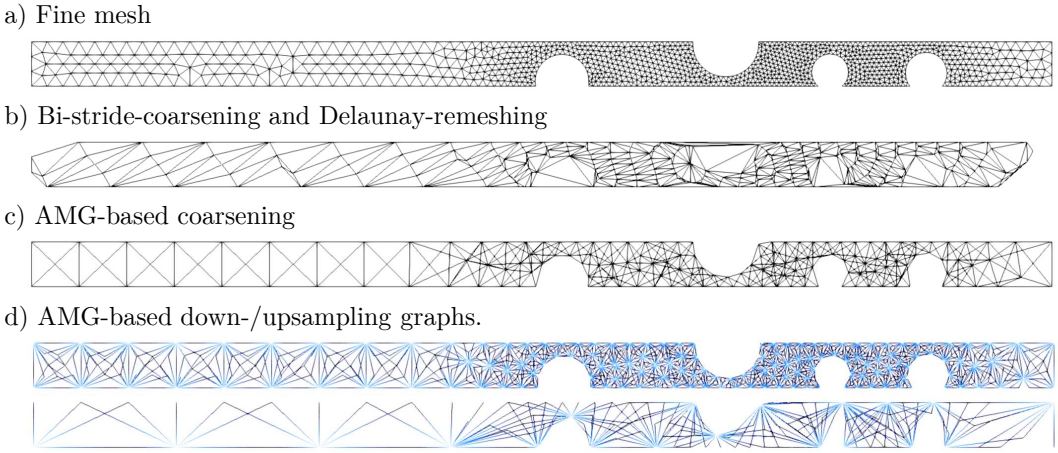
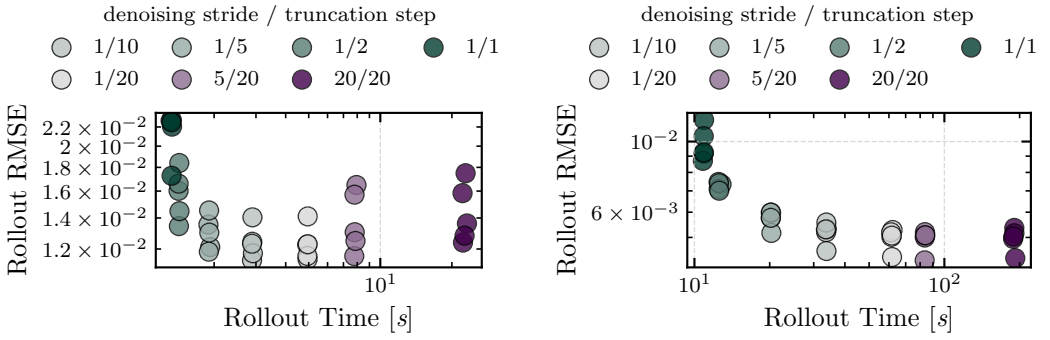


Figure 10: Comparison of AMG-based mesh coarsening to Bi-stride-coarsening and Delaunay-remeshing (BSDL). **a)** the original, fine mesh. **b)** the mesh after two BSDL coarsening steps. **c)** the mesh after one AMG coarsening step. This mesh has approximately as many nodes as the mesh in **b)**. **d)** up- and downsampling edges from level 0 to 1 (top) and level 1 to 2 (bottom). Bright blue indicates coarse nodes of level 1 (top) or 2 (bottom), respectively.

Inference speed. Each point in Figure 11 corresponds to a rollout setting of ROBIN on IMPACTPLATE and DEFORMINGPLATE. As for BENDINGBEAM in Section 5, decreasing the truncation step reduces wall time while a truncation step of k_{tr}=2 is already sufficient to obtain a higher accuracy than all baselines across all datasets (cf. Table 3). The ROBIN default $(1/20)$ is significantly faster than conventional diffusion inference and is even slightly more accurate on IMPACTPLATE, which we attribute to anchoring low-frequency components and reducing drift.



(a) IMPACTPLATE. (b) DEFORMINGPLATE.

Figure 11: RMSE rollout error and inference time of different inference variants of ROBIN on a) IMPACTPLATE and b) DEFORMINGPLATE. ROBIN, i.e., the variant (1/20), is most accurate on IMPACTPLATE and on par on DEFORMINGPLATE with the slower variants (5/20) and conventional inference (20/20). Reducing the truncation step k_{tr} increases speed while decreasing accuracy. The one step variant (1/1) achieves the largest speed-up on DEFORMINGPLATE, but also loses the most accuracy there.

