PHYSIFORMER: Learning to Simulate Mechanics in World Space

Yiming Chen Yushi Lan Andrea Vedaldi
Visual Geometry Group, University of Oxford
{yiming,yushi,vedaldi}@robots.ox.ac.uk

Abstract

We present PHYSIFORMER, a diffusion transformer for physically-plausible 3D object motion. Unlike video world models that operate in view-dependent pixel space, PHYSIFORMER represents objects as 3D meshes expressed in world coordinates. Given the initial vertex positions and velocities, as well as object material type, rigid or elastic, the model samples future vertex trajectories. While related neural physics approaches build on ad-hoc latent spaces or explicitly enforce rigidity and causality, PHYSIFORMER shows that excellent results can be obtained without any such inductive biases, by casting vertex trajectory prediction as a single denoising diffusion process directly in world coordinates. The probabilistic formulation captures uncertainty in the learned dynamics, enabling diverse plausible futures from initial conditions, making this framework potentially useful for applications with unobserved uncertainty. The model features attention factorised over time, space, and objects for efficiency, enabling permutation-invariant multi-object reasoning without needing explicit object encoding. Trained on over 100k simulated trajectories, PHYSIFORMER generates rigid and elastic mechanics, and generalises to mixed-material settings, unseen real-world geometries, and larger object counts. It substantially outperforms autoregressive baselines in trajectory accuracy, rigidity preservation, and momentum-based physical consistency. Our results position coordinate-space diffusion as a promising step toward view-invariant, geometry-aware world modelling for robotics, graphics, and physical design. Visualisations, code, and models are available at https://yimingc9.github.io/physiformer.

1 Introduction

Predicting how physical systems evolve over time is central to spatial intelligence and world modelling and has applications to robotics, computer graphics, engineering, and more. Motivated by large-scale video datasets and the success of diffusion transformer architectures [44], several authors [11, 43] have sought to reduce the problem of modelling physical systems to video generation. However, a video is a fairly indirect and ambiguous representation of the state of a physical system, as it intertwines geometry and motion with viewpoint, lighting, and occlusion.

Here we reformulate the problem directly in the space of 3D models, similar to physics simulators like MuJoCo [53] and PyBullet [16]. We are inspired by RenderFormer [60], which models light transport over triangle meshes, and ask whether a neural network can similarly predict mechanical dynamics using 3D meshes as a representation of the geometry of the system. We choose 3D meshes because of their ubiquity in applications, making this investigation directly relevant to them.

To answer our question, we introduce PHYSIFORMER, a diffusion transformer that takes as input 3D mesh vertices representing one or more objects and per-vertex initial velocities. As shown in fig. 1,

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

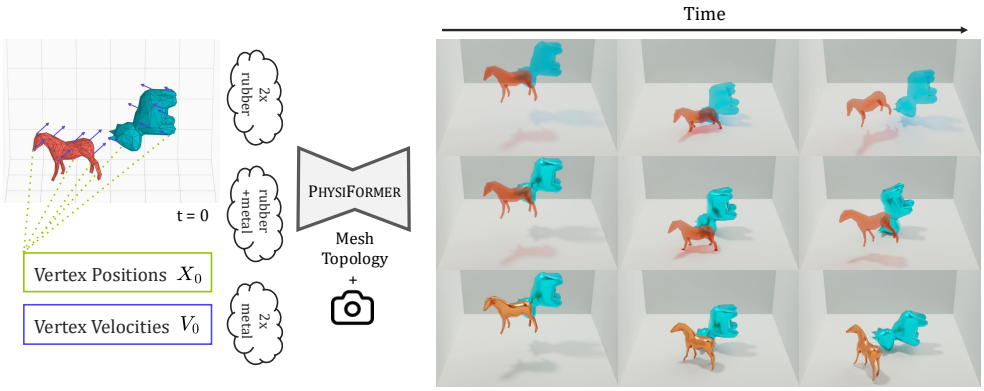


Figure 1: **PHYSIFORMER overview.** Given initial per-vertex positions $X_0 \in \mathbb{R}^{N\times 3}$ and velocities $V_0 \in \mathbb{R}^{N\times 3}$, and material conditions of (1) rigid, (2) deformable, or (3) mixed, PHYSIFORMER jointly generates full-sequence future vertex trajectories through a single diffusion sampling process, producing physically plausible multi-object dynamics, with face connectivity used only at inference. Outputs can be rendered as 4D meshes with arbitrary textures, lighting, and views.

PHYSIFORMER generates the trajectory of mesh vertices over a temporal window, approximating how the system should behave under the laws of mechanics and capturing properties such as inertia, gravity, and collisions. We also attach different material properties to the objects, allowing them to be either rigid or elastic.

We build our model on top of a general-purpose Diffusion Transformer (DiT) [44], conditioned on the initial state of the system to generate the system's future as a sample from an underlying distribution of physically plausible trajectories. While the architecture is mostly generic, we introduce small but effective modifications to better suit the problem of modelling mechanics. In particular, we factorise self-attention across time and space to improve the model's efficiency, as well as objects, which implicitly encodes the existence of separate objects in the scene without the need to introduce special tokens or embeddings, and in a permutation-invariant manner. Unlike recent approaches [15], we perform diffusion in *raw 3D coordinate space*—without the added complexity of learning latent features via autoencoders—following the Just image Transformers (JiT) [32] framework.

We train our model on simulated scenes containing either all rigid or all elastic objects undergoing collisions and complex motion. Once trained, the model generalises to novel object combinations, geometries, dynamic and stationary configurations, mixed-materials, and larger object counts.

A key feature of our model is that it generates the entire trajectory of the system in one go. This is a departure from the autoregressive (AR) approach often used to model physics [49, 47, 34, 64], where the model is trained to predict the next state of the system given the current state. AR models are motivated by mechanics, where, under mild assumptions, instantaneous vertex positions and velocities fully characterize the physical state and thereby define a Markovian system. In fact, authors have applied this AR assumption with some success even when the state is only partially observable in a feature space [3, 28].

In our experiments, we compare our approach against AR alternatives. We show that PHYSIFORMER obtains far better performance. For one, this is due to a mismatch between training and testing statistics for AR frameworks, which can be mitigated but not removed entirely by Diffusion Forcing [13] and Self Forcing [24]. Moreover, this is due to irreducible error accumulation, which, for instance, causes the shape of rigid objects to gradually deform over time. In practice, AR modelling works well for particle systems [49, 47, 34, 64], where maintaining the long-term coherence of 3D shape is irrelevant, or in models that hard-code assumptions such as rigidity [2, 46], which makes them far less generalisable (for example unable to model elastic objects).

To summarise, our contributions are fourfold: (1) we introduce a unified neural model that simulates mechanical systems with both rigid and elastic objects directly in 3D trajectory space; (2) we demonstrate that a general-purpose diffusion transformer with factorised attention over time, space, and objects can model such dynamics effectively; (3) we show that generative modelling produces diverse plausible futures under incomplete conditioning, without explicitly specifying all simulator parameters; and (4) we find that autoregressive approaches with the same input formulation are

limited by error accumulation, or require ad-hoc design choices to compensate for this. Our findings suggest that spatial intelligence can be supported effectively on top of 3D meshes, which may have direct applications to robotic simulation, gaming, content creation, and engineering.

2 Related Work

Traditional Physical Simulation. The world is governed by physical laws, and modelling system behavior via physics-based simulation has long been a central goal in engineering. Substantial progress has been made in motion and material modelling [42, 39, 55], contact resolution [10, 31], and time integration [5], spanning discrete and continuum representations for structures such as cloth, thin shells, and fluids. Despite their strong physical grounding and realism, these approaches remain computationally expensive, algorithmically complex, and difficult to generalize, in part due to domain-specific optimisation constraints.

Per-scene Optimized Physical Dynamics. Recent work couples 3D scene representations (e.g., NeRFs [40] and 3D Gaussian splats [29]) with learned dynamics or inverse-physics optimization, typically overfitting per scene to recover deformable object motion from videos [64, 58, 65, 25]. While these methods enable joint reasoning over geometry, appearance, and motion, they often require dense multi-view capture or tracking supervision, and remain limited by strong simulator assumptions (e.g., spring-mass models, simplified contact).

Learning-based Physics Simulation. Developments in deep learning have ushered in data-driven approaches to physical simulation. Convolutional networks have been effective for learning physics on regular grids [21, 7]. To model more general dynamics, graph neural networks (GNNs) [48] provide a natural framework for particle-based simulation. Particle states are represented as node features, interactions are encoded by edges, and the system is updated by message-passing conditioned on the previous state [6, 41, 47, 34, 64, 20]. DPI-Net [33] builds interaction graphs dynamically with hierarchical propagation to handle rigid bodies, deformable objects, and fluids, while Transformer with Implicit Edges (TIE) [49] replaces edge-based message passing with attention. Several works further adapt neural simulation to mesh-based representations: MeshGraphNets [45] extend message passing to mesh discretizations, FIG-Net [2] and HopNet [57] introduce face-based and higher-order interactions for contact-rich dynamics, and HCMT [59] combines hierarchical mesh structures with Transformer modules for flexible-body collisions. This line of work shows the value of mesh-aware inductive biases, but often requires additional geometric machinery such as explicit connectivity, topology preprocessing, hierarchical structures, or learned shape representations as in SDF-Sim [46]. PhysCtrl [56] generates physics-grounded 3D point trajectories first conditioned on material parameters and applied forces, then uses them to guide controllable image-to-video synthesis. In contrast, PHYSIFORMER models dynamics as diffusion over raw 3D vertex trajectories, supporting multiple objects and materials without specialized contact modeling.

Autoregressive Prediction in Visual Feature Spaces. As noted in the previous section, deterministic autoregressive models have become widely adopted for future prediction. This pipeline can be applied to the Vision Foundation Model (VFM) feature space. Given four context frames, DINO-Foresight [28] predicts latent futures that can be decoded into interpretable outputs such as depth and segmentation, demonstrating the practical effectiveness of autoregressive prediction.

Diffusion Models for World Simulation. Video Diffusion Models (VDMs) [11, 8] are rapidly advancing visual content generation to convincingly simulate the physical world. However, recent analyses find that VDM outputs frequently violate Newtonian dynamics [30] and do not learn physical laws from videos alone [27], limiting their suitability for settings like robotics where physical fidelity is critical. To address this, several works inject explicit physics signals into VDMs, enabling force-conditioned image-to-video generation with strong generalization [19, 36, 14]. While effective, these approaches require objective-specific training. Moreover, diffusion in pixel space is inherently view-dependent, which complicates enforcing viewpoint-invariant physical consistency. Motivated by this limitation, prior work explores diffusion over 3D representations [38, 62], but these methods are largely restricted to generating single, static objects. More recent efforts add dynamics [63, 15], yet they remain focused on single-object settings and leverage motion in videos for animation.

