A Neural Material Point Method for Particle-based Simulations

Omer Rochman Sharabi [*1] **Sacha Lewin** [*1] **Gilles Louppe** [1]

Abstract

Mesh-free Lagrangian methods are widely used for simulating fluids, solids, and their complex interactions due to their ability to handle large deformations and topological changes. These physics simulators, however, require substantial computational resources for accurate simulations. To address these issues, deep learning emulators promise faster and scalable simulations, yet they often remain expensive and difficult to train, limiting their practical use. Inspired by the Material Point Method (MPM), we present NeuralMPM, a neural emulation framework for particle-based simulations. NeuralMPM interpolates Lagrangian particles onto a fixed-size grid, computes updates on grid nodes using image-to-image neural networks, and interpolates back to the particles. Similarly to MPM, NeuralMPM benefits from the regular voxelized representation to simplify the computation of the state dynamics, while avoiding the drawbacks of mesh-based Eulerian methods. We demonstrate the advantages of NeuralMPM on several datasets, including fluid dynamics and fluid-solid interactions. Compared to existing methods, NeuralMPM reduces training times from days to hours, while achieving comparable or superior long-term accuracy, making it a promising approach for practical forward and inverse problems. Our code and experiments will be made available soon.

1. Introduction

The Navier-Stokes equations describe the time evolution of fluids and their interactions with solid materials. As analytical solutions rarely exist, numerical methods are required to approximate the solutions. These methods can be broadly categorized into Eulerian and Lagrangian approaches, each with its own strengths and weaknesses. On the one hand, Eulerian methods discretize the fluid domain on a fixed grid, simplifying the computation of the dynamics, but requiring high-resolution meshes to solve small-scale details in the flow. Lagrangian methods, on the other hand, represent the fluid as virtual moving particles defining the system's state, hence maintaining a high level of detail in regions of high density. While effective at handling deformations and topological changes, Lagrangian methods struggle, however, with collisions and interactions with rigid objects.

Regardless of the discretization strategy, large-scale high-resolution numerical simulations are computationally expensive, limiting their practical use in downstream tasks such as forecasting, inverse problems, or computational design. To address these issues, deep learning emulators have shown promise in accelerating simulations by learning an emulator model that can predict the system's state at a fraction of the cost. Next to their speed, neural emulators also have the strategic advantage of being differentiable, enabling their use in inverse problems and optimization tasks (Allen et al., 2022; Zhao et al., 2022; Liu et al., 2018; Colburn & Majumdar, 2021; Forte et al., 2022; Kumar et al., 2020). Moreover, they have the potential to be learned directly from real data, bypassing the costly and resource-intensive process of building a simulator (Lam et al., 2023; Pfaff et al., 2021; Lemos et al., 2023; Bourilkov, 2019; Jumper et al., 2021; He et al., 2019; Nasser & Yusof, 2023; Arpaia et al., 2021). In this direction, particle-based neural emulators (Sanchez-Gonzalez et al., 2020; Ummenhofer et al., 2020; Prantl et al., 2022) have seen success in accurately simulating fluids and generalizing to unseen environments. These emulators, however, suffer from the same issues as traditional Lagrangian methods, with collisions and interactions with rigid objects being particularly challenging. These emulators may also require long training and inference times, limiting their practical use.

Taking inspiration from the hybrid Material Point Method (MPM) (Sulsky et al., 1993; Nguyen et al., 2023) that combines the strengths of both Eulerian and Lagrangian methods, we introduce NeuralMPM, a neural emulation framework for particle-based simulations. As in MPM, NeuralMPM maintains Lagrangian particles to represent the system's state but models the system dynamics on voxelized representations. In this way, NeuralMPM benefits from a

[*]Equal contribution [1]University of Liege, Liege, Belgium. Correspondence to: Omer Rochman Sharabi <o.rochman@uliege.be>.

AI for Science Workshop at the 41^{st} International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

regular grid structure to simplify the computation of the state dynamics but avoids the drawbacks of mesh-based Eulerian methods. By interpolating the particles onto a fixed-size grid, it also bypasses the need to perform an expensive neighbor search at every timestep, substituting it with two interpolation steps based on cheap voxelization (Nourian et al., 2016; Xu et al., 2021; Prantl et al., 2022). Furthermore, by defining the system dynamics on a grid, NeuralMPM can leverage well-established neural network architectures, such as image-to-image neural networks or neural operators. The resulting inductive bias allows the model to more easily process the global and local structures of the point cloud, instead of having to discover them, and frees capacity for learning the dynamics of the system represented by the grid. Compared to previous approaches (Sanchez-Gonzalez et al., 2020; Ummenhofer et al., 2020; Prantl et al., 2022), these improvements reduce the training time from days to hours, while achieving higher or comparable accuracy.

2. Computational Fluid Dynamics

Computational fluid dynamics simulations can be classified into two broad categories, Eulerian and Lagrangian, depending on how the discretization of the continuous fluid is handled (Rakhsha et al., 2021). In Eulerian simulations, the domain is discretized with a mesh, with state variables u_i^t (such as mass or momentum) maintained at each mesh point i. Well-known examples of Eulerian simulations are the finite difference method, where the domain is divided into a uniform regular grid (also called an Eulerian grid), and the finite element method, where the domain is divided into regions, or elements, that may have different shapes and density, allowing to increase the resolution in only some areas of the domain (Iserles, 2008; Morton & Mayers, 2005). Lagrangian simulations, on the other hand, discretize the fluid as a set of virtual moving particles $\{p_i^t, u_i^t\}_{i=1}^N$, each described by its position p_i^t and state variables u_i^t that include the particle velocity v_i^t. To simulate the fluid, the particles move according to the dynamics of the system, producing a new set of particles $\{p_i^{t+1}, u_i^{t+1}\}_{i=1}^N$ at each timestep. Simulations in Lagrangian coordinates are particularly useful when the fluid is highly deformable, as the particles can move freely and adapt to the fluid's shape. Among Lagrangian methods, Smoothed Particle Hydrodynamics (SPH) is one of the most popular, where the fluid is represented by a set of particles that interact with each other through a kernel function that smooths the interactions. SPH has been widely used in large-scale astrophysical simulations, such as galactic dynamics (Wissing & Shen, 2023; Few et al., 2016) or planetary collision (Kegerreis et al., 2022), and in ocean engineering (Gotoh & Khayyer, 2016; Tan et al., 2023; Lyu et al., 2022) to model deformations and fractures, and interactions between solids and fluids (Monaghan, 2012; Vacondio et al., 2021; Lind et al., 2020).

Hybrid Eulerian-Lagrangian methods combine the strengths of both frameworks. Like Lagrangian methods, they carry the system state information via particles, thereby automatically adjusting the resolution to the local density of the system. By using a regular grid, however, they simplify gradient computation, make entity contact detection easier, and prevent cracks from propagating only along the mesh. Among hybrid methods, the Material Point Method has gained popularity for its ability to handle large deformations and topological changes. MPM combines a regular Eulerian grid with moving Lagrangian particles. It does so in four main steps: (1) the quantities carried by the particles are interpolated onto a regular grid $G^t = \mathbf{p2g}(\{p_i^t, u_i^t\})$ using a particle-to-grid (**p2g**) function, (2) the equations of motion are solved on the grid, where derivatives and other quantities are easier to compute, resulting in a new grid state $G^{t+1} = f(G^t)$, (3) the resulting dynamics are interpolated back onto the particles as $\{u_i^{t+1}\} = \mathbf{g2p}(G^{t+1}, \{p_i^t\})$, using a grid-to-particle (**g2p**) function, (4) the positions of the particles are updated by computing particle-wise velocities and using an appropriate integrator, such as Euler, i.e., $p_i^{t+1} = p_i^t + \Delta t v_i^{t+1}$. The grid values are then reset for the next step. MPM has been used in soft tissue simulations (Ionescu et al., 2005), in molecular dynamics (Lu et al., 2006), in astrophysics (Li & Liu, 2002), in fluid-membrane interactions (York II et al., 2000), and in simulating cracks (Daphalapurkar et al., 2007) and landslides (Llano Serna et al., 2015). MPM is also widely used in the animation industry, perhaps most notably in Disney's 2013 film Frozen (Stomakhin et al., 2013), where it was used to simulate snow.

Notwithstanding the success of numerical simulators, they remain expensive, slow, and, most of the time, non-differentiable. In recent years, differentiable neural emulators have shown great promise in accelerating fluid simulations, most notably in a series of works to emulate SPH simulations. Graph network-based simulators (GNS) (Sanchez-Gonzalez et al., 2020) use a graph neural network (GNN) and a graph built from the local neighborhood of the particles to predict the acceleration of the system. The approach requires building a graph out of the point cloud at every timestep to obtain structural information about the cloud, which is an expensive operation. In addition, the GNN needs to extract global information from its nodes, which is only possible with a high number of message-passing steps, resulting in a large computational graph and long training and inference times. The number of message-passing steps, together with the construction of the graph at each timestep, virtually guarantees that it will not be possible to train GNS using autoregressive rollouts. As autoregressive training is not available, the stability of the learned dynamics can be compromised, making the model prone to diverging or oscillating. Noise injection training strategies can be used to increase the stability of the rollouts, but the magnitude of the noise becomes a critical parameter. An alternative approach is the continuous convolution (CConv) (Ummenhofer et al., 2020), an extension of convolutional networks to point clouds. In this method, a convolutional kernel is applied to each particle by interpolating the values of the kernel at the positions of its neighbors, which are found via spatial hashing on GPU, a cheaper alternative to tree-based searches that allows for autoregressive training. Finally, Deep Momentum Conserving Fluids (DMCF) (Prantl et al., 2022) build upon CConv to design a momentum-conserving architecture. Nevertheless, to account for long-range interactions, the authors introduce different branches, with different fields of view, into their network. The number of branches, and their hyperparameters, need to be tuned to capture global dependencies, leading to long training times even with optimized CUDA kernels.