Baselines. Figure 12 and Figure 13 visualize the rollout displacement and von Mises stress prediction of ROBIN, HCMT, MGN, and BSMS on IMPACTPLATE and DEFORMINGPLATE, respectively.

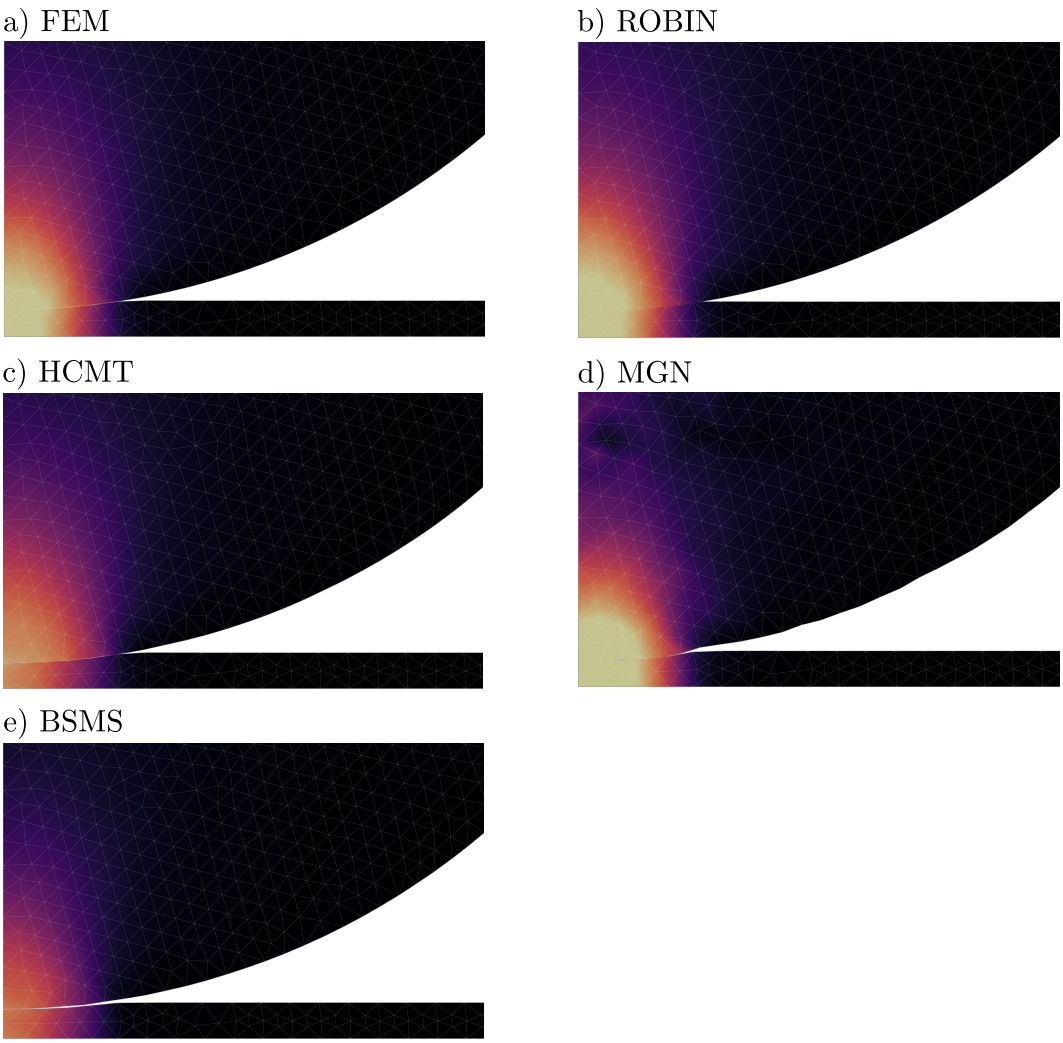


Figure 12: Comparison of the rollout deformation prediction and von Mises stress prediction (color, yellow is high) on IMPACTPLATE to the ground truth of the FEM. ROBIN most accurately resolves the deformation at the contact surface and the resulting stress. The deformation prediction of HCMT and BSMS are close to the FEM prediction, though stress is underestimated. MGN predicts accurately the global modes but exhibits local disturbances.