3 Method

We consider the problem of animating a 3D scene in a physically-plausible manner. We model the scene as a triangular mesh, with N vertices total, given by positions $X(t) \in \mathbb{R}^{N\times 3}$ indexed by time $t \in \mathbb{R}$, and faces $\mathcal{F} \subset \{1, \dots, N\}^3$, defined by triplets of vertex indices. A scene can contain several objects, modelled as different connected components of the mesh. We denote the instantaneous velocities of the vertices as $V(t) = dX/dt|_t \in \mathbb{R}^{N\times 3}$. Given the initial scene configuration, $X(0)$ and $V(0)$, our goal is to draw a sample from a stochastic process $(X(t))_{t>0}$ that is consistent with the underlying physics of the scene. We aim to learn a model of this process from data.

We also assume that, given sufficient initial information, the data can be generated deterministically by a physics simulator (e.g., Genesis [4]). This means that there exists a function $\mathcal{S}$ which takes in the state of the system $(X(t), V(t))$ at time t and contextual information Y independent of time to output the future trajectory $(X(t'), V(t'))_{t'\geq t} = \mathcal{S}(X(t), V(t)|Y)$. Here, Y specifies all properties sufficient to carry out the simulation, such as the mesh topology, gravity, friction and restitution.

Under these assumptions, the dynamics are *Markovian*, which makes autoregressive next-state prediction a natural modelling choice. However, in practice, it is challenging for a model to learn all relevant state and contextual information, and the Markov assumption is only approximately satisfied. In our experiments we pass to the model some material properties $y \subset Y$ of the objects, but leave other constant properties implicit, to be determined during learning.

Partially because of this, we find that one-shot trajectory generation is substantially more effective than autoregressive rollout. Another benefit of such a model is to avoid exposure bias due to the mismatch between teacher-forced training and autoregressive rollout at test time.

Discretizing time. To simplify modelling with little loss of generality, we discretize time, and consider trajectories $X, V \in \mathbb{R}^{T\times N\times 3}$ with T steps $t \in \{1, \dots, T\}$. We use the symbols $X_t = X(t)$ and $V_t = V(t)$ to denote the t-th slices of these tensors. Our goal is to learn the conditional trajectory distribution $p(X \mid X_0, V_0, y)$. We compare that to autoregressive methods that model the transition $p(X_{t+1}, V_{t+1} \mid X_t, V_t, y)$ and roll it out iteratively. Note that autoregressive generation requires predicting both X and V because velocity is part of the Markov state, whereas our one-shot formulation predicts X directly (alternatively, V_t can be approximated as $V_t \propto X_t - X_{t-1}$).

3.1 PHYSIFORMER

In order to model the distribution $p(X|X_0, V_0, y)$, we propose PHYSIFORMER (fig. 2), a diffusion model that uses a JiT [32]-style objective and a Diffusion Transformer (DiT) [44] backbone. While we make a point to use a general-purpose architecture, we make minimal modifications to capture the structure of our data. Instead of introducing object-identity embeddings to distinguish tokens that belong to different objects, we interleave global and per-object spatial attention to make the model object-aware while also being invariant to the order or identity of the objects. By denoising full trajectories as a single prediction target, PHYSIFORMER infers positions jointly across time, vertices, and objects, promoting globally consistent generated motion.

Diffusion Model. We briefly summarise the diffusion framework we use. Let $x \sim p(x)$ be a random vector from the target data distribution, and let $\epsilon \sim \mathcal{N}(0, I)$ be a normal noise vector of the same dimension. During training, we construct a noised sample $z_\tau = \tau x + (1-\tau)\epsilon$ by mixing data and noise according to $\tau \in [0, 1]$ [35, 37, 1]. Thus, z_τ is pure noise for $\tau = 0$ and follows the data distribution for $\tau = 1$. The flow velocity $v(z_\tau, x, \tau) = dz_\tau/d\tau = (x - z_\tau)/(1-\tau)$ is the derivative of z_τ with respect to τ (not to be confused with the vertex velocity V above). The model is trained to predict the marginal velocity $v(z_\tau, \tau) = \mathbb{E}_{x,\epsilon}[v(z_\tau, x, \tau) \mid z_\tau]$. For this, we train a neural network $x_\theta(z_\tau, \tau)$ with parameters θ expressing the flow velocity as $v(z_\tau, \tau) = v(z_\tau, x_\theta(z_\tau, \tau), \tau)$ and minimising the loss

$$\mathcal{L}(\theta; \tau) = \mathbb{E}_{x,\epsilon} \left\| v(z_\tau, x_\theta(z_\tau, \tau), \tau) - v(z_\tau, x, \tau) \right\|^2 = \mathbb{E}_{x,\epsilon} \left\| \frac{x_\theta(\tau x + (1-\tau)\epsilon, \tau) - x}{1-\tau} \right\|^2. \quad (1)$$

During training, we sample τ from a logit-normal distribution $\text{logit}(\tau) \sim \mathcal{N}(\mu, \sigma^2)$, where $\mu = -0.8$ and $\sigma = 0.8$. At inference time, generation proceeds by integrating the corresponding ordinary differential equation (ODE), $dz_\tau/d\tau = v_\theta(z_\tau, \tau)$, from $\tau = 0$ to 1 to obtain a sample $z_1 \approx x$. In practice, we solve the ODE numerically with the Heun integrator with 50 steps.

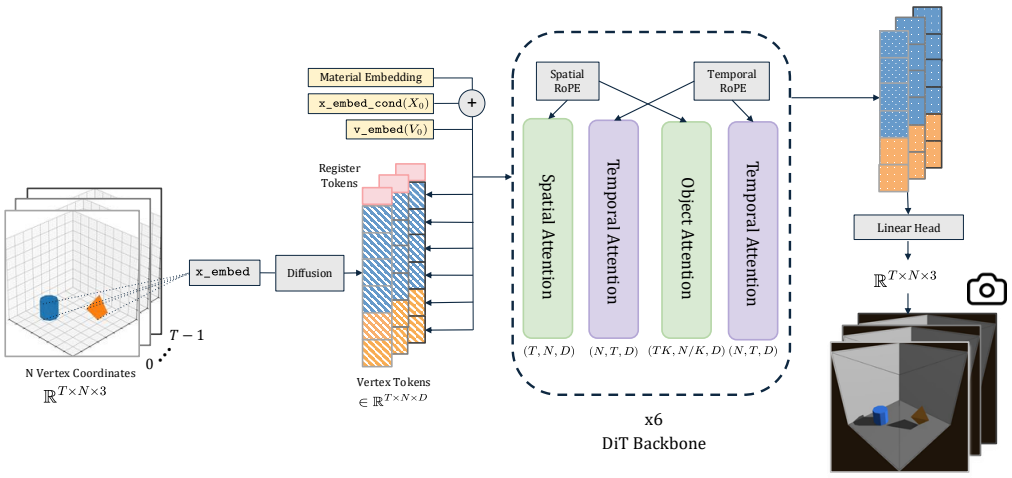


Figure 2: **PHYSIFORMER Architecture**. During training, input mesh vertex coordinates in $\mathbb{R}^{T\times N\times 3}$ are projected into hidden dimension $D = 1024$ via a linear embedder `x_embed`, and diffused with noise according to the flow-matching schedule. Each noised vertex token is additively conditioned on first-frame position and velocity embeddings (via separate `x_embed_cond` and `v_embed`) and a material embedding. We use 16 prepended global register tokens to aggregate context across the factorized DiT-L backbone with 4×6 layers. The tokens are replicated and consolidated over time, vertices, and objects for spatial, temporal, and object-level attention, respectively, each using its corresponding RoPE. At inference, iterative denoising produces clean vertex trajectories, which are projected back to coordinate space via a linear head and assembled into triangle meshes using the provided face connectivity for view-invariant rigid and elastic 4D mesh rendering.

Because the network outputs the "clean" data x while minimising the velocity prediction error, this is called x-prediction with v-loss. As suggested by JiT [32], x belongs to a lower-dimensional data manifold than v (see the Manifold Assumption [12]), which simplifies prediction.

3.2 Diffusion Architecture

Our goal is to model the distribution $p(X|X_0, V_0, y)$ of vertex trajectories conditioned on initial position and velocity, and contextual information y about the scene. We thus design a neural network that, given X_0, V_0, y and a noised version z of the data X, denoises it to predict the "clean" data X, i.e., $X \approx x_\theta(z_\tau, X_0, V_0, y, \tau)$. We base our model on general-purpose Diffusion Transformer (DiT) [44] with modifications that capture the structure of our data, as explained below.

Encoding. The transformer requires the data to be converted to a stream of tokens. To this end, the noisy vertices in $z \in \mathbb{R}^{T\times N\times 3}$ are individually projected to a D-dimensional space using a linear layer and the result is flattened to a sequence of TN D-dimensional tokens.

The network is conditioned on the initial states $X_0, V_0 \in \mathbb{R}^{N\times 3}$, which are projected to dimension D using separate embedding functions to distinguish initial conditions from model input. The resulting $N \times D$ tensors are broadcast-summed to the vertex embeddings.

Additionally, we condition the model on two types of object materials by adding material embeddings via a two-layer $\mathrm{MLP}_{\mathrm{mat}}$. Rigid and elastic materials are embedded with inputs 0 and 1 respectively, assigned to objects and broadcast to vertices for efficiency. The success of this simple conditioning in PHYSIFORMER suggests its capability to model various physical properties with explicit conditioning in a similar fashion. The result of the encoder is thus a sequence z' of TN tokens, each of dimension D, encoding z, X_0, V_0 and the material properties in y.

Structured Attention. The sequence z' is processed by a stack of DiT blocks. By themselves, these would operate using self-attention on the entire sequence of length TN, with cost $\mathcal{O}(T^2N^2)$. We

suggest instead structuring attention for efficiency, and also use it to implicitly encode information in the model. As done in prior works [8, 9], we employ alternating spatio-temporal attention; our novelty is further factorising spatial attention into full spatial and object-level attention. Concretely, for spatial attention, the TN tokens are reshaped as (T, N, D) treating T as batch dimension, so attention is applied independently within each of the T frames. For object-level attention, padding the K objects so that they have an equal number of vertices N/K, tokens are grouped by object as $(TK, N/K, D)$, so attention is applied within each object at each time step. For temporal attention, tokens are reshaped as (N, T, D), so each vertex attends across time. This factorisation reduces the attention cost to $\mathcal{O}(TN^2 + NT^2)$. The interplay between local and global spatial attention makes the model aware of the different objects without using explicit object identifiers and in a manner that is insensitive to the order of the objects in the token sequence.