3. NeuralMPM

We consider a Lagrangian system evolving in time and defined by the positions p_i^t and velocities v_i^t of a set of N particles $i = 1, ..., N$. For notational simplicity, we denote with P^t and V^t the set of positions and velocities of all particles at time t and with $S^t = (P^t, V^t)$ the full state of the system. The evolution of the particles is described by a function f mapping the current state of the system to its next state $S^{t+1} = f(S^t)$. Given a starting system $S^0 = (P^0, V^0)$, its full trajectory, or rollout, is denoted by $S^{1:T}$. Our goal is to build an emulator $\hat{S}_\theta(\cdot)$ capable of predicting a full rollout $\hat{S}_\theta^{1:T}(S^0)$ of T timesteps from the initial state S^0. Following MPM, NeuralMPM operates in four steps, as illustrated in Figure 1:

Step 1: Voxelization. Using the positions P^t, the velocities V^t of the particles in the point cloud are first interpolated onto a regular fixed-size grid. This interpolation is performed through *voxelization*, which divides the domain into regular volumes (voxels). Each grid node is identified as the center of a voxel (e.g., square in 2D) in the domain, and the velocities of the particles in the voxel are averaged to give the node's velocity. Similarly, the density is computed as the normalized number of particles in the voxel. This results in the grid tensor G^t that contains the grid velocities V_g^t and density D_g^t.

Step 2: Processing. Taking advantage of the regular grid representation of the cloud, the grid velocities $\{\hat{V}^i\}_{i=t+1}^{t+m}$ of the next m timesteps are predicted using an image-to-image neural network, such as a U-Net (Ronneberger et al., 2015) or a neural operator (Li et al., 2021).

Step 3: Update of particle velocities. The predicted velocities $\hat{V}^{t+1}$ at the next timestep are then interpolated back to the particle level onto the positions P^t using bilinear interpolation. The velocity of each particle is computed as a weighted average of the four surrounding grid velocities, based on its Euclidean distance to each of them.

Step 4: Update of particle positions. Finally, the positions of the particles are updated with Euler integration using the next velocities and known current positions of the particles, that is $\hat{P}^{t+1} = P^t + \Delta t \hat{V}^{t+1}$. Steps 3 and 4 are performed m times to compute the next m positions from the set of grid velocities computed at step 2.

Additional features of the individual particles can be included in the grid tensor G^t by interpolating them in the same way as the velocities. Local, such as boundary conditions, or global, such as gravity or external forces, features are represented as grid channels. For simulations with multiple types of particles, the velocity and density of each material are interpolated independently and stacked as channels in the grid tensor G_t.

NeuralMPM is trained end-to-end on a set of trajectories $S^{0:T}$ to minimize the mean squared error $||P^{t+1} - \hat{P}_\theta^{t+1}(S^t)||_2^2$ between the ground-truth and predicted next positions of the particles. At inference time, the model is exposed to much longer sequences, which requires carefully stabilizing the rollout procedure to prevent the accumulation of large errors over time. To address this, we first make use of autoregressive training (Prantl et al., 2022; Ummenhofer et al., 2020), where the model is unrolled K times on its own predictions, producing a sequence of $\hat{S}^k = \hat{S}_\theta(\hat{S}^{k-1})$ for $k = 1, ..., K$ and initial input $\hat{S}^0 = S^0$, before backpropagating the error through the entire rollout. Unlike more costly methods that require alternative stabilization strategies, such as noise injection (Sanchez-Gonzalez et al., 2020), NeuralMPM's efficiency makes autoregressive training possible. Nevertheless, to further stabilize the training, we couple autoregressive training with time bundling (Brandstetter et al., 2022), resulting in a training strategy where the model predicts m steps $\hat{S}^{1:m}$ at once from a single initial state, inside an outer autoregressive loop of K steps of length m. We show in Section 4 that this training strategy leads to more accurate rollouts.

4. Experiments

We conduct a set of diverse experiments to demonstrate the accuracy, speed, and generalization capabilities of NeuralMPM. Specifically, we examine its robustness to hyperparameter and architectural choices through an ablation study (4.1). We compare NeuralMPM to GNS (Sanchez-Gonzalez et al., 2020) and DMCF (Prantl et al., 2022) in terms of accuracy, training time, convergence, and inference speed (4.2). We also evaluate the generalization capabilities of NeuralMPM (4.3) and illustrate how its differentiabil-

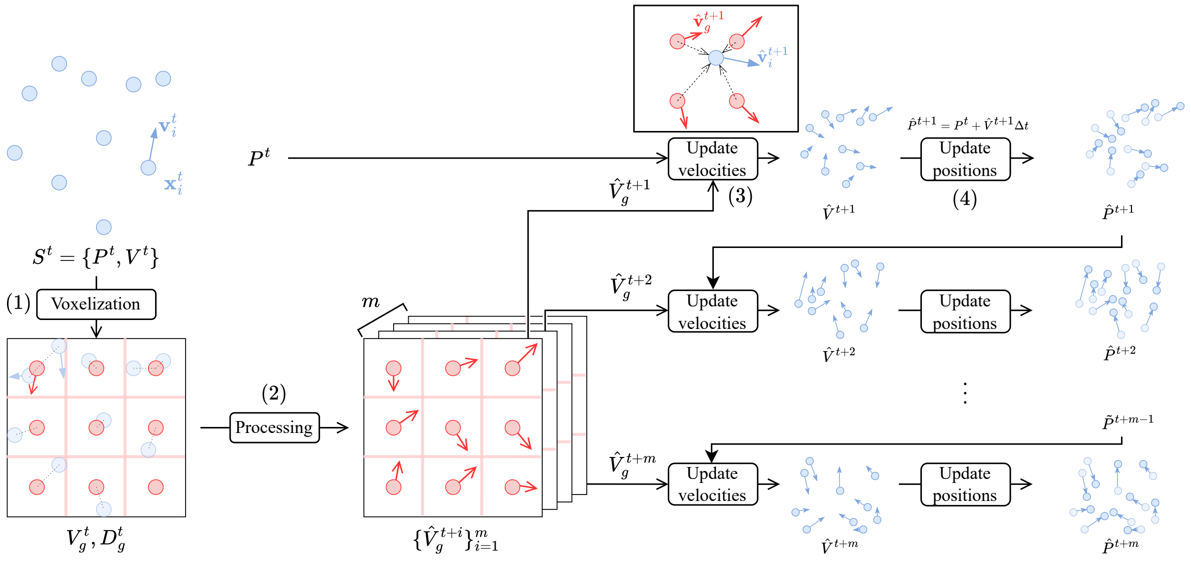

Figure 1. NeuralMPM works in 4 steps. (1) The positions P^t and velocities V^t of the particles are used to compute the velocity V_g^t and density D_g^t of each grid node through voxelization. (2) From this grid, the processor neural network predicts the grid velocities at the next m timesteps. The next m positions are computed iteratively by (3) performing bilinear interpolation of the predicted velocities onto the previous positions and (4) updating the positions using the predicted velocities.

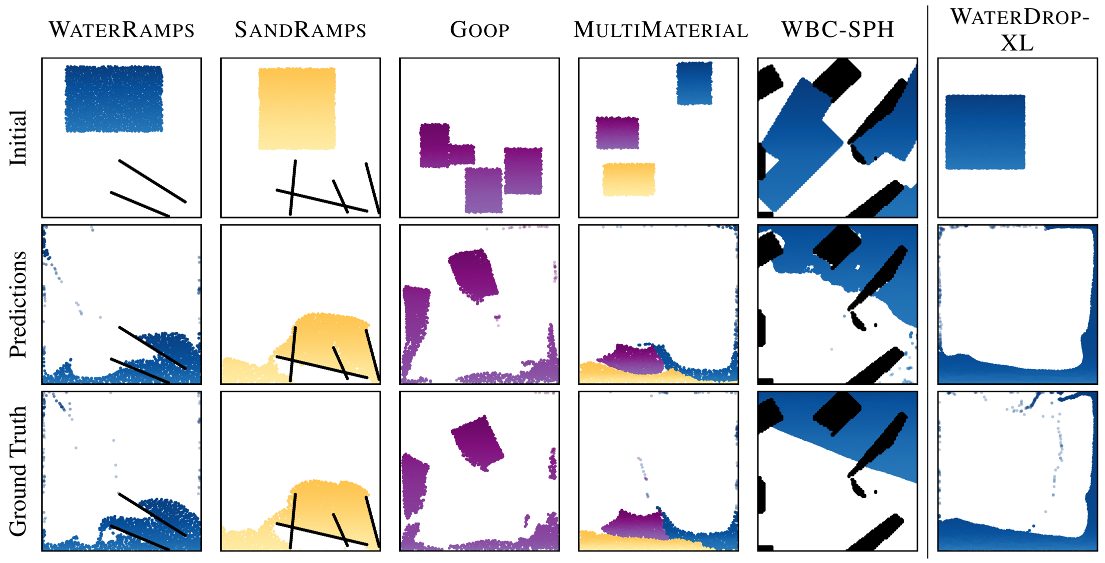

Figure 2. **Example snapshots.** We train and evaluate NeuralMPM on WATERRAMPS, SANDRAMPS and GOOP, each consisting of a single material, on MULTIMATERIAL that mixes water, sand and goop, and on WBC-SPH with more complex obstacles. We also evaluate the generalization of NeuralMPM on WATERDROP-XL, an unseen environment with more particles. NeuralMPM is able to learn various kinds of materials, their interactions, and their interactions with solid obstacles. The model is also able to generalize to unseen environments with more particles.

ity can be leveraged to solve an inverse design problem (4.4). Through these experiments, we demonstrate that NeuralMPM is a flexible, accurate, and fast method for emulating complex particle-based simulations.