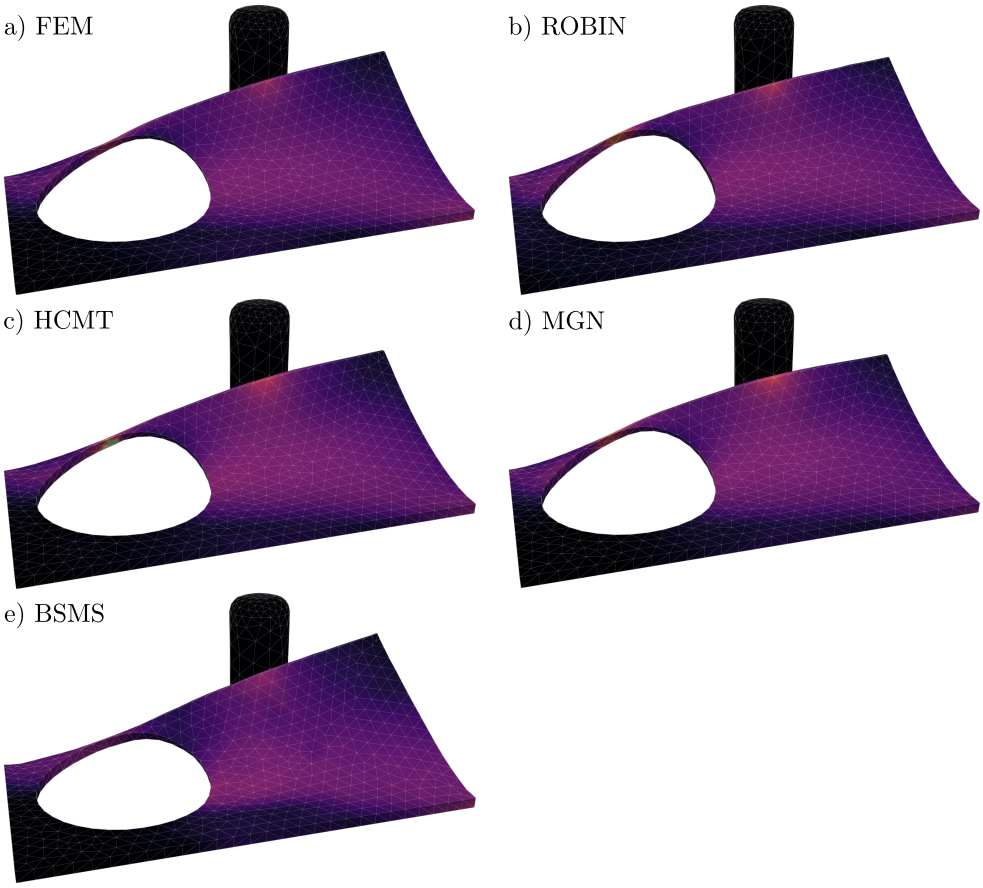


Figure 13: Rollout deformation and von Mises stress prediction (color, yellow is high) on DEFORMINGPLATE of ROBIN, the baselines and the FEM. All models accurately reproduce the part deformation. HCMT slightly overestimates and BSMS slightly underestimates the stress at the thin wall between the hole and the boundary.

Diffusion truncation. The rollouts in Figure 14 across truncation steps k_{tr} illustrate coarse-to-fine frequency behavior of ROBIN. Early steps capture the global, low-frequency shape, whereas additional steps sharpen high-frequency details, e.g., high stresses in thin geometrical features. Even with early truncation, ROBIN maintains coherent global modes. Longer schedules primarily refine local features and minimize the accumulation of high-frequency errors, while retaining the global deformation pattern.

Figure 15 visualize the rollout displacement RMSE and displacement gradient RMSE of different ROBIN variants on DEFORMINGPLATE and IMPACTPLATE. As for BENDINGBEAM, early denoising steps are critical for reducing global displacement error. Later diffusion steps focus on high-frequency solution components. On IMPACTPLATE, the gradient RMSE shows a modest increase while ROBIN attains the lowest displacement RMSE overall. We hypothesize that partially denoising states (small m) stabilizes low-frequency components and reduces drift. In contrast, fully denoising (larger m) suppresses short-term, high-frequency error accumulation. However, this trade-off does not affect other datasets with significantly longer rollouts.

Quantitative results. Table 3 lists the quantitative results of all experiments considered in this work and used for the visualizations.