Spatial and Temporal Rotational Positional Encoding. We inject spatio-temporal position information into the transformer blocks with rotary positional encodings (RoPE) [52], which naturally encode relative information consistent with the global spatio-temporal translation invariance of physical dynamics. RoPE is applied separately in temporal and spatial attention: temporal attention uses standard 1D RoPE over time indices, while full and object-level spatial attention use coordinate-conditioned RoPE, following RenderFormer [60], so attention depends on relative 3D vertex offsets. Specifically, we multiply each of x, y, z by log-spaced base-2 frequencies, concatenate the resulting phases, and convert them to $\sin$ / $\cos$ coefficients for the standard block-wise 2×2 RoPE rotations of query and key vectors. If the number of coordinate-derived phases differs from the per-head rotary dimension, we pad or truncate them; zero-padding gives identity rotations, leaving the remaining channel pairs unrotated. Sixteen shared register tokens are replicated and consolidated across factorised attention to aggregate global information, with details in section A.1.

4 Experiments

We evaluate the ability of PHYSIFORMER to simulate the dynamics of rigid and deformable objects.

4.1 Dataset

We create a synthetic dataset with the Genesis physics simulator [4]. Each scene is represented as $(X_0, V_0, X, \mathcal{F}, M)$, where $X_0, V_0 \in \mathbb{R}^{N\times 3}$ are initial vertex positions and velocities, $X \in \mathbb{R}^{T\times N\times 3}$ is the ground-truth vertex trajectory, $\mathcal{F} \in \mathbb{Z}^{N_f\times 3}$ is the triangular face connectivity with N_f faces, and $M \in \{0, 1\}^N$ denotes per-vertex material, rigid or elastic. Objects correspond to connected mesh components with a shared material; the initial velocities are sampled per object and converted to per-vertex velocities; the elastic material is defined to have a fixed Young's modulus.

We generate 49-frame trajectories in a bounded $[-1, 1]^3$ container with timestep $\Delta t = 1/240$s. Across scenes, we randomize the number of objects, object sizes, shapes, materials, and initial conditions, while keeping density and environmental parameters fixed. We choose simulator parameters to make collisions as elastic as the simulator allows: friction is minimized, while the simulator's standard damping is retained for numerical stability. As a result, collisions are near-elastic but not perfectly energy-conserving.

We generate four datasets of increasing complexity, summarized in table 1. Mesh templates used in training are visualised in fig. 6 in the appendix. All scenes are simulated in the same bounded environment, where objects may collide with each other and the box walls. In the floor-start rigid settings, objects are placed on the floor without overlap and assigned random initial linear velocities, with zero velocity for a random subset. A small 2° orientation jitter allows initially stationary but unstable objects to fall under gravity. The airborne-start setting follows the same setup but with

Table 1: Summary of the four datasets used in our experiments.

D_1	10k rigid floor-start scenes with 1–5 convex objects from 15 templates ((a) in fig. 6). Each object has 4–20 vertices, with at most 88 vertices per scene.
D_2	15k rigid floor-start scenes with 1–5 objects from 25 convex and 10 concave templates ((a) and (b) in fig. 6). Objects have 4–88 vertices, with at most 356 vertices per scene.
D_3	60k airborne-start rigid scenes: 35k with 1–5 objects and 25k with 6–10 objects. In each object count group, 10k scenes include nonzero initial angular velocity. Objects are selected from the same mesh templates as D_2.
D_4	20k elastic scenes with 1–5 objects. 10k floor-start and 10k airborne-start. Objects are selected from the same mesh templates as D_2.

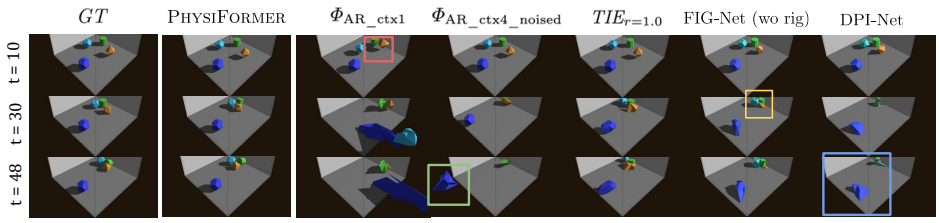


Figure 3: **Qualitative comparison of PHYSIFORMER against autoregressive baselines on trained 10k rigid object data.** At $t = 10$, rigidity is not preserved in $\Phi_{\text{AR_ctx1}}$, but objects remain rigid across all other models. As t increases, all autoregressive baselines diverge due to error accumulation: objects penetrate each other, escape the bounding box, and deform severely. PHYSIFORMER consistently maintains object rigidity and produces physically plausible long-horizon motion.

objects spawning in the air. Train/validation/test splits are precomputed using a fixed-seed stratified split to match the data distribution. All training scenes contain a single material type.

4.2 PHYSIFORMER Implementation Details

PHYSIFORMER is trained from scratch following JiT's framework in section 3.1. The model predicts vertex positions, which are combined with the original face connectivity for rendering during inference. We sample noise as $\epsilon \sim \mathcal{N}(0, I) \times \texttt{noise_scale}$, where $\texttt{noise_scale} = 0.1$, and analyze this choice in section 4.8. Optimization uses AdamW, and we maintain an exponential moving average (EMA) of parameters with decay 0.9999. Training uses automatic mixed precision (AMP) with bf16. We use PyTorch Flash SDPA for speed and compatibility with input-dependent masking. We train on 2 NVIDIA H100 GPUs with 94GB memory, with an effective batch size of 64 and $lr = 4\text{e}{-5}$.

For comparison with AR baselines, we train a model using the DiT-L backbone on D_1 for 70k iterations, denoted as PHYSIFORMER-L-10k. We employ linear learning-rate warm-up over 780 steps, followed by a cosine decay schedule toward $lr = 5\text{e}{-6}$. We train PHYSIFORMER by finetuning PHYSIFORMER-L-10k on rigid dynamics in the $D_1 + D_2 + D_3$ rigid dataset for 27k iterations, and then further finetune on elastic object motions in D_4 for 12k iterations. For the latter, we ensure that the model sees a 60/40 ratio of rigid to elastic material scenes. Inference uses EMA weights, and we clamp first-frame positions to X_0 during sampling. We use 50 Heun sampling steps for all sampled results in the main paper, though we show that fewer steps yield comparable performance in section B.

4.3 Baselines

Graph Network-Based Autoregressive Methods. TIE [49] is an autoregressive transformer that uses implicit edges defined by tokens in attention to mimic a graph neural network for next-timestep system-state prediction. In particle-based dynamics prediction, TIE outperforms established GNN baselines [33, 47], continuous convolution methods [54], and transformer-based models with explicit edge encoding [18]. Our training setup is analogous to a particle-based system, as we use mesh faces only at inference time. Therefore, we use TIE as a strong baseline. For fair comparison, we treat each vertex as a separate particle without hard-coding rigidity like in TIE's setup as discussed in section 4.4. TIE uses attention to model implicit edges between particles within radius r. Given the bounding box and range of motion in our dataset, we train the model with $r = 0.4$ and $r = 1.0$. We show additional results with larger radii in the appendix table 6. We also include FIG-Net [2] which uses face-face contact edges and enforces object rigidity at inference. For comparability, we remove the inference-time hard-coded rigidity for comparison. DPI-Net learns particle dynamics using dynamic graphs and hierarchical message passing with material-specific predictors; we again treat mesh vertices as particles without train-time enforced rigidity for fair comparison. MeshGraphNets [45] learns physical dynamics through message passing on mesh graphs, but does not demonstrate coupled dynamics of multiple freely moving objects. In the appendix section C and table 4, we show results for FIG-Net with enforced rigidity and MeshGraphNets on single object dynamics.

Autoregressive Model Φ_{AR}. We design and implement a transformer autoregressive framework (Φ_{AR}) for next-timestep mesh vertex position prediction. We optimise performance in two ways

Table 2: **Comparison of PHYSIFORMER-L-10k against baselines** on 250 test samples, trained on 10k rigid dataset. MSE measures mean squared vertex-coordinate error against GT trajectories. Rigidity Loss measures implicit rigidity preservation (Equation (2)). Momentum Error measures inference momentum difference from initial system momentum compared against that of GT (Equation (3)). We show results for first 10 predicted frames and full 49-frame trajectory. Note that first 10 predictions are zero-indexed frames 3-12 for FIG-Net, 4-13 for ctx4, and 1-10 for remaining one-frame methods; 49-frame results contain ground truth frames. Best is **bolded**; second best is *italic*. As PHYSIFORMER-L-10k is a generative model, we report a single sample (main row) together with the mean μ and standard deviation σ across 5 generations. PHYSIFORMER-L-10k is best across all metrics for the full horizon among baselines without enforced rigidity.

	MSE ($\times 10^{-3}$)↓		Rigidity Loss ($\times 10^{-4}$)↓		Momentum Error↓	
Method	first 10 pred	49-frame	first 10 pred	49-frame	first 10 pred	49-frame
PHYSIFORMER (Ours)	**0.0953**	**9.55**	**0.0412**	**0.0939**	0.227	**0.274**
5 generations μ	0.0883	9.55	0.0407	0.185	0.231	0.278
5 generations σ	0.00829	0.213	0.000467	0.0991	0.00265	0.00431
$TIE_{r=0.4}$	0.157	17.1	0.328	31.0	0.140	0.470
$TIE_{r=1.0}$	0.173	*14.9*	*0.137*	20.6	*0.113*	0.368
FIG-Net (w/o rigidity)	0.780	20.5	1.20	24.4	0.261	0.422
DPI-Net	*0.132*	16.4	0.327	24.2	**0.0985**	*0.316*
$\Phi_{\text{AR_ctx1}}$	3.14	231	1.56	154	0.889	2.11
$\Phi_{\text{AR_ctx4}}$	1.57	107	0.173	29.9	0.478	1.65
$\Phi_{\text{AR_ctx4_noised}}$	2.03	124	0.150	*20.0*	0.473	1.57
$\Phi_{\text{AR_fullctx}}$	1.47	155	0.418	83.5	0.511	1.74
$\Phi_{\text{AR_fullctx_noised}}$	0.929	191	0.301	40.8	0.356	1.79

inspired by previous works [13, 28]. First, we explore variable context window lengths with fixed lengths 1 and 4, and variable full past context. Longer windows should provide more anchoring information for more stable rollout. Second, to bridge the train-test domain gap between ground-truth-conditioned training and self-conditioned inference, we investigate noise injection in the context window. The design choices and implementation details are in the appendix section A.2.

4.4 Evaluation Metrics

Trajectory MSE Error. Following [49, 47, 54, 33], given the ground-truth and predicted trajectories $X, \hat{X} \in \mathbb{R}^{T\times N\times 3}$, we define the mean-square error (MSE) as $\mathcal{L}_{\text{MSE}}(\hat{X}|X) = \|\hat{X} - X\|_2^2/(TN)$.