Data. We consider 5 datasets with variable sequence lengths, numbers of particles, and materials. The first three datasets, WATERRAMPS, SANDRAMPS, and GOOP, contain a single material, water, sand, and goop, respectively, with different material properties. The first two datasets contain random ramp obstacles to challenge the model's generalization capacity. The fourth dataset, MULTIMATERIAL, mixes the three materials together in the same simulations. These four datasets are taken from (Sanchez-Gonzalez et al., 2020) and were simulated using the Taichi-MPM simulator (Hu et al., 2018). They each contain 1000 trajectories for training and 30 (GOOP) or 100 (WATERRAMPS, SANDRAMPS, MULTIMATERIAL) for validation and testing. The fifth dataset, WBC-SPH (Prantl et al., 2022), was generated using a high-fidelity SPH solver (Adami et al., 2012) and contains water, random obstacles, and variable gravity. It contains 30 trajectories for training and 9 for validation and testing. Rollout snapshots of NeuralMPM compared to ground truth for each dataset are shown in Figure 2.

Protocol. NeuralMPM is trained on a set of full trajectories, with varying initial conditions and number of particles. The training batches are sampled randomly in time and across sequences. We use the Adam optimizer (Kingma & Ba, 2014) with the following learning rate schedule: a linear warm-up over 100 steps from 10^{-5} to 10^{-3}, 900 steps at 10^{-3}, then a cosine annealing (Loshchilov & Hutter, 2017) for $100,000$ iterations. We use a batch size of 128, $K = 3$ autoregressive steps per iteration, bundle $m = 8$ timesteps per model call (resulting in 24 predicted states), and a grid size of 64×64. For most of our experiments, we use a U-Net (Ronneberger et al., 2015) with three downsampling blocks with a factor of 2, 64 hidden channels, a kernel size of 3, and MLPs with three hidden layers of size 64 for pixel-wise encoding and decoding into a latent space. For a fair comparison, we ran training and inference for NeuralMPM, DMCF, and GNS on the exact same hardware. GNS and DMCF were trained for a maximum of 72 hours, while NeuralMPM required 20 hours or less to converge. Further details on training can be found in Appendix A.

e add Gaussian noise to ground-truth predicted grids and particles positions, respectively of 10^{-3} and 3×10^{-4}.

4.1. Ablation study

To study the robustness of NeuralMPM to hyperparameter and architectural choices, we start with the default architecture and hyperparameters and ablate its components individually to examine their impact on performance. We vary the number K of autoregressive steps with and without noise, the number of bundled timesteps m predicted by a single model call, and the depth and number of hidden channels of the network. We also investigate adding noise to stabilize rollouts, either directly to the particles' positions or to the grid-level representation after voxelization.

Figure 3 summarizes the ablation results. A larger number K of autoregressive steps yields more accurate rollouts without the need to add noise. Indeed, injecting noise does not improve accuracy and is even detrimental for $K = 4$. Individually tuning the noise levels for grids and particles can modestly lower error rates, but is either very sensitive or negligible. The model performs better when bundling more timesteps, enabling faster rollouts as a single forward pass predicts more steps. We found $m = 8$ to be optimal with the other default hyperparameters, outperforming larger bundling. This is because more network capacity is needed to extract information for the next 16 or 32 timesteps from a single state. Instead, we opted for a shallower and narrower network to balance speed and memory footprint with performance gains. In terms of network architecture, we kept the U-Net as we found the FNO (Li et al., 2021) to underperform in most of our experiments (see Appendix B). We find the U-Net's width and depth to have a minor impact on performance, confirming that a larger network is not needed. The grid size, however, is critical. A low resolution loses fine details, while a high resolution turns meaningful structures, such as liquid blobs or walls, into isolated voxels.

Given these results, we conduct the rest of our experiments using the blue parameters in Figure 3, except for WBC-SPH. As that dataset contains more particles and has a much longer rollout, we opted for bundling more steps ($m = 16$), coupled with a more expressive depth of 4. Crucially, as the water density is higher in that dataset, we increase the grid size to 128 to capture finer interactions.

4.2. Comparison with previous work

We compare NeuralMPM against GNS (Sanchez-Gonzalez et al., 2020) and DMCF (Prantl et al., 2022). We use the official implementations and training instructions released by the authors to assess training times, inference times, as well as accuracy. We compare against both GNS and DMCF on WATERRAMPS, SANDRAMPS, and GOOP. We also compare against GNS on MULTIMATERIAL, and against DMCF on WBC-SPH.

Accuracy. We report quantitative results comparing the long-term accuracy in Table 1 and show trajectories of NeuralMPM in Figure 2. Additional snapshots and comparisons with the baselines can be found in Appendix B, and in Appendix C. On the mono-material datasets WATERRAMPS, SANDRAMPS, and GOOP, NeuralMPM performs compet-

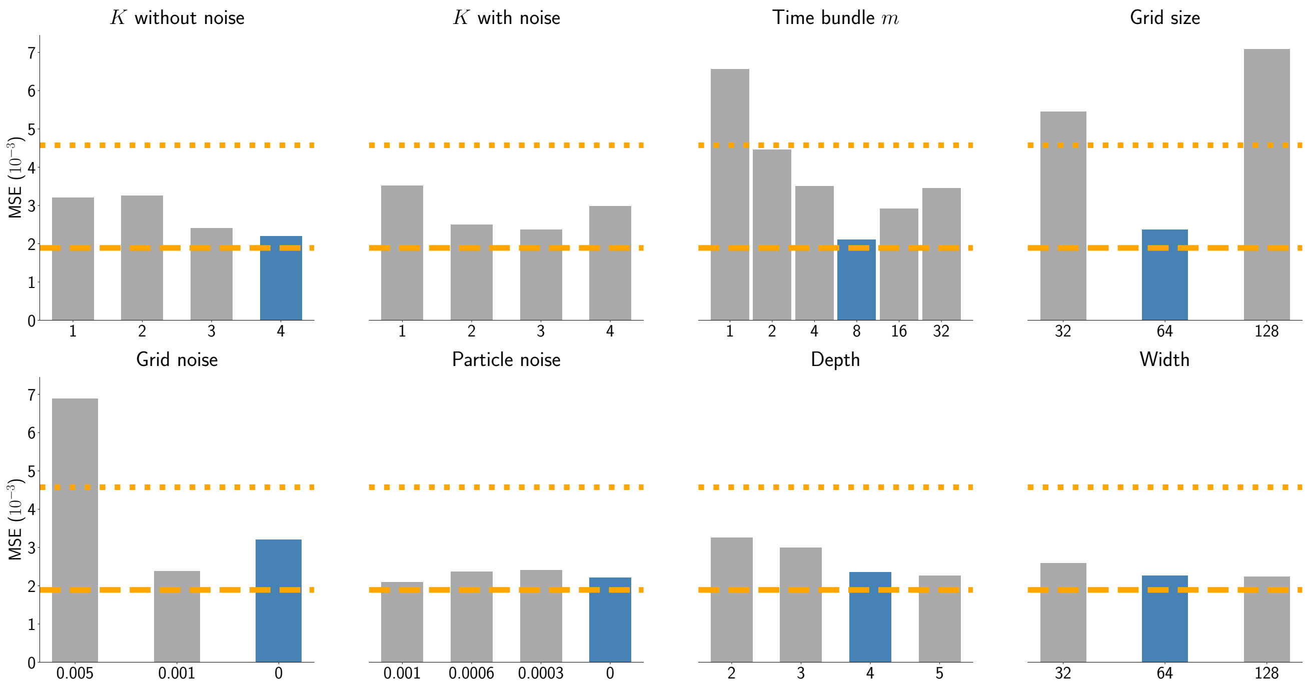

Figure 3. **Ablation results.** Mean squared error (MSE) of full rollouts on unseen test data for GOOP. The default parameters are in blue. The dashed orange line shows the best rollout MSE of GNS as reported in (Sanchez-Gonzalez et al., 2020) on the same data (1.89×10^{-3}), while the dotted line (4.57×10^{-3}) indicates the MSE we obtained for GNS after 72h (5M+ training steps). NeuralMPM is robust to hyperparameter changes, with the biggest effects coming from the number of timesteps bundled together (m) and grid noise. For a rollout of length T, the model is called T/m times, meaning lower values of m require maintaining stability for longer. Autoregressive training coupled with time bundling is sufficient to stabilize the model, eliminating the need for noise injection. Although GNS reportedly outperforms NeuralMPM by a small margin, these results could not be reproduced in our experiments.