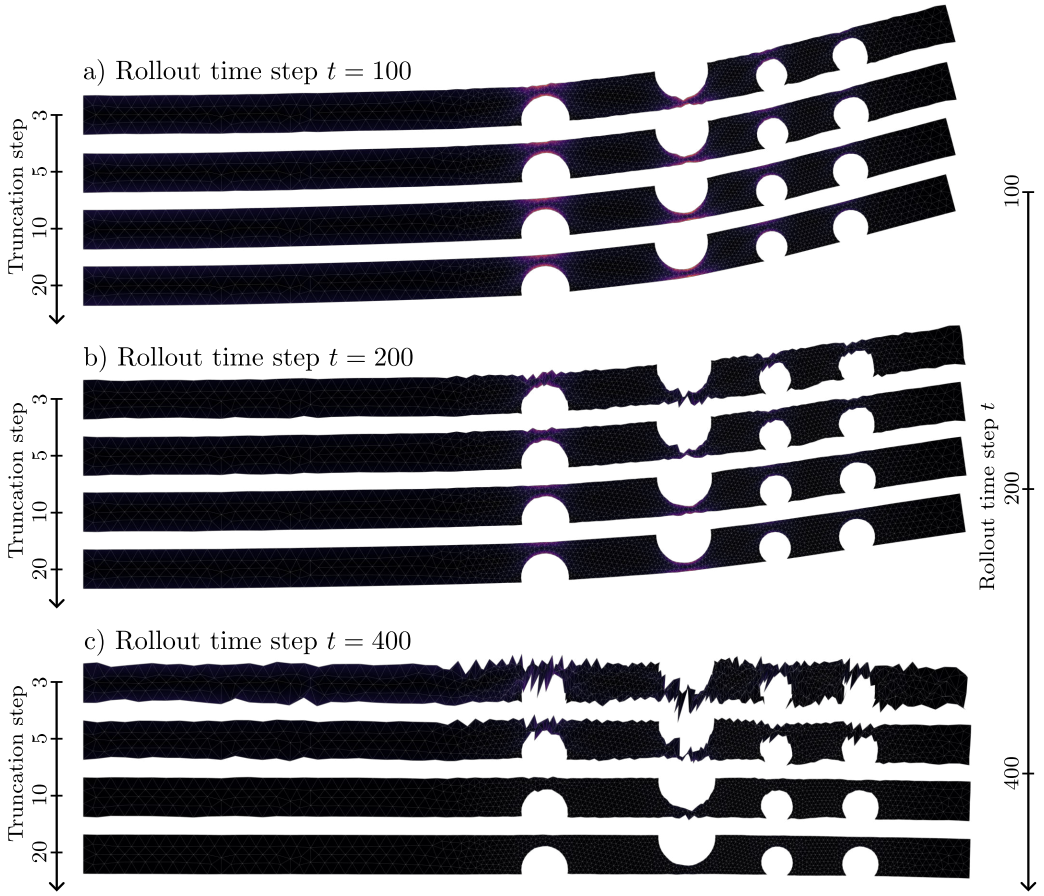
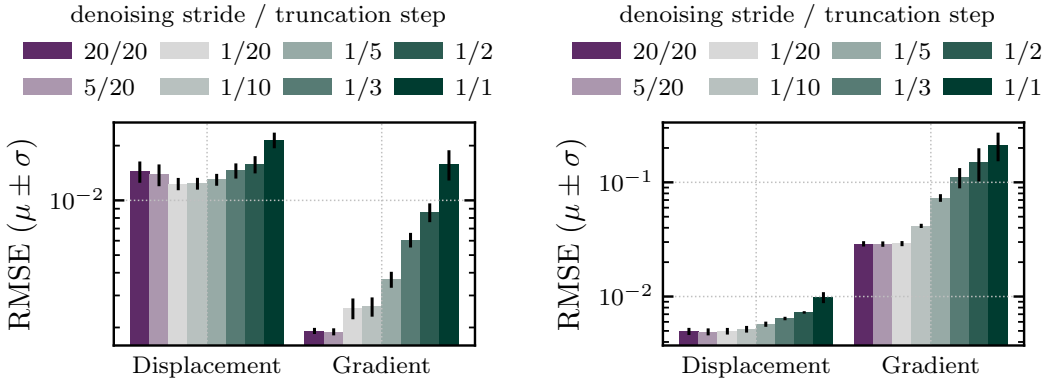


Figure 14: Effect of diffusion truncation on rollouts. Figures a), b), and c) show predicted deformations and von Mises stress prediction (color, yellow is high) at $t = 100$, $t = 200$, and $t = 400$, respectively. In each figure, rows correspond to truncation steps $k_{\text{tr}} = 3, 5, 10, 20$. Using a low truncation step k_{tr} increases inference speed and enables robust predictions of the global deformation modes. However, it also causes local mesh degradation due to the accumulation of high-frequency errors, as observed in MSE trained one step models [9].