Rigidity Preservation. Maintaining object rigidity is a challenge, as seen in severe object deformation across AR baselines in fig. 3. In TIE's original Boxbath dataset [33], rigidity is hard-coded via a single predicted rigid transformation for the rigid object (box), which is used to derive motion for all constituent particles. Thus, we evaluate our model's ability to *implicitly* learn rigidity by averaging the deviation across all objects and frames from a rigid transformation of each object's first-frame position. We use the Kabsch algorithm [26] to compute a best-fit rigid transform with rotation matrix $R \in SO(3)$ and translation vector $\boldsymbol{b} \in \mathbb{R}^3$ and define the rigidity error as:

$$\mathcal{L}_{\text{rigid}}(\hat{X}|X_0, \mathcal{F}) = \frac{1}{3T|\,\text{cc}(\mathcal{F})|}\sum_{t=1}^{T}\sum_{O\in\text{cc}(\mathcal{F})}\frac{1}{|O|}\min_{(R,\boldsymbol{b})\in SE(3)}\|\hat{X}_{t,O} - X_{0,O}R - \boldsymbol{b}\|_F^2, \qquad (2)$$

where $\text{cc}(\mathcal{F})$ denotes the connected components/objects of the mesh, $O \subseteq \{1, \ldots, N\}$ is an object, and $X_{*,O}$ denotes the subset of vertices that belong to the component.

Momentum Prediction. We also test the ability of the model to predict the momentum of the system, a fundamental physical property. We compute the momentum of object O at timestep t by approximating the velocity of the center of mass as the average of the vertex velocities computed using finite difference $P_{t,O}(X) = m(O)\sum_{i\in O}(X_{t,i} - X_{t-1,i})/(|O|\Delta t)$ for $t \geq 1$, with the given V_0 replacing the finite difference at $t = 0$. We then denote the total momentum at time t as $P_t(X) = \sum_{O\in\text{cc}(\mathcal{F})} P_{t,O}(X)$. We measure the per-sample momentum error by comparing the

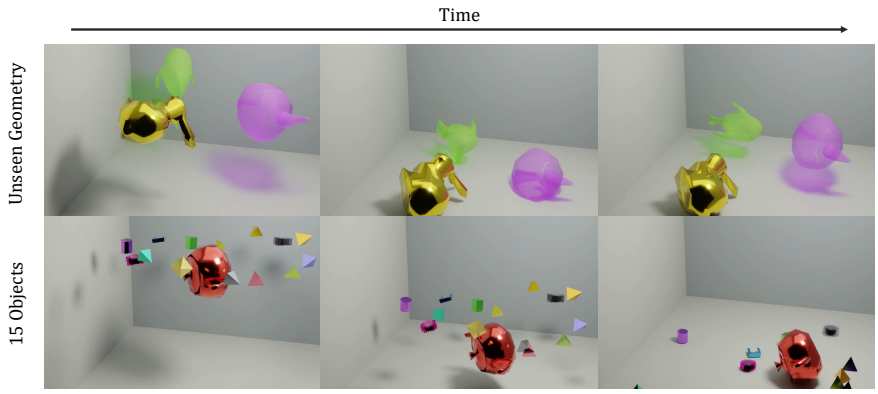


Figure 4: **PHYSIFORMER generalizes to complex real-world object geometries and object counts not seen during training**. Top: Inference on 2 deformable objects (fish and teapot) plus 1 rigid bunny, each with 100 vertices per object. Deformation is most visible for the middle-frame purple teapot. PHYSIFORMER allows mixed-material inference although training only saw uniform material across all objects per scene. Bottom: Inference on 15 rigid objects although training saw at most 10. Extension to more objects is direct owing to object-level attention, which implicitly encodes object identity in the architecture design.

predicted and ground-truth momentum changes from the initial frame:

$$\mathcal{L}_{\text{mom}}(\hat{X}|X, X_0, \mathcal{F}) = \sqrt{\frac{1}{T}\sum_{t=1}^{T}\left\|[P_t(\hat{X}) - P_0(\hat{X})] - [P_t(X) - P_0(X)]\right\|_2^2}. \tag{3}$$

We report the test-set average, normalised by the RMS of the ground-truth initial total momentum across the test set; lower is better, with 0 indicating momentum changes identical to the ground truth. We assume constant mass for computational feasibility; because rigidity is largely not preserved in AR baselines, their momentum error should be interpreted as a rough approximation.

Shape Deformation Recovery We also quantitatively evaluate PHYSIFORMER's ability to model elasticity when trained on such data by measuring shape deformation recovery. We define a new elasticity metric $\mathcal{L}_{\text{elastic}}$ as detailed in the appendix D.1 and observed that, over 250 test set samples, PHYSIFORMER's predicted recovered deformation amount differs from GT by only 1.5% of the initial object size on average. As objects are fairly uniform in size and complexity and exhibit relatively large deformations upon impact, this illustrates PHYSIFORMER's effectiveness at modelling shape recovery upon impact.

4.5 Comparisons Against Baselines

For fairness, all baselines are trained on the 10k rigid-body dataset D_1 using a stratified 9500/250/250 train/validation/test split and are compared with PHYSIFORMER-L-10k trained on the same data. Results are shown in table 2 and fig. 3. All AR approaches suffer from error accumulation, with mesh deformation and trajectory divergence over time. Short-horizon autoregressive rollouts over the first 10 steps are meaningfully better. Across Φ_{AR} variants, a longer context window and noise injection help with stability. For TIE, a larger radius models a broader interaction neighborhood, yielding better results. Importantly, we show in the appendix table 6 that all AR models perform well when conditioned on *ground-truth* data at each timestep, highlighting their ability to learn in-distribution dynamics. Note that PHYSIFORMER-L-10k displays a large standard deviation for MSE across generations (table 2), but MSE alone is not a good indicator of physical plausibility. Slight contact-angle differences yield different reaction forces in collisions, causing trajectory divergence, while generated rigid motion can differ from ground truth yet remain physically plausible.

4.6 Generalisation to Unseen Geometries and Object Numbers

PHYSIFORMER-L-10k generalises well to object geometries and counts not seen in training as shown in the appendix section E. PHYSIFORMER is trained on extended data with more complex motion

and multiple material properties, and displays impressive generalisation to three settings. First, it generalises to complex, real-world meshes, with far more vertices per object, although training used only the simpler mesh primitives depicted in (a) and (b) in fig. 6. Second, it generalises to object counts beyond those seen in training. Third, it generalises to mixed-material scenes although training saw only uniform material per scene. We illustrate these claims in fig. 4 and on the project page.

4.7 PHYSIFORMER vs. Physics Simulators

While physics simulators provide high-fidelity trajectories by numerically integrating physical laws, PHYSIFORMER offers a powerful learned alternative. First, it generates physically plausible motion from only initial positions and velocities, without specifying a full physical state such as density or friction as input. Second, once trained, it has a capped inference cost, enabling efficient rollout even for deformable and contact-rich scenes that are costly to simulate. On an 80-thread Intel Xeon Gold 6338 CPU node, Genesis rigid-body simulation averaged 1–6.5s per sample for 1–10 objects. However, elastic-body simulation averaged 20–36s per sample for 1–5 objects excluding rendering, about 8.5–15$\times$ the 2.4s PHYSIFORMER inference with 25 steps on one H100 GPU (table 3), albeit on different hardware. Though physics simulators can be sped up, their material-dependent timing gap highlights a key advantage of learned rollout: PHYSIFORMER uses a fixed number of network evaluations rather than costly per-scene simulation. Third, it generalises to complex, real-world mesh geometries that can be difficult for simulators to handle robustly. In challenging scenes, where simulators may fail due to contact-resolution artifacts or objects leaving the simulation bounding box as shown in fig. 7 in the appendix, PHYSIFORMER can still produce plausible samples.

4.8 Ablations

Noise Scale in Diffusion. Standard diffusion models typically assume normalized inputs, such as VAE latents, and sample noise from a standard Gaussian. For raw coordinate signals, however, noise scale is important for generation quality [32, 23]. As shown in table 7 in the appendix trained on smaller DiT-B based models with 1k data, which we call PHYSIFORMER-B-1k, we sample $\epsilon \sim \mathcal{N}(0, I) \times$ `noise_scale` and find `noise_scale` $= 0.1$ to work best: smaller values hinder generalisation, while larger values make denoising harder and often introduce jitter. We hypothesize that this narrower noise distribution is effective because trajectories are strongly conditioned on the first-frame state. It also stabilizes our coordinate-derived spatial RoPE, whose inputs are noised trajectories during training and early sampling.

Object-Level Attention vs Object-ID Embedding We compare PHYSIFORMER-L-10k's factorised per-object attention with a baseline that uses alternating full spatial and temporal attention plus learnable per-vertex object-ID embeddings in $\mathbb{R}^{(N_{\text{obj}}+1)\times D}$, including a padding ID. On 10k rigid trajectories over 49 frames, the two perform similarly: PHYSIFORMER (underlined) has slightly worse MSE (<u>9.6e-3</u> vs. 9.1e-3), comparable momentum error (<u>0.274</u> vs 0.277), and better rigidity loss (<u>1.9e-5</u> vs. 4.8e-5), with comparable qualitative results. Object-ID embeddings do not extrapolate to more objects, since unseen object indices have no learned embeddings.

5 Conclusion

We presented PHYSIFORMER, a unified diffusion transformer that learns multi-material, multi-object mechanics as full-trajectory coordinate diffusion in world space. By generating future mesh motion jointly rather than autoregressively, PHYSIFORMER avoids error accumulation while preserving object coherence through factorised spatial, temporal, and object-level attention. It improves trajectory accuracy, rigidity preservation, and momentum-based physical consistency over autoregressive baselines, and generalises to unseen geometries, larger object counts, and mixed-material settings. Its current limits are fixed trajectory length and mesh resolution, motivating longer-horizon generation, spatial compression, and physics-aware objectives for contact. PHYSIFORMER uses a general diffusion objective without explicit collision or object-consistency constraints, leading to occasional spurious contacts, interpenetration, and rare orientation discontinuities. These artifacts may be reduced through contact-focused training and tailored physical-consistency losses. Overall, PHYSIFORMER points toward geometry-level world models for physically plausible 3D dynamics.

Acknowledgments and Disclosure of Funding

Yiming Chen is supported by the Rhodes Scholarship. Yushi Lan and Andrea Vedaldi are supported by the European Research Council (ERC) grant CoG 101001212-UNION. We thank Isambard-AI and Dawn AIRR supercomputers (project code: 0261–5548–9011–1) for supporting this work. The authors declare no competing interests.

References

[1] Michael S Albergo and Eric Vanden-Eijnden. Building normalizing flows with stochastic interpolants. In *Proc. ICLR*, 2023.

[2] Kelsey R Allen, Yulia Rubanova, Tatiana Lopez-Guevara, William Whitney, Alvaro Sanchez-Gonzalez, Peter Battaglia, and Tobias Pfaff. Learning rigid dynamics with face interaction graph networks. *arXiv preprint arXiv:2212.03574*, 2022.