itively with GNS and better than DMCF in terms of mean squared error (MSE). For MULTIMATERIAL, NeuralMPM reduces the MSE by almost half, which we attribute to it being a hybrid method, known to better handle interactions, mixing, and collisions between different materials. In terms of Earth Mover's Distance (EMD), NeuralMPM outperforms both baselines, suggesting that NeuralMPM is better at capturing the spatial distribution of the particles. However, on WBC-SPH, NeuralMPM falls behind DMCF. This dataset is challenging as SPH solvers outperform MPM solvers in this domain. It is also limited to only 30 training simulations, compared to 1000 in others, with trajectories reaching a steady state after 1000 timesteps (out of 3200), strongly reducing the quantity of data with information about the dynamics. The dataset also features variable gravity across simulations, which DMCF directly integrates as an external force in its update step. This approach could improve NeuralMPM's performance on this dataset by offloading learning to apply gravity from the model.

Training. In Figure 4, we report the evolution of the mean squared error of full emulated rollouts on the held-out test set during training, for each method, along with predicted snapshots at increasing training durations. NeuralMPM converges significantly faster than both baselines while reaching lower error rates. Furthermore, the convergence of the training procedure and quality of the architecture can be assessed much earlier during training, effectively saving compute and enabling the development of more refined final models. Moreover, NeuralMPM is also more memory-efficient, which enables the use of higher batch sizes of 128, as opposed to only 2 in GNS and DMCF.

Inference time. Table 2 reports the average inference time for one model call and a full rollout on WATERRAMPS. NeuralMPM, despite predicting multiple frames per model call, achieves faster inference time per call than the baselines while maintaining comparable accuracy. Time bundling results in an even larger gap for full rollouts. Additionally, our method's improved memory efficiency allows for over 100 parallel simulations, whereas GNS and DMCF face memory limitations with more than a few.

Data (Simulator)	N	T	NeuralMPM		GNS		DMCF	
			MSE↓	EMD↓	MSE↓	EMD↓	MSE↓	EMD↓
WaterRamps (MPM)	2.3k	600	13.92	**68**	**13.13**	91	20.45	105
SandRamps (MPM)	3.3k	400	3.12	**61**	**3.11**	84	7.60	97
Goop (MPM)	1.9k	400	**2.18**	**55**	4.57	81	5.25	85
MultiMaterial (MPM)	2k	1000	**9.6**	**66**	14.79	105	-	-
WBC-SPH (SPH)	15k	3200	55.4	-	-	-	**47.8**	-

Table 1. Number of particles N, sequence length T, rollout MSE ($\times 10^{-3}$) and rollout EMD ($\times 10^{-3}$).

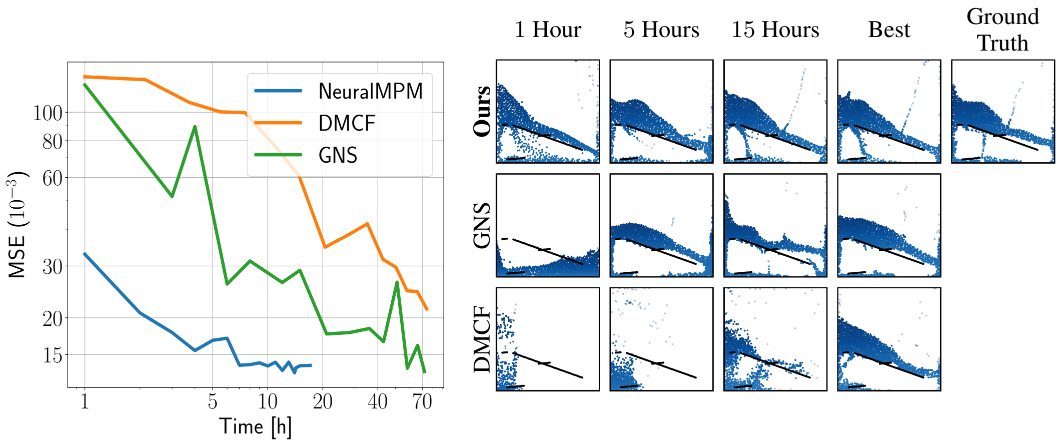

Figure 4. **Training convergence.** (Left) NeuralMPM trains and converges much faster than GNS and DMCF. (Right) Snapshots of models trained for increasing durations then unrolled until the same timestep on a held-out simulation. For a fair comparison, out-of-bounds particles in GNS and DMCF were clamped.

	NeuralMPM (Ours)	**GNS**	**DMCF**
One model call ($T = 1$)	8.04 ± 0.43	13.91 ± 2.83	12.85 ± 8.41
Rollout ($T = 600$)	606.63 ± 7.69	8344.94 ± 1700.8	9448.17 ± 804.07

Table 2. Inference time (in ms) of NeuralMPM and baselines on WaterRamps. Times were averaged over all validation trajectories. NeuralMPM predicts 8 frames in a single model call and still outperforms the two baselines per call, which further widens the gap for the total rollout time. For comparison, GNS reports a computation time of 71ms to simulate one timestep with Taichi-MPM.

4.3. Generalization

One notable advantage of NeuralMPM is its invariance to the number of particles, as the transition model only processes the voxelized representation. To demonstrate this, we train a model on WaterRamps, which contains about 2.3k particles and 600 timesteps, and evaluate it on WaterDrop-XL, which features about four times more particles, 1000 timesteps, and no obstacles. An example snapshot is displayed in Figure 2. The larger number of particles only affects interpolation between the grid and particles, which has a negligible impact on total inference time, making the model nearly as fast despite 4 times more particles. We also validate this quantitatively by comparing a model trained directly on WaterDrop-XL with a model trained solely on WaterRamps. With the same training budget, the latter achieves a lower MSE at 20.92×10^{-3} against 28.09×10^{-3}.

4.4. Inverse design problem

Finally, we demonstrate the application of NeuralMPM for inverse problems on a toy inverse design task that consists in optimizing the direction of a ramp to make the particles reach a target location, similar to (Allen et al., 2022). We place a blob of water at different starting locations, and we then place a ramp at some location, with a random initial angle α. The goal is to spin that ramp by tuning α in order to make the water end up at a desired location. The main challenges of this task are the long-range time horizon of the goal and the presence of nonlinear physical dynamics. We proceed by selecting the point where we want the water to end up and compute the average distance between the point and particles at the last simulation frame (Tests 1 & 2) or at an intermediate frame (Test 3). We then minimize the distance via gradient descent, leveraging the differentiability

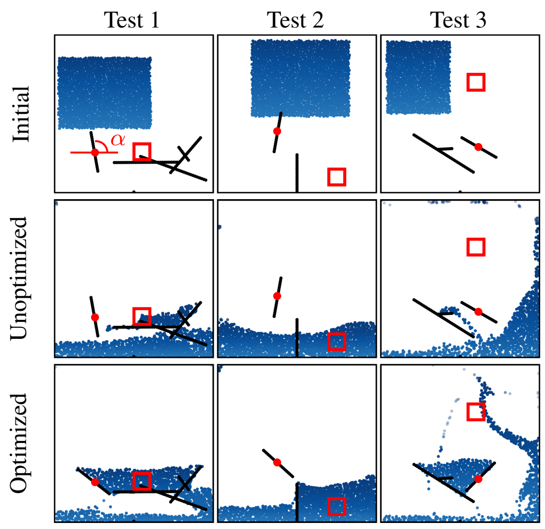

Figure 5. **Inverse design problem.** We exploit NeuralMPM's differentiability to optimize the angle α of a ramp, anchored at the red dot, in order to get the water close to the red square region.

of NeuralMPM to solve this inverse design problem. We show three example optimizations in Figure 5.

5. Conclusion

Summary. We presented NeuralMPM, a neural emulation framework for particle-based simulations inspired by the hybrid Eulerian-Lagrangian Material Point Method. We have shown its effectiveness in simulating a variety of materials and interactions, its ability to generalize to larger systems and its use in inverse problems. Crucially, NeuralMPM trains in a fraction of the time it takes to train alternative approaches, and is substantially faster at inference time. By interpolating particles onto a fixed-size grid, global information is distilled into a voxelized representation that is easier to learn and process with powerful image-to-image models. The use of voxelization allows NeuralMPM to bypass expensive graph constructions, and the interpolation leads to easier generalization to a larger number of particles and constant runtime. The lack of expensive graph construction and message passing also allows for more autoregressive steps and parallel rollouts.