(a) IMPACTPLATE. (b) DEFORMINGPLATE.

Figure 15: Comparison of the rollout displacement RMSE and displacement gradient RMSE for different ROBIN setting (denoising stride m / truncation step k_{tr}) on a) IMPACTPLATE and b) DEFORMINGPLATE. Early diffusion steps substantially reduce the displacement RMSE and later steps the local displacement gradient RMSE. The fast inference setting (1/20) remains accurate and yields slightly lower displacement RMSE on IMPACTPLATE, but also slightly higher gradient RMSE.

Table 3: Quantitative results on BENDINGBEAM, IMPACTPLATE, DEFORMINGPLATE, and the large-mesh dataset BENDINGBEAMLARGE. We report displacement RMSE [10^{-3}], gradient RMSE of the displacement field [10^{-3}], and rollout wall-clock time [s]. Values are mean $\pm$ std over the test set across 5 seeds (solver timings excluded). The row *Numerical solving* gives the average runtime of the high-fidelity solver (maximum in parentheses). *Baselines* contrasts different models, *Variants* sweep the ROBI settings (denoising stride / truncation step) and additionally report Gradient RMSE, *Ablations* remove ROBIN components, and *Generalization to large meshes* evaluates the upscaling capabilities of ROBIN.

	BENDINGBEAM		**IMPACTPLATE**		**DEFORMINGPLATE**	
Numerical solving	Time [s]		Time [s]		Time [s]	
	46.2 (max. 186.1)		742.6 [22]		1157.2 [5]	
Baselines						
	RMSE [10^{-3}]	Time [s]	RMSE [10^{-3}]	Time [s]	RMSE [10^{-3}]	Time [s]
ROBIN	29.00 $\pm$ 1.00	15.03 $\pm$ 0.04	12.33 $\pm$ 0.96	4.94 $\pm$ 0.01	4.98 $\pm$ 0.33	61.60 $\pm$ 0.29
HCMT	121.53 $\pm$ 1.87	25.82 $\pm$ 0.28	19.57 $\pm$ 0.38	4.33 $\pm$ 0.05	8.04 $\pm$ 0.13	31.57 $\pm$ 0.14
MGN	189.52 $\pm$ 70.87	21.40 $\pm$ 0.15	54.07 $\pm$ 5.88	2.93 $\pm$ 0.03	8.76 $\pm$ 0.29	24.42 $\pm$ 0.22
BSMS	141.98 $\pm$ 7.96	4.70 $\pm$ 0.04	63.52 $\pm$ 32.36	2.16 $\pm$ 0.01	13.59 $\pm$ 5.21	13.28 $\pm$ 0.07
Variants (denoising stride / truncation step)						
	RMSE [10^{-3}]	Time [s]	RMSE [10^{-3}]	Time [s]	RMSE [10^{-3}]	Time [s]
20/20	28.90 $\pm$ 1.91	162.43 $\pm$ 0.89	14.43 $\pm$ 1.93	22.61 $\pm$ 0.36	4.96 $\pm$ 0.36	190.10 $\pm$ 1.33
5/20	28.99 $\pm$ 1.93	43.39 $\pm$ 0.52	13.87 $\pm$ 1.90	7.83 $\pm$ 0.06	4.92 $\pm$ 0.35	83.61 $\pm$ 0.16
1/20	29.00 $\pm$ 1.00	15.03 $\pm$ 0.04	12.33 $\pm$ 0.96	4.94 $\pm$ 0.01	4.98 $\pm$ 0.33	61.60 $\pm$ 0.29
1/10	30.35 $\pm$ 2.41	11.03 $\pm$ 0.10	12.38 $\pm$ 0.92	2.90 $\pm$ 0.01	5.18 $\pm$ 0.35	33.71 $\pm$ 0.10
1/5	35.81 $\pm$ 3.30	9.88 $\pm$ 0.19	13.01 $\pm$ 0.98	1.91 $\pm$ 0.01	5.73 $\pm$ 0.30	20.24 $\pm$ 0.04
1/3	42.83 $\pm$ 4.54	9.57 $\pm$ 0.11	14.58 $\pm$ 1.40	1.55 $\pm$ 0.01	6.42 $\pm$ 0.23	14.85 $\pm$ 0.11
1/2	48.94 $\pm$ 7.25	9.47 $\pm$ 0.11	15.79 $\pm$ 1.71	1.42 $\pm$ 0.00	7.26 $\pm$ 0.18	12.59 $\pm$ 0.13
1/1	64.23 $\pm$ 11.86	9.49 $\pm$ 0.12	21.46 $\pm$ 2.12	1.32 $\pm$ 0.01	10.79 $\pm$ 0.52	7.11 $\pm$ 0.04
	Gradient RMSE [10^{-3}]		Gradient RMSE [10^{-3}]		Gradient RMSE [10^{-3}]	
20/20	12.75 $\pm$ 0.70		1.92 $\pm$ 0.07		28.98 $\pm$ 1.56	
5/20	12.74 $\pm$ 0.65		1.89 $\pm$ 0.08		28.83 $\pm$ 1.57	
1/20	12.77 $\pm$ 0.72		2.55 $\pm$ 0.33		29.10 $\pm$ 1.50	
1/10	37.58 $\pm$ 4.71		2.61 $\pm$ 0.30		41.59 $\pm$ 1.76	
1/5	94.00 $\pm$ 20.90		3.68 $\pm$ 0.37		73.13 $\pm$ 5.64	
1/3	147.97 $\pm$ 65.13		6.07 $\pm$ 0.56		110.96 $\pm$ 22.33	
1/2	200.44 $\pm$ 128.29		8.61 $\pm$ 1.02		150.33 $\pm$ 48.48	
1/1	324.05 $\pm$ 285.65		23.41 $\pm$ 4.85		157.30 $\pm$ 8.34	
Ablations						
	RMSE [10^{-3}]		RMSE [10^{-3}]		RMSE [10^{-3}]	
ROBIN	29.00 $\pm$ 1.00		12.33 $\pm$ 0.96		4.98 $\pm$ 0.33	
10 diff. steps	33.28 $\pm$ 2.43		14.58 $\pm$ 1.99		5.86 $\pm$ 0.61	
5 diff. steps	41.33 $\pm$ 6.27		15.60 $\pm$ 1.74		6.40 $\pm$ 0.56	
w/o shared layer	66.05 $\pm$ 7.36		12.88 $\pm$ 1.58		5.65 $\pm$ 0.26	
State prediction	596.38 $\pm$ 70.93		130.50 $\pm$ 30.80		137.61 $\pm$ 30.54	
w/o hierarchy	122.38 $\pm$ 1.52		57.02 $\pm$ 3.48		12.04 $\pm$ 0.24	
w/o diffusion	43.83 $\pm$ 1.16		99.47 $\pm$ 47.30		9.78 $\pm$ 2.12	
HCMT model	111.05 $\pm$ 1.15		29.18 $\pm$ 1.43		14.86 $\pm$ 0.42	