[3] Mido Assran, Adrien Bardes, David Fan, Quentin Garrido, Russell Howes, Mojtaba Komeili, Matthew Muckley, Ammar Rizvi, Claire Roberts, Koustuv Sinha, Artem Zholus, Sergio Arnaud, Abha Gejji, Ada Martin, Francois Robert Hogan, Daniel Dugas, Piotr Bojanowski, Vasil Khalidov, Patrick Labatut, Francisco Massa, Marc Szafraniec, Kapil Krishnakumar, Yong Li, Xiaodong Ma, Sarath Chandar, Franziska Meier, Yann LeCun, Michael Rabbat, and Nicolas Ballas. V-JEPA 2: Self-supervised video models enable understanding, prediction and planning. *arXiv*, 2506.09985, 2025.

[4] Genesis Authors. Genesis: A generative and universal physics engine for robotics and beyond, 2024. URL `https://github.com/Genesis-Embodied-AI/Genesis`.

[5] David Baraff and Andrew Witkin. Large steps in cloth simulation. In *Proc. SIGGRAPH*, 1998.

[6] Peter W. Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, and Koray Kavukcuoglu. Interaction networks for learning about objects, relations and physics. In *Proc. NeurIPS*, 2016.

[7] Saakaar Bhatnagar, Yaser Afshar, Shaowu Pan, Karthik Duraisamy, and Shailendra Kaushik. Prediction of aerodynamic flow fields using convolutional neural networks. *Computational Mechanics*, 64(2), 2019.

[8] Andreas Blattmann, Tim Dockhorn, Sumith Kulal, Daniel Mendelevitch, Maciej Kilian, Dominik Lorenz, Yam Levi, Zion English, Vikram Voleti, Adam Letts, Varun Jampani, and Robin Rombach. Stable video diffusion: Scaling latent video diffusion models to large datasets. *arXiv.cs*, abs/2311.15127, 2023.

[9] Andreas Blattmann, Robin Rombach, Huan Ling, Tim Dockhorn, Seung Wook Kim, Sanja Fidler, and Karsten Kreis. Align your latents: High-resolution video synthesis with latent diffusion models. In *Proc. CVPR*, 2023.

[10] Sofien Bouaziz, Sebastian Martin, Tiantian Liu, Ladislav Kavan, and Mark Pauly. Projective dynamics: Fusing constraint projections for fast simulation. In *Proc. SIGGRAPH*, 2014.

[11] Tim Brooks, Bill Peebles, Connor Holmes, Will DePue, Yufei Guo, Li Jing, David Schnurr, Joe Taylor, Troy Luhman, Eric Luhman, Clarence Ng, Ricky Wang, and Aditya Ramesh. Video generation models as world simulators. Technical report, OpenAI, 2024.

[12] Olivier Chapelle, Bernhard Schölkopf, and Alexander Zien. A discussion of semi-supervised learning and transduction. In *Semi-Supervised Learning*. The MIT Press, 2006.

[13] Boyuan Chen, Diego Marti Monso, Yilun Du, Max Simchowitz, Russ Tedrake, and Vincent Sitzmann. Diffusion forcing: Next-token prediction meets full-sequence diffusion. *arXiv*, 2407.01392, 2024.

[14] Boyuan Chen, Hanxiao Jiang, Shaowei Liu, Saurabh Gupta, Yunzhu Li, Hao Zhao, and Shenlong Wang. PhysGen3D: Crafting a miniature interactive world from a single image. *CVPR*, 2025.

[15] Hongyuan Chen, Xingyu Chen, Youjia Zhang, Zexiang Xu, and Anpei Chen. Motion 3-to-4: 3D motion reconstruction for 4D synthesis. *arXiv*, 2601.14253, 2026.

[16] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning, 2016. URL `http://pybullet.org`.

[17] Timothée Darcet, Maxime Oquab, Julien Mairal, and Piotr Bojanowski. Vision transformers need registers. *Proc. ICLR*, 2024.

[18] Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *Proc. AAAI Workshop*, 2021.

[19] Nate Gillman, Charles Herrmann, Michael Freeman, Daksh Aggarwal, Evan Luo, Deqing Sun, and Chen Sun. Force prompting: Video generation models can learn and generalize physics-based control signals. In *Proc. NeurIPS*, volume 38, 2025.

[20] Artur Grigorev, Michael J Black, and Otmar Hilliges. Hood: Hierarchical graphs for generalized modelling of clothing dynamics. In *Proc. CVPR*, 2023.

[21] Xiaoxiao Guo, Wei Li, and Francesco Iorio. Convolutional neural networks for steady flow approximation. In *Proc. SIGKDD*, 2016.

[22] Alex Henry, Prudhvi Raj Dachapally, Shubham Pawar, and Yuxuan Chen. Query-key normalization for transformers. *arXiv*, 2020.

[23] Emiel Hoogeboom, Jonathan Heek, and Tim Salimans. simple diffusion: End-to-end diffusion for high resolution images. In *Proc. ICML*, 2023.

[24] Xun Huang, Zhengqi Li, Guande He, Mingyuan Zhou, and Eli Shechtman. Self forcing: Bridging the train-test gap in autoregressive video diffusion. *arXiv*, 2506.08009, 2025.

[25] Hanxiao Jiang, Hao-Yu Hsu, Kaifeng Zhang, Hsin-Ni Yu, Shenlong Wang, and Yunzhu Li. PhysTwin: Physics-informed reconstruction and simulation of deformable objects from videos. In *Proc. ICCV*, 2025.

[26] Wolfgang Kabsch. A solution for the best rotation to relate two sets of vectors. *Foundations of Crystallography*, 32(5), 1976.

[27] Bingyi Kang, Yang Yue, Rui Lu, Zhijie Lin, Yang Zhao, Kaixin Wang, Gao Huang, and Jiashi Feng. How far is video generation from world model: A physical law perspective. In *Proc. ICML*, 2025.

[28] Efstathios Karypidis, Ioannis Kakogeorgiou, Spyros Gidaris, and Nikos Komodakis. DINO-foresight: Looking into the future with DINO. In *Proc. NeurIPS*, 2025.

[29] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3D Gaussian Splatting for real-time radiance field rendering. *Proc. SIGGRAPH*, 42(4), 2023.

[30] Minh-Quan Le, Yuanzhi Zhu, Vicky Kalogeiton, and Dimitris Samaras. What about gravity in video generation? post-training Newton's laws with verifiable rewards. *arXiv*, 2512.00425, 2025.

[31] Minchen Li, Danny M Kaufman, and Chenfanfu Jiang. Codimensional incremental potential contact. In *Proc. SIGGRAPH*, 2021.

[32] Tianhong Li and Kaiming He. Back to basics: Let denoising generative models denoise. In *CVPR*, 2025.

[33] Yunzhu Li, Jiajun Wu, Russ Tedrake, Joshua B Tenenbaum, and Antonio Torralba. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. In *Proc. ICLR*, 2019.

[34] Xingyu Lin, Yufei Wang, Zixuan Huang, and David Held. Learning visible connectivity dynamics for cloth smoothing. In *Proc. CoRL*, 2021.

[35] Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv.cs*, abs/2210.02747, 2022.

[36] Shaowei Liu, Zhongzheng Ren, Saurabh Gupta, and Shenlong Wang. PhysGen: Rigid-body physics-grounded image-to-video generation. In *Proc. ECCV*, 2024.

[37] Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. In *Proc. ICLR*, 2023.

[38] Zhen Liu, Yao Feng, Michael J. Black, Derek Nowrouzezahrai, Liam Paull, and Weiyang Liu. Meshdiffusion: Score-based generative 3d mesh modeling. In *ICLR*, 2023.

[39] Eder Miguel, Derek Bradley, Bernhard Thomaszewski, Bernd Bickel, Wojciech Matusik, Miguel A. Otaduy, and Steve Marschner. Data-driven estimation of cloth simulation models. In *Proc. Eurographics*, 2012.

[40] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing scenes as neural radiance fields for view synthesis. In *Proc. ECCV*, 2020.

[41] Damian Mrowca, Chengxu Zhuang, Elias Wang, Nick Haber, Fei-Fei Li, Josh Tenenbaum, and Daniel L K Yamins. Flexible neural representation for physics prediction. In *Proc. NeurIPS*, 2018.

[42] Matthias Müller, David Charypar, and Markus Gross. Particle-based fluid simulation for interactive applications. In *Proc. Eurographics*, 2003.

[43] Jack Parker-Holder and Shlomi Fruchter. Genie 3: A new frontier for world models, 2025. URL `https://deepmind.google/blog/genie-3-a-new-frontier-for-world-models/`.

[44] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proc. ICCV*, 2023.

[45] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W. Battaglia. Learning mesh-based simulation with graph networks. In *Proc. ICLR*, 2021.

[46] Yulia Rubanova, Tatiana Lopez-Guevara, Kelsey R. Allen, William F. Whitney, Kimberly Stachenfeld, and Tobias Pfaff. Learning rigid-body simulators over implicit shapes for large-scale scenes and vision. In *Proc. NeurIPS*, volume 37, 2024.

[47] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter W. Battaglia. Learning to simulate complex physics with graph networks. In *Proc. ICML*, 2020.

[48] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Trans. on Neural Networks*, 2009.

[49] Yidi Shao, Chen Change Loy, and Bo Dai. Transformer with implicit edges for particle-based physics simulation. In *Proc. ECCV*, 2022.

[50] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. In *Proc. NAACL*, 2018.

[51] Noam M. Shazeer. GLU variants improve transformer. In *arXiv*, 2020.

[52] Jianlin Su, Murtadha H. M. Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. RoFormer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 2024.

[53] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *Proc.IROS*, 2012.

[54] Benjamin Ummenhofer, Lukas Prantl, Nils Thuerey, and Vladlen Koltun. Lagrangian fluid simulation with continuous convolutions. In *Proc. ICLR*, 2020.

[55] Pascal Volino, Nadia Magnenat-Thalmann, and Francois Faure. A simple approach to nonlinear tensile stiffness for accurate cloth simulation. In *Proc. SIGGRAPH*, 2009.

[56] Chen Wang, Chuhao Chen, Yiming Huang, Zhiyang Dou, Yuan Liu, Jiatao Gu, and Lingjie Liu. Physctrl: Generative physics for controllable and physics-grounded video generation. In *NeurIPS*, 2025.

[57] Amaury Wei and Olga Fink. Integrating physics and topology in neural networks for learning rigid body dynamics. *Nature Communications*, 16(1), 2025.

[58] Tianyi Xie, Zeshun Zong, Yuxing Qiu, Xuan Li, Yutao Feng, Yin Yang, and Chenfanfu Jiang. PhysGaussian: Physics-integrated 3D Gaussians for generative dynamics. In *Proc. CVPR*, 2024.

[59] Youn-Yeol Yu, Jeongwhan Choi, Woojin Cho, Kookjin Lee, Nayong Kim, Kiseok Chang, ChangSeung Woo, Ilho Kim, SeokWoo Lee, Joon Young Yang, et al. Learning flexible body collision dynamics with hierarchical contact mesh transformer. In *Proc. ICLR*, 2024.