Limitations. Like other approaches, NeuralMPM is limited by the computation used to process the structure of the point cloud. In our case, voxelization means we cannot deal with particles that lie outside of the domain and are limited to regular grids. Additionally, the size of the voxels is coupled to the density of the fluid, understood as the number of particles per given volume. If the voxels are too large, the model will fail to reproduce finer details. On the other hand, if they are too small, the model might be overwhelmed by the lack of local structure. Unsurprisingly, NeuralMPM performs better on domains where numerical MPM methods are strong and will struggle in domains not suited for MPM.

Future work. Our work is only a first step towards hybrid Eulerian-Lagrangian neural emulators, leaving many avenues for future research. Extending NeuralMPM to 3D systems is a natural continuation of this work. Future studies could also explore alternative particle-to-grid and grid-to-particle functions, like the non-uniform Fourier transform (Fessler & Sutton, 2003), or more sophisticated interpolation methods from classical MPM literature (Nguyen et al., 2023). Additionally, incorporating features commonly used in classical SPH and MPM simulations, such as viscosity, pressure, or temperature, presents another promising direction for future work. A less traditional direction is to make NeuralMPM probabilistic and encode richer distributional information about the particles in the grid nodes, instead of maintaining only a mean value. This could potentially improve NeuralMPM's ability to resolve subgrid phenomena. Advances in Lagrangian Particle Tracking (Schröder & Schanz, 2023) will eventually make it possible to create datasets from real-world data, enabling the training of NeuralMPM directly from data without the need for the costly design process of a numerical simulator. Lastly, losses that take into account the distribution of particles and are invariant under particle permutation, such as the earth mover's distance, are increasingly becoming an option to consider, as more efficient differentiable approximations emerge.

Impact statement. While the immediate societal impact of this work will likely be minor, learned emulators have the potential to have a profoundly positive social impact by facilitating large weather simulations, efficient inverse design methods, and scientific advance.

References

Adami, S., Hu, X., and Adams, N. A generalized wall boundary condition for smoothed particle hydrodynamics. *Journal of Computational Physics*, 231(21):7057–7075, 2012.

Allen, K., Lopez-Guevara, T., Stachenfeld, K. L., Sanchez Gonzalez, A., Battaglia, P., Hamrick, J. B., and Pfaff, T. Inverse design for fluid-structure interactions using graph network simulators. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 13759–13774. Curran Associates, Inc., 2022.

Arpaia, P., Azzopardi, G., Blanc, F., Bregliozzi, G., Buffat, X., Coyle, L., Fol, E., Giordano, F., Giovannozzi, M., Pieloni, T., Prevete, R., Redaelli, S., Salvachua, B., Salvant, B., Schenk, M., Camillocci, M. S., Tomás, R., Valentino, G., Van der Veken, F., and Wenninger, J. Machine learning for beam dynamics studies at the cern large hadron collider. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 985:164652, January 2021.

Bourilkov, D. Machine and deep learning applications in particle physics. *International Journal of Modern Physics A*, 34(35):1930019, December 2019.

Brandstetter, J., Worrall, D. E., and Welling, M. Message passing neural PDE solvers. In *International Conference on Learning Representations*, 2022.

Colburn, S. and Majumdar, A. Inverse design and flexible parameterization of meta-optics using algorithmic differentiation. *Communications Physics*, 4(1):65, 3 2021.

Cuturi, M., Meng-Papaxanthos, L., Tian, Y., Bunne, C., Davis, G., and Teboul, O. Optimal transport tools (ott): A jax toolbox for all things wasserstein. *arXiv preprint arXiv:2201.12324*, 2022.

Daphalapurkar, N. P., Lu, H., Coker, D., and Komanduri, R. Simulation of dynamic crack growth using the generalized interpolation material point (gimp) method. *International Journal of Fracture*, 143(1):79–102, 01 2007.

Fessler, J. and Sutton, B. Nonuniform fast fourier transforms using min-max interpolation. *IEEE Transactions on Signal Processing*, 51(2):560–574, 2003.

Few, C. G., Dobbs, C., Pettitt, A., and Konstandin, L. Testing hydrodynamics schemes in galaxy disc simulations. *Monthly Notices of the Royal Astronomical Society*, 460 (4):4382–4396, 05 2016.

Fey, M. and Lenssen, J. E. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

Forte, A. E., Hanakata, P. Z., Jin, L., Zari, E., Zareei, A., Fernandes, M. C., Sumner, L., Alvarez, J., and Bertoldi, K. Inverse design of inflatable soft membranes through machine learning. *Advanced Functional Materials*, 32 (16):2111610, 2022.

Gotoh, H. and Khayyer, A. Current achievements and future perspectives for projection-based particle methods with applications in ocean engineering. *Journal of Ocean Engineering and Marine Energy*, 2(3):251–278, 8 2016.

He, S., Li, Y., Feng, Y., Ho, S., Ravanbakhsh, S., Chen, W., and Póczos, B. Learning to predict the cosmological structure formation. *Proceedings of the National Academy of Sciences*, 116(28):13825–13832, June 2019.

Hu, Y., Fang, Y., Ge, Z., Qu, Z., Zhu, Y., Pradhana, A., and Jiang, C. A moving least squares material point method with displacement discontinuity and two-way rigid body coupling. *ACM Trans. Graph.*, 37(4), 07 2018.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. pmlr, 2015.

Ionescu, I., Guilkey, J., Berzins, M., Kirby, R. M., and Weiss, J. Computational simulation of penetrating trauma in biological soft tissues using the material point method. *Stud Health Technol Inform*, 111:213–218, 2005.

Iserles, A. *A First Course in the Numerical Analysis of Differential Equations*. Cambridge Texts in Applied Mathematics. Cambridge University Press, 2 edition, 2008.

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., Bridgland, A., Meyer, C., Kohl, S. A. A., Ballard, A. J., Cowie, A., Romera-Paredes, B., Nikolov, S., Jain, R., Adler, J., Back, T., Petersen, S., Reiman, D., Clancy, E., Zielinski, M., Steinegger, M., Pacholska, M., Berghammer, T., Bodenstein, S., Silver, D., Vinyals, O., Senior, A. W., Kavukcuoglu, K., Kohli, P., and Hassabis, D. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 8 2021.

Kegerreis, J. A., Ruiz-Bonilla, S., Eke, V. R., Massey, R. J., Sandnes, T. D., and Teodoro, L. F. A. Immediate origin of the moon as a post-impact satellite. *The Astrophysical Journal Letters*, 937(2):L40, 2022.

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

Kumar, S., Tan, S., Zheng, L., and Kochmann, D. M. Inverse-designed spinodoid metamaterials. *npj Computational Materials*, 6(1):73, 6 2020.

Lam, R., Sanchez-Gonzalez, A., Willson, M., Wirnsberger, P., Fortunato, M., Alet, F., Ravuri, S., Ewalds, T., Eaton-Rosen, Z., Hu, W., Merose, A., Hoyer, S., Holland, G., Vinyals, O., Stott, J., Pritzel, A., Mohamed, S., and Battaglia, P. Learning skillful medium-range global weather forecasting. *Science*, 382(6677):1416–1421, 2023.

Lemos, P., Jeffrey, N., Cranmer, M., Ho, S., and Battaglia, P. Rediscovering orbital mechanics with machine learning. *Machine Learning: Science and Technology*, 4(4):045002, 10 2023.

Li, S. and Liu, W. K. Meshfree and particle methods and their applications. *Applied Mechanics Reviews*, 55(1): 1–34, 01 2002.

Li, Z., Kovachki, N. B., Azizzadenesheli, K., liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations*, 2021.

Lind, S. J., Rogers, B. D., and Stansby, P. K. Review of smoothed particle hydrodynamics: towards converged lagrangian flow modelling. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 476(2241):20190801, 2020.

Liu, Z., Zhu, D., Rodrigues, S. P., Lee, K.-T., and Cai, W. Generative model for the inverse design of metasurfaces. *Nano Letters*, 18(10):6570–6576, 10 2018.

Llano Serna, M. A., Muniz-de Farias, M., and Martínez-Carvajal, H. E. Numerical modelling of alto verde landslide using the material point method. *DYNA*, 82(194): 150–159, 11 2015.

Loshchilov, I. and Hutter, F. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017.

Lu, H., Daphalapurkar, N., Wang, B., Roy, S., and Komanduri, R. Multiscale simulation from atomistic to continuum – coupling molecular dynamics (md) with the material point method (mpm). *Philosophical Magazine*, 86:2971–2994, 2006.

Lyu, H.-G., Sun, P.-N., Huang, X.-T., Zhong, S.-Y., Peng, Y.-X., Jiang, T., and Ji, C.-N. A review of sph techniques for hydrodynamic simulations of ocean energy devices. *Energies*, 15(2), 2022.

Monaghan, J. Smoothed particle hydrodynamics and its diverse applications. *Annual Review of Fluid Mechanics*, 44(Volume 44, 2012):323–346, 2012.

Morton, K. W. and Mayers, D. F. *Numerical Solution of Partial Differential Equations: An Introduction*. Cambridge University Press, 2 edition, 2005.

Nasser, M. and Yusof, U. K. Deep learning based methods for breast cancer diagnosis: A systematic review and future direction. *Diagnostics (Basel)*, 13(1), January 2023.