	BENDINGBEAMLARGE	
Numerical solving	Time [s]	
	108.30 (max. 4248.01)	
Generalization to large meshes		
	RMSE [10^{-3}]	Time [s]
Pre-trained	77.59 $\pm$ 4.97	30.94 $\pm$ 0.63
From scratch	215.50 $\pm$ 12.14	30.95 $\pm$ 0.58

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The claims in the abstract and introduction are fully supported by our method section, as well as in the qualitative and quantitative results in the experiments section. The appendix provides more detailed results where required.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The conclusion discusses current limitations of the approach, including the scope of the paper and assumptions made.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA]

Justification: The paper does not introduce any new theoretical results.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We detail our method in Section 3, and provide additional information in the appendix where required. We detail our experimental setup and datasets used in the experiments section.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [No]

Justification: We do not provide data and code at the time of submission, but will open-source both after acceptance.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: We specify high-level training and test details in Section 4, and provide further details in Appendices D, B and C.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We report mean and standard deviation across five seeds for all main experiments. When evaluating the runtime-error tradeoff, we directly report information for all seeds without aggregation, explicitly showing the performance difference between runs.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We provide information on the compute resources for all experiments in Appendix C.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We have read the NeurIPS Code of Ethics. We made sure that our research complies to the Code of Ethics in every respect.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We include a discussion on broader impact in Appendix A.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.

- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: We do not use pretrained language models, image generators or similar high-risk models in our approach, and do not scrape datasets. We still discuss potential cases for miss-use of our learned simulator in Appendix A.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We use two publicly available datasets, namely ImpactPlate and Deforming-Plate. We credit the original papers of both explicitly.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.

- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: At time of submission, we do not release new assets. After acceptance, we will open-source our BendingBeam dataset, and provide appropriate documentation alongside it.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: We do not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [NA]

    Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.