[60] Chong Zeng, Yue Dong, Pieter Peers, Hongzhi Wu, and Xin Tong. RenderFormer: transformer-based neural rendering of triangle meshes with global illumination. In *Proc. SIGGRAPH*, 2025.

[61] Biao Zhang and Rico Sennrich. Root mean square layer normalization. In *Proc. NeurIPS*, 2019.

[62] Biao Zhang, Jiapeng Tang, Matthias Niessner, and Peter Wonka. 3DShape2VecSet: A 3D shape representation for neural fields and generative diffusion models. In *ACM Transactions on Graphics*, 2023.

[63] Bowen Zhang, Sicheng Xu, Chuxin Wang, Jiaolong Yang, Feng Zhao, Dong Chen, and Baining Guo. Gaussian variation field diffusion for high-fidelity video-to-4D synthesis. In *Proc. ICCV*, 2025.

[64] Mingtong Zhang, Kaifeng Zhang, and Yunzhu Li. Dynamic 3D Gaussian tracking for graph-based neural dynamics modeling. In *Proc. CoRL*, 2024.

[65] Licheng Zhong, Hong-Xing Yu, Jiajun Wu, and Yunzhu Li. Reconstruction and simulation of elastic objects with spring-mass 3D Gaussians. In *Proc. ECCV*, 2024.

PHYSIFORMER: Learning to Simulate Mechanics in World Space

Supplementary Material

A Method Continued

A.1 PHYSIFORMER

Register Tokens. In input data tokenization, we further prepend $N_{\text{reg}} = 16$ shared, learnable register tokens, yielding per-sample token embeddings in $\mathbb{R}^{(N_{\text{reg}}+T\cdot N)\times D}$. The register tokens capture global context and can potentially remove high-frequency noise in embeddings [17]. Across the temporal and spatial attention blocks, register tokens are replicated across the corresponding dimensions (time for full spatial blocks, objects and time for object-level spatial blocks, vertices for temporal blocks) before attention. They are consolidated back to a single set of tokens via averaging after attention, allowing global information to be shared consistently across factorised operations. Within temporal blocks, temporal RoPE assigns register tokens a fixed time index of 0. For spatial blocks, each register token uses the mean position of the vertices selected by its context mask for spatial RoPE.

DiT Details. Up to the modifications described above, the DiT blocks are standard. Each block uses RMSNorm [61] and AdaLN [44] conditioning from the diffusion timestep embedding to produce shift/scale modulation and gated residuals, followed by non-causal multi-head self-attention with QK-normalization [22] and a feed-forward network with SwiGLU activations [51]. By default, we train PHYSIFORMER, which mirrors DiT-L in size, instantiating 24 blocks in total. Each block uses 16 attention heads and hidden size $D = 1024$.

A.2 Baseline Φ_{AR}

Autoregressive Dynamics Modeling. The autoregressive predictor Φ_{AR} outputs next-timestep velocities for N vertices given context length L. Starting with $L = 1$, given inputs $X_t \in \mathbb{R}^{1\times N\times 3}$ and $V_t \in \mathbb{R}^{1\times N\times 3}$, the model predicts next-step velocities $V'_{t+1} \in \mathbb{R}^{1\times N\times 3}$ from the final context-frame velocity token. As prior work showed modeling relative offsets is more natural for attention than absolute values [50], we predict velocity and recover positions by integration, $X'_{t+1} = X_t + \Delta t\, V'_{t+1}$, where Δt is defined during data generation. During training, we normalize positions and velocities using the dataset-wide global mean and standard deviation computed separately for each quantity. We supervise on velocity loss $\mathcal{L} = \texttt{SmoothL1}(V'_{t+1}, V_{t+1})$, with $\beta = 1.0$. At inference, we start from the ground-truth state at $t = 0$, predict V'_{t+1}, integrate to obtain X'_{t+1}, and iteratively roll out by feeding (X'_{t+1}, V'_{t+1}) back into Φ_{AR}.

Context Window. Intuitively, a model Φ_{AR} that can see more of its past outputs should be able to predict the future with higher consistency. The context window determines the dependency horizon of the autoregressive framework. In the spirit of DINO-Foresight [28], which faithfully predicts next timestep latent DINO features autoregressively with context window 4, we implement Φ_{AR_ctx4} with $L = 4$ where inputs are $X_{t-4:t} \in \mathbb{R}^{4\times N\times 3}$ and $V_{t-4:t} \in \mathbb{R}^{4\times N\times 3}$ and output is V'_t. At inference, we roll out predictions in a sliding-window fashion, starting with the first 4 ground-truth timesteps.

Train-time Noise Injection. For our autoregressive framework on mesh vertex prediction (Φ_{AR}), we employ noise injection during training to bridge the train-test domain gap. In theory, by injecting controlled noise into the context window during training, the model learns to self-correct prediction errors accumulated during autoregressive inference. We implement $\Phi_{\text{AR_ctx4_noised}}$ and carefully tune the noise level to $\epsilon_{pos} \sim \mathcal{N}(0, 0.008^2)$ and $\epsilon_{vel} \sim \mathcal{N}(0, 0.08^2)$ according to the level of error accumulation observed during autoregressive rollout. We also implement $\Phi_{\text{AR_full_ctx}}$ where at training time, prediction for timestep t is conditioned on the full past history $0, \ldots, t-1$. At inference time, the model takes in ground truth frame 0 and grows context window as predictions roll out.

Φ_{AR} Architecture and Training Details. Φ_{AR} is a pre-norm Transformer encoder that predicts next-timestep vertex velocities with maximum N vertices, given context length L. We tokenize vertex positions and velocities with shape $L \times N \times 3$, masking samples with fewer vertices. Each 3D coordinate is encoded with 8 Fourier features and projected to a $C = 384$-dimensional token

Table 3: **Physics Simulator Stepping Time vs PHYSIFORMER Inference Time.** We analyse the effects of denoising step numbers on evaluation metric performance and inference time on a single NVIDIA H100 GPU; momentum error is reported over the full 49 frames. Rigid objects (Left): over 350 test samples with 61 vertices on average chosen using randomised stratified selection to reflect data distribution, we observe a linear upward trend for per-sample inference time. While MSE is best at smaller denoising steps, rigidity is best preserved at larger denoising step numbers. Accounting for the extent to which MSE reflects physical plausibility and taken together with qualitative renderings, 25 denoising steps performs well and is comparable to 50 denoising steps. Elastic objects (Right): over 40 test samples with 60 vertices on average chosen using randomised stratified selection to reflect data distribution, per-sample inference time is similar to that of rigid object scenes and exhibits linear increase. MSE is lowest with a single denoising step, while momentum error is best at 5–10 denoising steps. Qualitative results on the project page show that 5 denoising steps and above appears physically plausible.

Denoising Steps	Inference Time	MSE	Rigidity Loss	Momentum Error
1	0.073s	**1.390e-2**	9.949e-4	0.5280
5	0.432s	1.582e-2	4.889e-5	**0.2483**
10	0.871s	1.667e-2	2.540e-5	0.2575
25	2.187s	1.832e-2	2.323e-5	0.3858
50	4.378s	1.944e-2	**2.187e-5**	0.6368

Denoising Steps	Inference Time	MSE	Momentum Error
1	0.081s	**1.245e-2**	1.0448
5	0.465s	1.403e-2	**0.7267**
10	0.939s	1.461e-2	0.7443
25	2.365s	1.544e-2	0.7598
50	4.745s	1.574e-2	0.9969

embedding. Tokens are augmented with a learnable shared base vertex token, temporal embeddings for relative ordering within context window, per-vertex object-ID embeddings, and a type embedding that distinguishes position from velocity tokens. We prepend $R = 4$ learnable register tokens and concatenate all position and velocity tokens. Φ_{AR} has four layers, eight attention heads, a feed-forward dimension of 1024, and dropout 0.1. Φ_{AR} is trained by enumerating all context window-prediction pairs for all samples (e.g. 49–4 = 45 pairs for $L = 4$). We use AdamW optimizer with a constant $lr = 1\mathrm{e}{-3}$ schedule and effective batch size 96. We train for 300k steps on one NVIDIA H100 GPU with 90GB memory.

B Inference Time and Denoising Step Numbers

For reported statistics and visualisations, we apply a 50-step Heun sampler during inference by default. PHYSIFORMER uses a DiT-L backbone, and we report number of denoising steps against inference time on a single NVIDIA H100 GPU and evaluation metric performance in table 3. Notably, we observe that fewer denoising steps yields outputs of comparable quality. On 250 test samples from the PHYSIFORMER-L-10k training dataset, Φ_{AR} variants are 4-layer Transformers with width 384 (section A.2) and require 0.2s on a single H100 machine; and TIE variants have around 770k parameters and require 0.47s on a single Quadro RTX 6000 machine (chosen for compatibility with legacy Python and CUDA dependencies in the original codebase). Although PHYSIFORMER is a larger model, we show that current AR baseline's smaller model can learn in-distribution dynamics section G.1 with GT-conditioned rollout and attribute failure scenarios in long-horizon autoregressive rollout. Additionally, we show with PHYSIFORMER that the larger model can capture significantly more nuanced dynamics on more complex meshes with increased generalisation capabilities.

C Baselines with Enforced Priors or Simplified Settings

FIG-Net [2] uses a graph neural network with mesh face-face contact edges and makes the assumption that objects are rigid to improve predictions at inference time. Its performance is nearly identical to ours (and slightly better) under this assumption, but the model does not generalise to the non-rigid case, whereas PHYSIFORMER is agnostic to such assumptions. As shown in table 4, with the rigid-transform prior active FIG-Net reaches a 49-frame MSE of 9.00×10^{-3} against 9.55×10^{-3} for PHYSIFORMER-L-10k, and a full-horizon momentum error of 0.276 against our 0.274; the gap is larger only over the first 10 predicted frames (0.0747 versus 0.227), where the hard-coded rigid

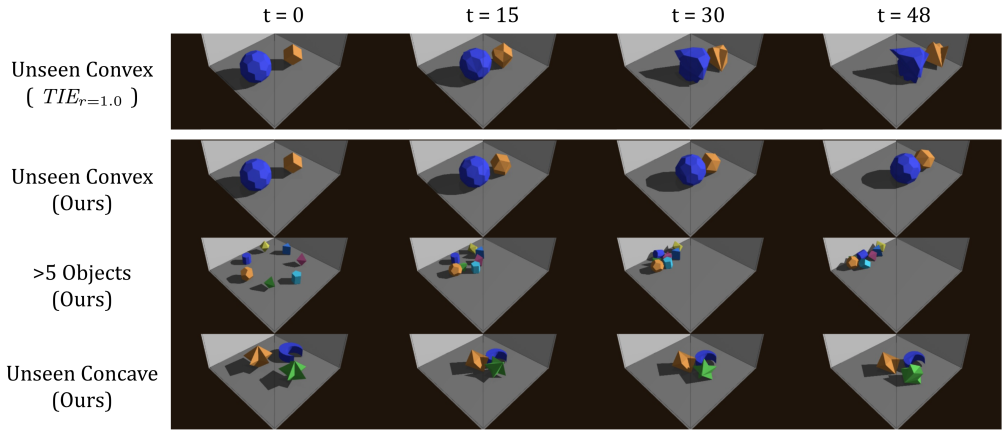


Figure 5: **PHYSIFORMER-L-10k generalizes to object geometries and counts not seen during training**, shown at $t = 0, 15, 30, 48$. The first row shows the best AR model ($TIE_{r=1.0}$) on two unseen convex objects. For the following rows, we have *top:* two unseen convex objects, *middle:* seven objects from seen convex templates, exceeding the training maximum of five, *bottom:* three objects with unseen concave geometry. PHYSIFORMER-L-10k produces physically plausible rigid-body dynamics across all settings, demonstrating robustness to topology, vertex count, and scene complexity beyond the training distribution.