Nguyen, V. P., Vaucorbeil, A. d., and Bordas, S. *The Material Point Method: Theory, Implementations and Applications (Scientific Computation) 1st ed. 2023 Edition*. 02 2023. ISBN 3031240693.

Nourian, P., Gonçalves, R., Zlatanova, S., Ohori, K. A., and Vu Vo, A. Voxelization algorithms for geospatial applications: Computational methods for voxelating spatial datasets of 3d city models containing 3d surface, curve and point data models. *MethodsX*, 3:69–86, 2016.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Pfaff, T., Fortunato, M., Sanchez-Gonzalez, A., and Battaglia, P. Learning mesh-based simulation with graph networks. In *International Conference on Learning Representations*, 2021.

Prantl, L., Ummenhofer, B., Koltun, V., and Thuerey, N. Guaranteed conservation of momentum for learning particle-based fluid dynamics. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 6901–6913. Curran Associates, Inc., 2022.

Rakhsha, M., Kees, C. E., and Negrut, D. Lagrangian vs. eulerian: An analysis of two solution methods for free-surface flows and fluid solid interaction problems. *Fluids*, 6(12), 2021.

Ronneberger, O., Fischer, P., and Brox, T. U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, pp. 234–241. Springer, 2015.

Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J., and Battaglia, P. Learning to simulate complex physics with graph networks. In III, H. D. and Singh, A. (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 8459–8468. PMLR, 7 2020.

Schröder, A. and Schanz, D. 3d lagrangian particle tracking in fluid mechanics. *Annual Review of Fluid Mechanics*, 55(Volume 55, 2023):511–540, 2023. ISSN 1545-4479.

Stomakhin, A., Schroeder, C., Chai, L., Teran, J., and Selle, A. A material point method for snow simulation. *ACM Trans. Graph.*, 32(4), 07 2013.

Sulsky, D., Chen, Z., and Schreyer, H. L. A particle method for history-dependent materials. *Computer Methods in Applied Mechanics and Engineering*, 118:179–196, 1993.

Tan, Z., Sun, P.-N., Liu, N.-N., Li, Z., Lyu, H.-G., and Zhu, R.-H. Sph simulation and experimental validation of the dynamic response of floating offshore wind turbines in waves. *Renewable Energy*, 205:393–409, 2023.

Ummenhofer, B., Prantl, L., Thuerey, N., and Koltun, V. Lagrangian fluid simulation with continuous convolutions. In *International Conference on Learning Representations*, 2020.

Vacondio, R., Altomare, C., De Leffe, M., Hu, X., Le Touzé, D., Lind, S., Marongiu, J.-C., Marrone, S., Rogers, B. D., and Souto-Iglesias, A. Grand challenges for smoothed particle hydrodynamics numerical schemes. *Computational Particle Mechanics*, 8(3):575–588, 5 2021.

Wissing, R. and Shen, S. Numerical dependencies of the galactic dynamo in isolated galaxies with sph. *Astronomy and Astrophysics*, 673:A47, May 2023.

Xu, Y., Tong, X., and Stilla, U. Voxel-based representation of 3d point clouds: Methods, applications, and its potential use in the construction industry. *Automation in Construction*, 126:103675, 2021.

York II, A. R., Sulsky, D., and Schreyer, H. L. Fluid–membrane interaction based on the material point method. *International Journal for Numerical Methods in Engineering*, 48(6):901–924, 2000.

Zhao, Q., Lindell, D. B., and Wetzstein, G. Learning to solve PDE-constrained inverse problems with graph networks. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., and Sabato, S. (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 26895–26910. PMLR, 7 2022.

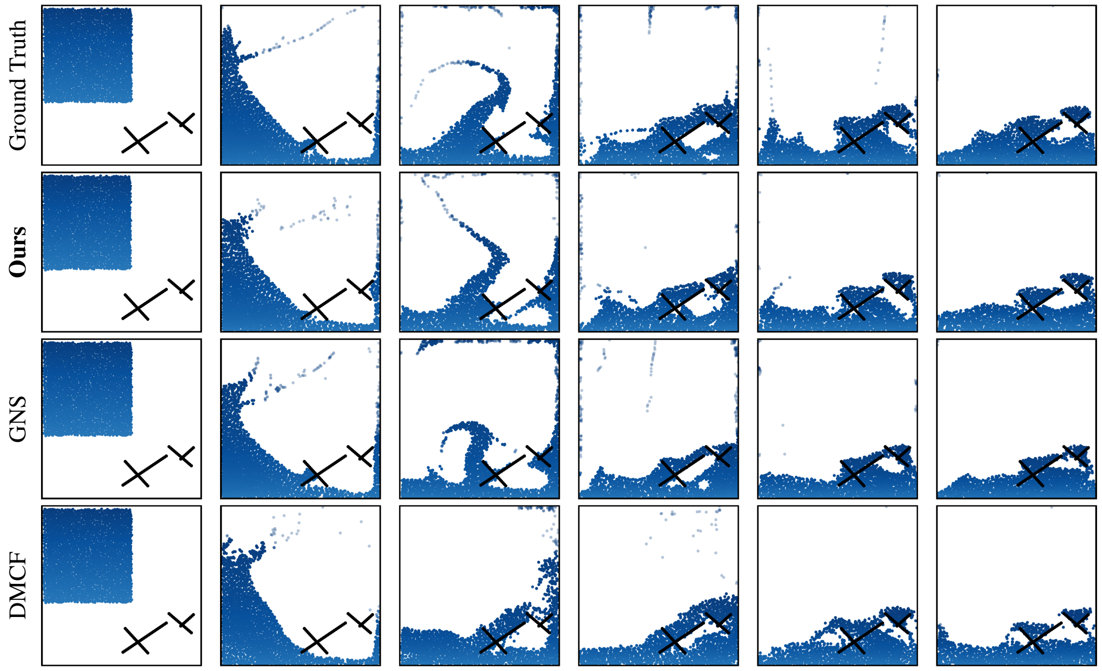

Figure 6. **Example of a WATERRAMPS trajectory against baselines.** We select a random test trajectory that was not seen during training and unroll predictions from NeuralMPM and the two baselines, starting from the same initial conditions. We display six snapshots, spaced evenly in time over the 600-step sequence.

A. Training details

Hardware. We run all our experiments using the same hardware: 4 CPUs, 24GB of RAM, and an NVIDIA RTX A5000 GPU with 24GB of VRAM. For reproducing the results of DMCF, we kept the A5000 GPU but it required up to 96GB of RAM for training.

Data Preprocessing. Similar to (Prantl et al., 2022), we slightly alter the original MPM datasets to add boundary particles, as the original data from (Sanchez-Gonzalez et al., 2020) does not have them. We define the velocity at a timestep to be $\mathbf{v}_t = \mathbf{v}_t - \mathbf{v}_{t-1}$. We therefore skip the first step during training for which no velocity is available.

Implementation. Our implementation, training scripts, experiment configurations, and instructions for reproducing results are publicly available at [URL]. We implement NeuralMPM using PyTorch (Paszke et al., 2019), and use PyTorch Geometric (Fey & Lenssen, 2019) for implementing efficient particle-to-grid functions, more specifically from the Scatter and Cluster modules. For memory efficiency, we do not store all (up to) 1,000 training trajectories in memory, and rather use a buffer of about 16 trajectories over which several epochs are performed before loading a new buffer of random trajectories.

Baselines. We use the official implementations and training instructions of GNS (Sanchez-Gonzalez et al., 2020) and DMCF (Prantl et al., 2022) to reproduce their results and conduct new experiments. More specifically, we train GNS as instructed for 5 million steps on all four datasets, using their provided configuration. For DMCF, we follow their default configurations, conducting 50k training iterations for WBC-SPH and 40k for WATERRAMPS, SANDRAMPS, and GOOP.

Normalization. We normalize the input of the model over each channel individually. We investigated computing the statistics across a buffer, resembling (Ioffe & Szegedy, 2015), and precomputing them on the whole training set and found no difference in performance. During inference, we use the precomputed statistics.

B. Supplementary results

Figure 6 shows snapshots on WaterRamps taken from rollouts of the best models for NeuralMPM, GNS, and DMCF.

Although we have used a U-Net architecture for the grid-to-grid processor, NeuralMPM can be used with any grid-to-grid processor and is not limited to that network. For example, in Figure 7 and Table 3 we present qualitative and quantitative ablation results, respectively, for NeuralMPM using an FNO network (Li et al., 2021) as the grid-to-grid processor. Results show that the FNO processor is slightly worse than the U-Net processor.