Table 4: **Baselines with enforced priors or simplified settings.** With rigidity enforced at inference, FIG-Net performs comparably to PHYSIFORMER-L-10k on the rigid dataset alone, but the prior does not extend to other material properties. MeshGraphNets is evaluated on single-object dynamics, a substantially easier setting in which no object–object collisions must be modelled.

	MSE ($\times 10^{-3}$)↓	Rigidity Loss ($\times 10^{-4}$)↓	$\mathcal{L}_{\text{mom}}$ ↓	
Method	49-frame	49-frame	first 10 pred	49-frame
PHYSIFORMER-L-10k (Ours)	9.55	0.0939	0.227	0.274
FIG-Net (enforced rigidity)	9.00	–	0.0747	0.276
MeshGraphNets (1 object, 2k)	1.91	5.8	0.0297	0.0886

transform removes early per-vertex noise outright. Without the enforced rigidity, objects quickly deform and penetrate each other. Moreover, barring the enforced rigid transform, it is non-trivial to extend FIG-Net to model deformable objects and it is unclear whether a model designed for rigid collision dynamics can be adapted. For example, the global object node is useful for rigid objects but may be less effective at summarising local object deformations and the contact edges require recomputation based on deformations. We train on 10k rigid samples, the same as PHYSIFORMER.

MeshGraphNets [45] represents the simulation mesh as a graph and predicts how each mesh node changes at the next timestep through message passing. It requires significant modifications to handle more than one dynamic object as there are no edges designed to capture interactions between moving objects. Hence, we trained it in the relatively simpler case of a single rigid object simulation on a dataset of 2k samples. In this case, as shown in table 4, object rigidity cannot be maintained: the Rigidity Loss of 5.8×10^{-4} is roughly $60\times$ our 0.0939×10^{-4}, despite the far simpler setting. The better performance on MSE (1.91×10^{-3} versus 9.55×10^{-3}) and $\mathcal{L}_{\text{mom}}$ (0.0886 versus 0.274 over the full horizon) is due to simpler object motions and collision patterns with fewer chances for trajectory deviation and momentum drift caused by object contact, and these numbers are therefore not comparable to the multi-object results in table 2.

D Evaluation Metrics Continued

D.1 Elasticity Loss

To evaluate, we define a new elasticity loss $\mathcal{L}_{\text{elastic}}$ to measure PHYSIFORMER's ability to capture shape deformation recovery after seeing elastic object training data.

For each object o, we measure the normalised shape deformation at frame t using the Kabsch-aligned vertex RMSE

$$S_o(t) = \frac{1}{L_o} \min_{R \in SO(3),\, c} \sqrt{\frac{1}{V_o} \sum_{i \in o} \|X_{t,i} - (RX_{0,i} + c)\|_2^2}, \tag{4}$$

where L_o is the initial bounding-box diagonal of the object and V_o its vertex count. The Kabsch alignment removes global translation and rotation, so $S_o(t)$ isolates shape change from rigid-body motion. We then define the recovered deformation amount as

$$A_o = \max_{t>0} S_o(t) - \min_{t > t_o^{\text{peak}}} \text{Avg}_3 \left[S_o(t)\right], \tag{5}$$

where Avg_3 denotes a three-frame moving average taken after the deformation peak t_o^{peak}, which avoids spuriously small values of $S_o(t)$ caused by discontinuities. The object-level error compares the recovered deformation of the prediction against that of the ground truth,

$$E_o = \left| A_o(\hat{X}) - A_o(X) \right|, \tag{6}$$

and we obtain $\mathcal{L}_{\text{elastic}}$ by averaging E_o over the objects within each sample and then equally across test samples, where lower is better. The metric is normalised by object size (captured with L_o) and measures recovery magnitude rather than deformation timing or bounce count.

D.2 Discussion

We follow prior work and use MSE on the ground-truth trajectory for evaluation [49, 47, 54, 33]. However, as is apparent from our video visualizations, deviation from the ground truth does not necessarily imply physical implausibility. In our setting, rigidity loss (when conditioned on rigid materials), momentum error, elasticity loss and qualitatively visualization assessment paint a more complete picture. To assess a model's ability to learn physics more fully, improved and more general evaluation metrics are needed in the field.

E PHYSIFORMER-L-10k Generalization

PHYSIFORMER-L-10k is trained solely on convex object geometries and 1–5 objects per scene.

As shown in fig. 5 and table 5, PHYSIFORMER-L-10k generalizes well to object geometries and counts not seen in training. As PHYSIFORMER-L-10k is a generative framework, we additionally include a best-of-5 generation statistic per sample. For unseen convex shapes, although objects in training have at most 20 vertices, PHYSIFORMER-L-10k is able to generate physically plausible motion for objects with up to 86 vertices, showing our framework's capability to generalize to unseen topology. For object numbers exceeding 5 (maximum seen in training), we see that PHYSIFORMER-L-10k still models interactions between all objects faithfully.

F PHYSIFORMER Training and Inference Meshes

(a) in fig. 6 shows mesh templates used to train PHYSIFORMER-L-10k and PHYSIFORMER-B-1k; (a) and (b) shows mesh templates used to train PHYSIFORMER. (c) shows select unseen meshes with significant increase in complexity that still exhibit good results in PHYSIFORMER inference.

G Additional Quantitative Analysis

G.1 AR Models

Both Φ_{AR} variants and TIE follow an autoregressive inference framework. In table 6, we compare self-conditioned autoregressive inference (the true inference setting) with ground truth-conditioned

Table 5: **PHYSIFORMER-L-10k generalization performance** across in-distribution and out-of-distribution settings on size 10k dataset. *Training test set*: held-out sequences from the training distribution (1–5 objects, 15 seen convex templates). *Unseen convex*: 100 trajectories with 1–5 objects from 10 novel convex meshes (6–86 vertices each). *6–10 obj. seen convex*: 100 trajectories with object counts exceeding the training maximum of 5, using seen convex templates. *Unseen concave*: 100 trajectories with 1–5 objects from 10 novel concave meshes (12–88 vertices each). Metrics follow table 2.

	MSE↓		Rigidity Loss↓	
Setting	1 sample	Best-of-5	1 sample	Best-of-5
Training test set	9.1e-3	**5.5e-3**	**4.8e-5**	**5.5e-6**
Unseen convex	8.6e-3	6.3e-3	1.3e-4	5.7e-5
6–10 obj. seen convex	1.1e-2	8.6e-3	6.4e-5	3.7e-5
Unseen concave	**7.3e-3**	6.1e-3	3.3e-4	3.1e-4

Table 6: **Comparison between self-conditioned and ground truth-conditioned inference for AR models.** We average evaluation metrics across 250 test samples from our 10k-trajectory dataset. For each metric, the first columns show self-conditioned rollout, which follows the true inference-time constraints of autoregressive models, where previous model outputs are used as input for future timestep prediction. The second columns show ground truth-conditioned rollout, in which the next-timestep prediction uses ground-truth context as input, as during training. The low error in the ground truth-conditioned setting indicates that the model generalizes well to in-distribution test data and suggests that error accumulation is the primary cause of long-range instability. The last two rows additionally report results for TIE with a larger implicit edge interaction radius, illustrating model performance when accounting for more granular interactions between individual particles.

	MSE↓		Rigidity Loss↓	
Method	Self Cond.	GT Cond.	Self Cond	GT Cond.
$\Phi_{\text{AR_ctx1}}$	0.22	1.9e-5	1.4e-2	9.8e-7
$\Phi_{\text{AR_ctx4}}$	0.10	1.2e-5	2.8e-3	6.3e-7
$\Phi_{\text{AR_ctx4_noised}}$	0.12	1.2e-5	1.9e-3	5.8e-7
$TIE_{r=0.4}$	0.017	3.4e-6	3.1e-3	3.3e-7
$TIE_{r=1.0}$	**0.015**	2.9e-6	2.8e-3	2.5e-7
$TIE_{r=2.0}$	0.017	**2.8e-6**	**2.0e-3**	**2.0e-7**
$TIE_{r=3.5}$	0.016	2.9e-6	**2.0e-3**	2.1e-7

inference (the training setting). We observe that the latter yields smaller losses and errors, as the model can learn from in-distribution data. The inference-time failure modes of AR models are therefore attributable to compounding errors during rollout.

Furthermore, for TIE, we explored increasing the interaction radius to 2.0 and 3.5. In a $2 \times 2 \times 2$ box, a radius of 2.0 models interactions among most particles, while a radius of 3.5 effectively connects every particle to every other particle via implicit edges. We omit these radii in the main paper because they offer only a slight advantage on certain metrics, and we aimed to match the radius setting in TIE, which connects particles only within a small neighborhood calibrated to the particles' range of motion. Results are shown in table 6.

G.2 PHYSIFORMER Chunked Long-Horizon Inference

Current PHYSIFORMER inference output length is determined by the training data length of 49 frames. Since inference only requires initial position and velocity, we explore PHYSIFORMER's potential for chunked long-horizon rollout. For each continuation chunk, the final generated frame of the previous chunk is used as the next initial position. The next initial velocity is estimated by finite differencing the final two generated frames. Full-horizon rigidity error increases with rollout length, rising from 6.99e-05 at 1x/49 frames to 6.87e-04 at 4x/193 frames, which yields visible deformations. This

Table 7: **Ablation study on diffusion noise scale and tokenization strategy.** All variants use PHYSIFORMER-B-1k (12 total DiT blocks, $D = 768$) trained on a 1k-sample subset, suffixes omitted in table for brevity. Subscript ns denotes the noise scale used to sample $\epsilon \sim \mathcal{N}(0, I) \times ns$. PHYSIFORMER$_{\text{tri},ns=0.1}$ replaces per-vertex tokens with per-triangle tokens (flattened 9D coordinates), analogous to RenderFormer [60].