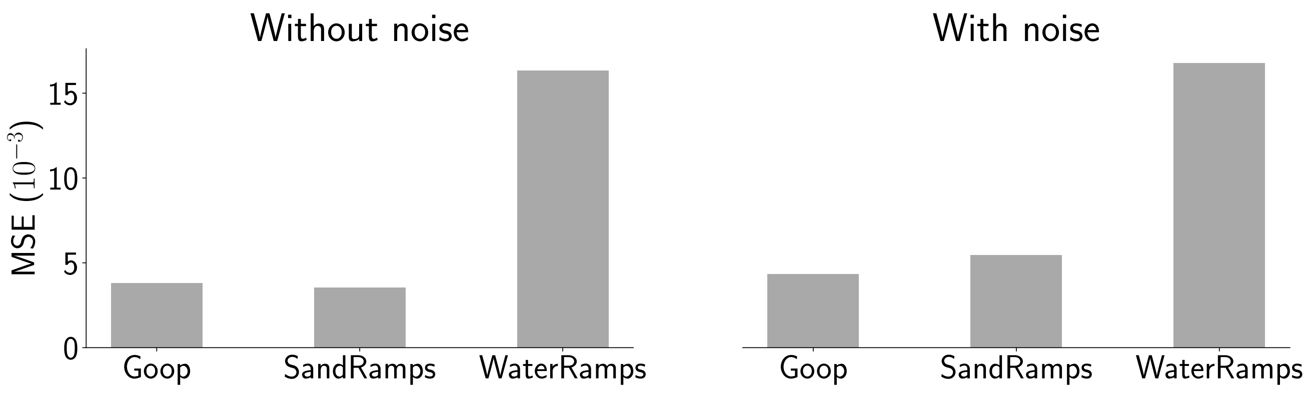

Figure 7. **FNO processor.** NeuralMPM with an FNO processor and default architecture. Rollout MSE ($\times 10^{-3}$) for different datasets.

Data	FNO without noise	FNO with noise
WATERRAMPS	16.8	16.3
SANDRAMPS	5.5	3.5
GOOP	4.3	3.8

Table 3. Rollout MSE ($\times 10^{-3}$) for NeuralMPM with an FNO processor and default architecture, with and without noise.

Table 4 reports the exact MSE rollout values that were reported in Figure 3 for GOOP.

Parameter	Value	MSE ($\times 10^{-3}$)
K (No noise)	1	3.2
	2	3.3
	3	2.4
	4	2.2
K (With noise)	1	3.5
	2	2.5
	3	2.4
	4	3.0
Time bundling m	1	6.6
	2	4.5
	4	3.5
	8	2.1
	16	2.9
	32	3.5
Grid size	32	5.5
	64	2.4
	128	7.1

Parameter	Value	MSE ($\times 10^{-3}$)
Grid noise	0	3.2
	0.001	2.4
	0.005	6.9
Particle noise	0	2.2
	0.0003	2.4
	0.0006	2.4
	0.001	2.1
U-Net Depth	2	3.3
	3	3.0
	4	2.4
	5	2.3
U-Net Width	32	2.6
	64	2.3
	128	2.2

Table 4. Ablation results for GOOP.

Finally, Figure 8 shows the error during rollouts for each dataset, both in terms of MSE and EMD. With both metrics, the error starts low and slowly accumulates over time. For the EMD, we use the Sinkhorn algorithm provided by (Cuturi et al., 2022).

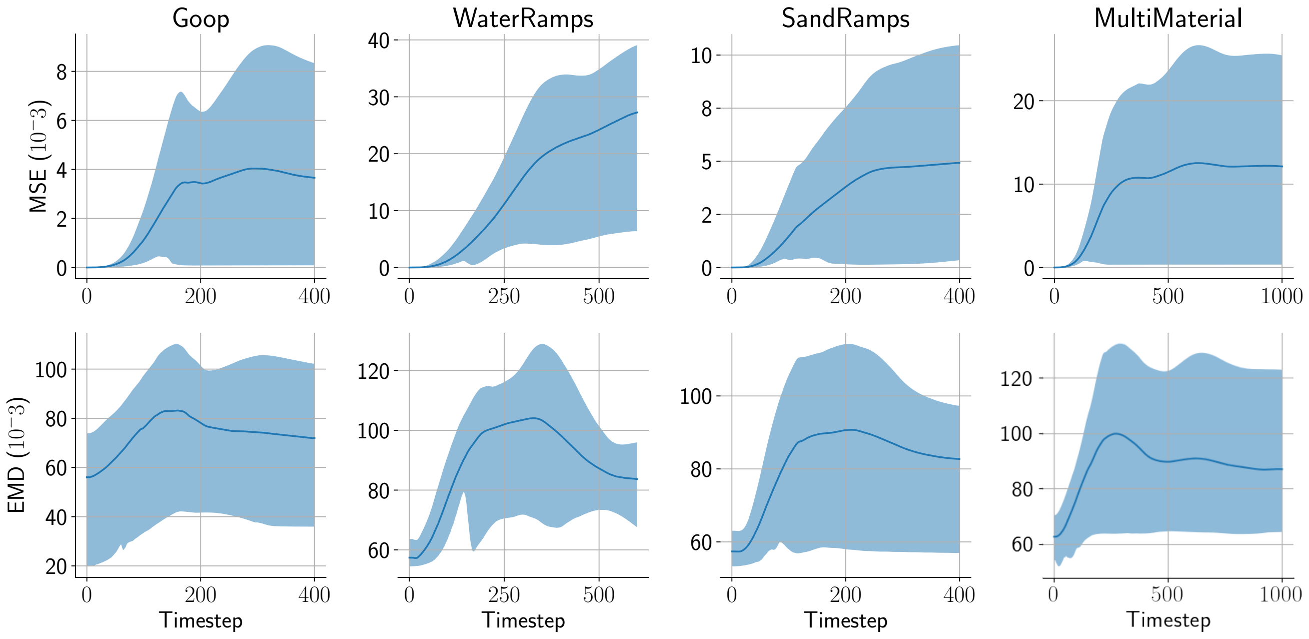

Figure 8. **Error propagation during rollout.** Shown is the mean and standard deviation of the MSE and EMD, computed over particles and simulations, at each timestep during the rollout. The accuracy decreases as errors accumulate.

C. Additional predicted trajectories

In addition to the trajectories in Figures 2 and 6, we show additional trajectories emulated with NeuralMPM for all datasets in Figures 9, 10, 11, 12, and 13. We also show trajectories on WaterDrop-XL emulated using a model that was trained on WaterRamps (see Section 4.3). We also release *videos* in the supplementary material, which we recommend watching to better see the details and limitations of NeuralMPM. This includes about 10 videos of emulated trajectories on held-out test simulations. Notably, we can observe the specific limitations of NeuralMPM on WBC-SPH, as shown in the latest trajectory depicted in Figure 13.

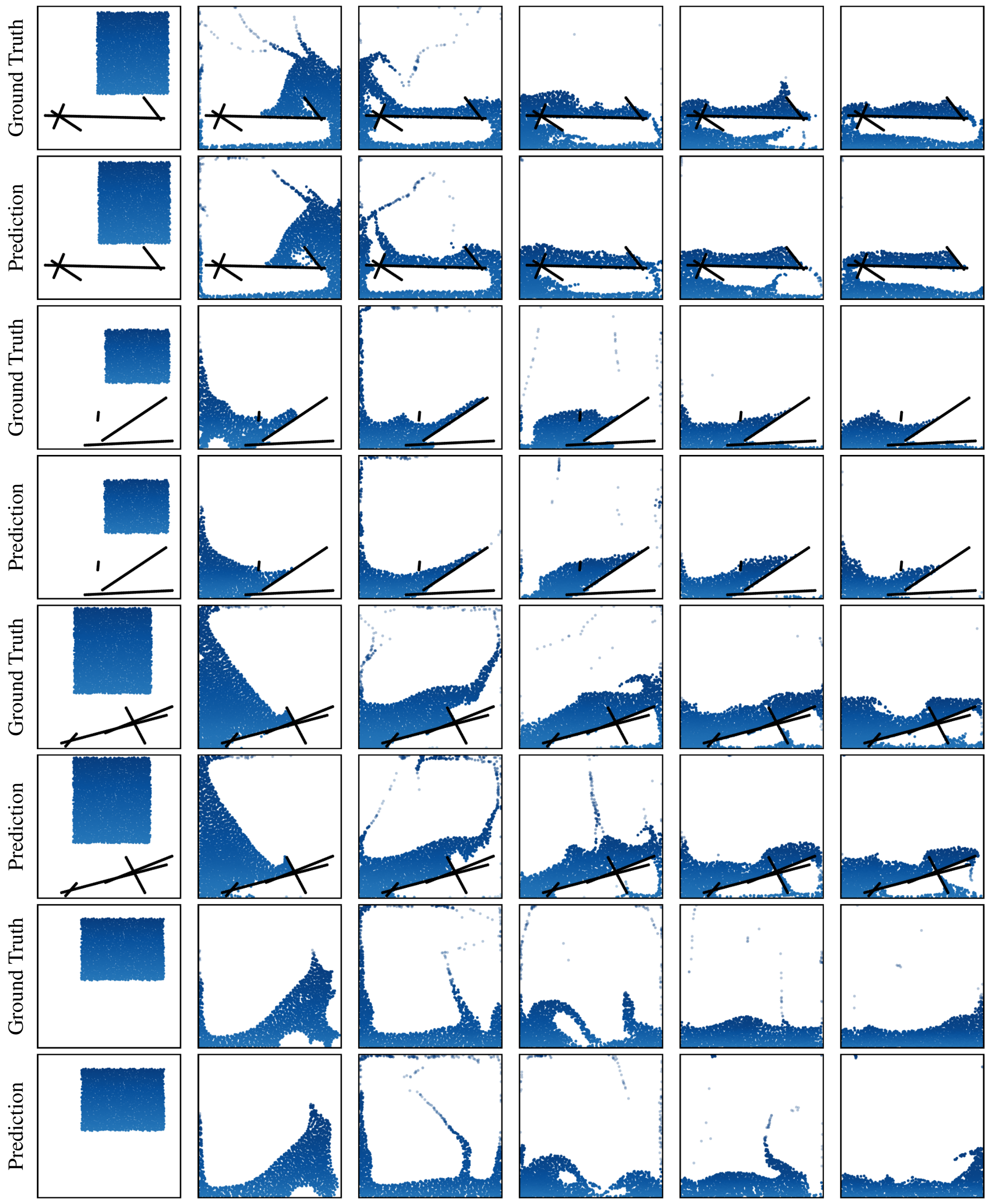

Figure 9. **Additional WATERRAMPS predicted trajectories.** Evenly spaced in time snapshots of predicted unrolled trajectories against ground truth. All trajectories are from the held-out test set.