	MSE ↓		Rigidity Loss ↓	
Method	1 sample	Best-of-5	1 sample	Best-of-5
PHYSIFORMER$_{ns=0.05}$	0.0073	0.0058	4.5e-4	**1.8e-4**
PHYSIFORMER$_{ns=0.1}$	**0.0066**	**0.0056**	**1.9e-4**	1.9e-4
PHYSIFORMER$_{ns=0.25}$	0.0072	0.0069	3.2e-4	2.7e-4
PHYSIFORMER$_{ns=0.5}$	0.0072	0.0070	2.8e-4	2.8e-4
PHYSIFORMER$_{\text{tri},ns=0.1}$	0.015	0.014	1.5e-3	1.4e-3

observation motivates further exploration, perhaps via more noise-robust initial-state conditioning in training with noise injection. Visualizations are available on the project page.

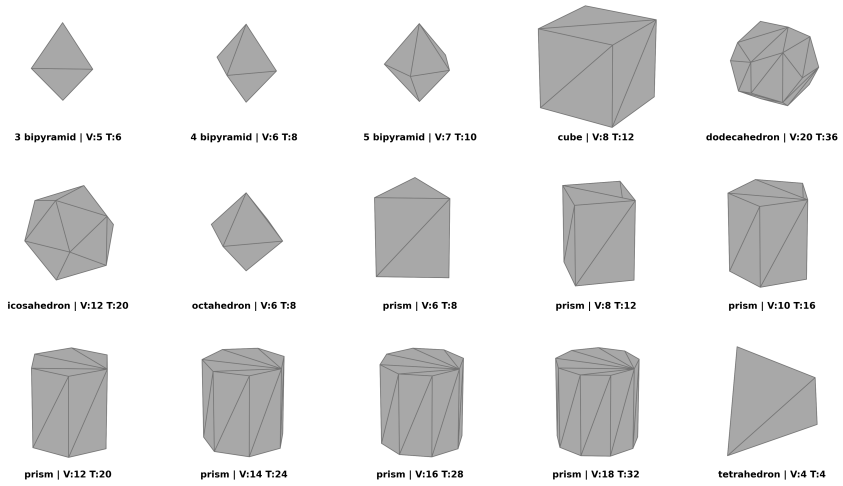
H Ablations Continued

Noise Scale in Diffusion. As explained in section 4.8, the noise scale applied in diffusion training is significant for performance. Quantitative evaluation results for varying noise scale are shown in table 7.

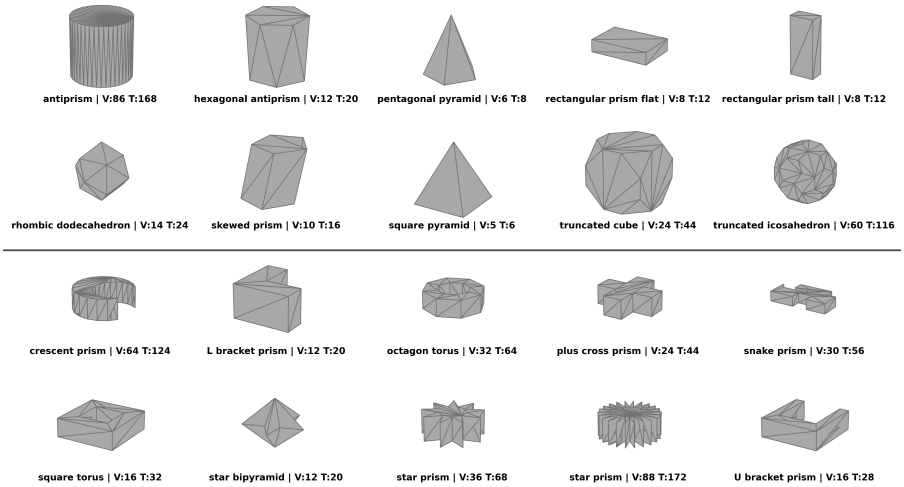
Mesh Tokenization. We also ablate a triangle tokenization analogous to RenderFormer [60], where each token is the $\mathbb{R}^{3\times3}$ vertex coordinates of a mesh triangle face, flattened to 9D. In theory, triangle tokens offer a richer local primitive than isolated vertices by implicitly encoding edge geometry, area, and a normal direction, which may simplify learning local surface cues. However, this induces redundant predictions as each true vertex appears in multiple triangles. Though we scatter-add and average into one vertex output at inference to ensure connectivity, vertex consistency is implicit during training and contributes to the poorer performance in table 7, especially for object rigidity.

I Limitations and Future Work

PHYSIFORMER demonstrates a unified framework for future mechanics prediction across multiple objects and materials. Currently, due to training dataset specifications, PHYSIFORMER supports generation over 49 frames and performs best with up to 356 vertices. We aim to incorporate Diffusion Forcing [13] for autoregressive diffusion to expand the length of generation. We also want to explore spatial compression with latent encoding [15] or learning a latent variational autoencoder like [62] for motion to support even more complex meshes without significantly increasing compute burden. Finally, PHYSIFORMER is fully data-driven and trained solely with the diffusion loss. This loss can be augmented with physical inductive biases, such as continuous collision detection (CCD), to mitigate failures like those in fig. 8, or with tailored losses to reduce rare object-orientation discontinuities in generated sequences.



(a) Mesh templates in dataset D_1 used to train PHYSIFORMER-L-10k and PHYSIFORMER-B-1k. Each mesh contains 4-20 vertices.



(b) Mesh templates in datasets D_2–D_4, used together with the templates from D_1 to train PHYSIFORMER. Each mesh contains 4-88 vertices.



(c) Out-of-distribution complex meshes with far more vertices per object used in PHYSIFORMER inference.

Figure 6: Mesh templates and real-world geometries used for dataset generation and out-of-distribution inference.

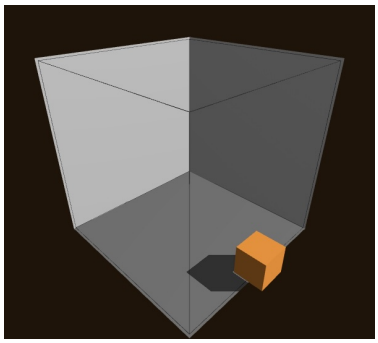


Figure 7: Physics simulator failure cases occur when boundary contacts are imperfectly resolved, allowing objects to escape the bounding box, especially at high velocities or with fewer simulation substeps.

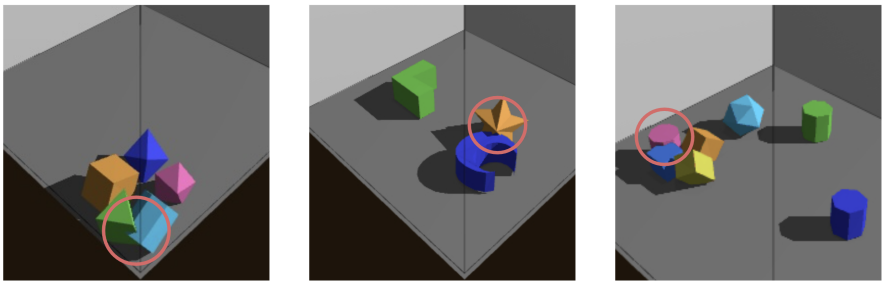


Figure 8: Examples of object overlap during inference

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 scope and contributions all center around showing PHYSIFORMER as a promising method for physical dynamics prediction using coordinate space diffusion, and we provide rigorous methodology and experimental evidence to show this. Moreover, all claims in the abstract and introduction regarding performance against baseline auto-regressive methods are shown quantitatively across 3 evaluation metrics that are either widely used in the literature (e.g.. MSE) or are physically grounded metrics (e.g.. Rigidity Loss and Momentum Error) created by the authors and qualitatively displayed in figures and supplemental video visualizations.

   Guidelines:

   - The answer [N/A] 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 [N/A] 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: We discuss limitations and failure cases of our methodology (and future work to tackle them) in both the conclusion section 5 and section I in the appendix. In section 4.4, we also acknowledge that the evaluation metrics are a mere proxy for the physical plausibility of model outputs and supplement quantitative results with direct qualitative visualizations. We also discuss computational efficiency of our proposed method in section B.

   Guidelines:

   - The answer [N/A] 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: [N/A]

   Justification: The paper does not have theoretical results. We do cite and define evaluation metrics in Section 4.4 and follow good practice of justifying and citing referenced theoretical claims as well as their assumptions.

   Guidelines:

   - The answer [N/A] 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: Yes, we carefully and rigorously describe all details of PHYSIFORMER's architecture in Section 3 and Section A.1 as well as training setup and hyperparameters in Section 4.2. We do the same for our baseline autoregressive model implementation Φ_{AR} in Section A.2. Code and checkpoints are released at `https://github.com/yimingc9/PhysiFormer`, and the full training dataset will be released shortly.

   Guidelines:

   - The answer [N/A] 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: [Yes]

   Justification: We have released model code, pretrained checkpoints, and a toy training example at `https://github.com/yimingc9/PhysiFormer`, following the guidelines outlined below; the full training dataset will be released there shortly.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.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://neurips.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) necessary to understand the results?

   Answer: [Yes]

   Justification: We do so for our method PHYSIFORMER in section 4.2 and section B. We also detail compute needed for baseline methods in section A.2 and section B.

   Guidelines:

   - The answer [N/A] 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: In table 2, for each metric, we show mean and standard deviation of PHYSIFORMER across 5 generations per sample to capture average distributional behavior of our model. All baselines are deterministic autoregressive rollout so this analysis does not apply.

   Guidelines:

   - The answer [N/A] 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 do so for our method PHYSIFORMER in section 4.2 and section B. We also detail compute needed for baseline methods in section A.2 and section B.

   Guidelines:

   - The answer [N/A] 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 carefully reviewed the Code of Ethics and confirm that we complied with them in the research process.

   Guidelines:

- The answer [N/A] 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: In the abstract, introduction, and conclusion, we discussed PHYSIFORMER's potentially applicability in domains like robotics, graphics, and physical design. A potential negative impact is that erroneous physical predictions could pose safety risks if used for robot planning without validation against real-world or simulated physics.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] 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., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: Our method generates data with outputs akin to widely available physical simulators. We also created training data using a popular physical simulation tool — Genesis.

    Guidelines:

    - The answer [N/A] 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: All original owners of assets are cited, and we follow their respective license and terms of use guidance.

    Guidelines:

    - The answer [N/A] 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: [Yes]

    Justification: The released code and checkpoints are documented in the repository, and the training dataset will be documented likewise upon its release shortly, following the guidelines below.

    Guidelines:

    - The answer [N/A] 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: [N/A]

    Justification: The paper does not involve crowdsourcing and research with human subjects.

    Guidelines:

    - The answer [N/A] 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: [N/A]

    Justification: The paper does not involve crowdsourcing and research with human subjects.

    Guidelines:

    - The answer [N/A] 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 rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: LLMs are only used for writing, editing, or formatting purposes (e.g. creating the visualization webpage in the supplemental materials). LLMs are also used to aid and confirm implementation of standard methods, not impacting the core methodology, scientific rigor, or originality of the research.

    Guidelines:

    - The answer [N/A] 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 in the NeurIPS handbook for what should or should not be described.