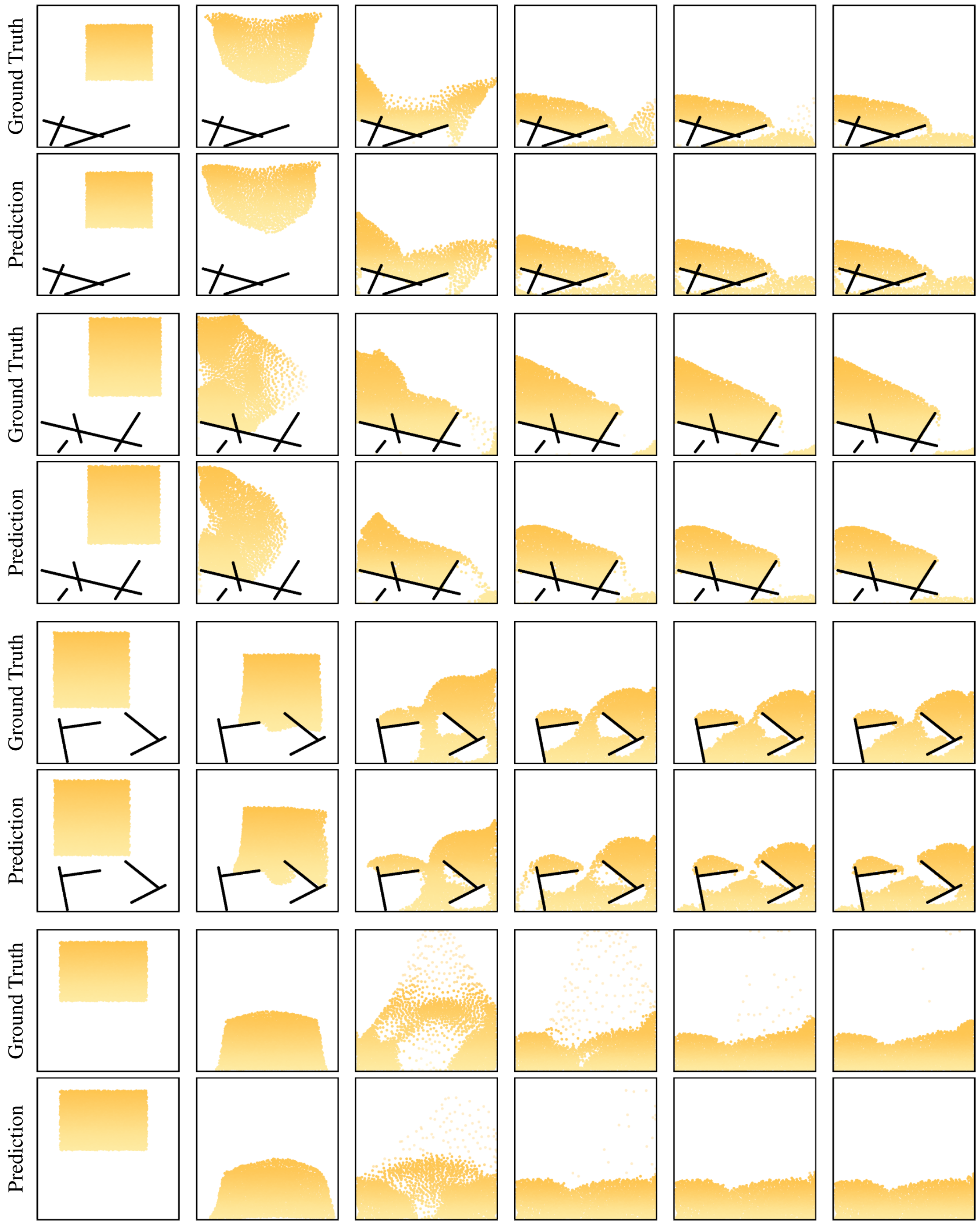

Figure 10. **Additional SANDRAMPS predicted trajectories.** Evenly spaced in time snapshots of predicted unrolled trajectories against ground truth. All trajectories are from the held-out test set.

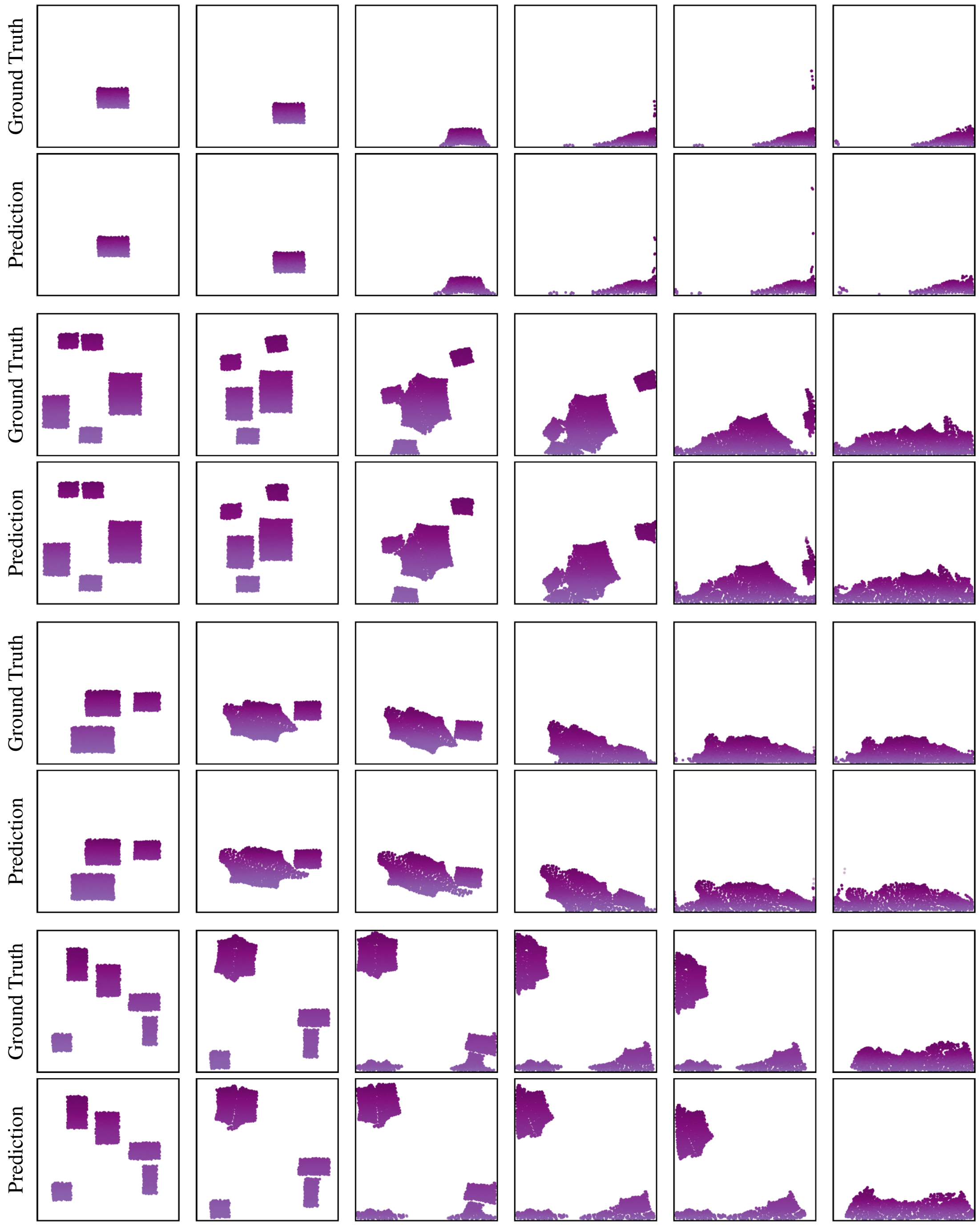

Figure 11. **Additional GOOP predicted trajectories.** Snapshots of predicted unrolled trajectories against ground truth. All trajectories are from the held-out test set. Due to GOOP quickly reaching equilibrium, more snapshots are taken in the first half of the trajectory.

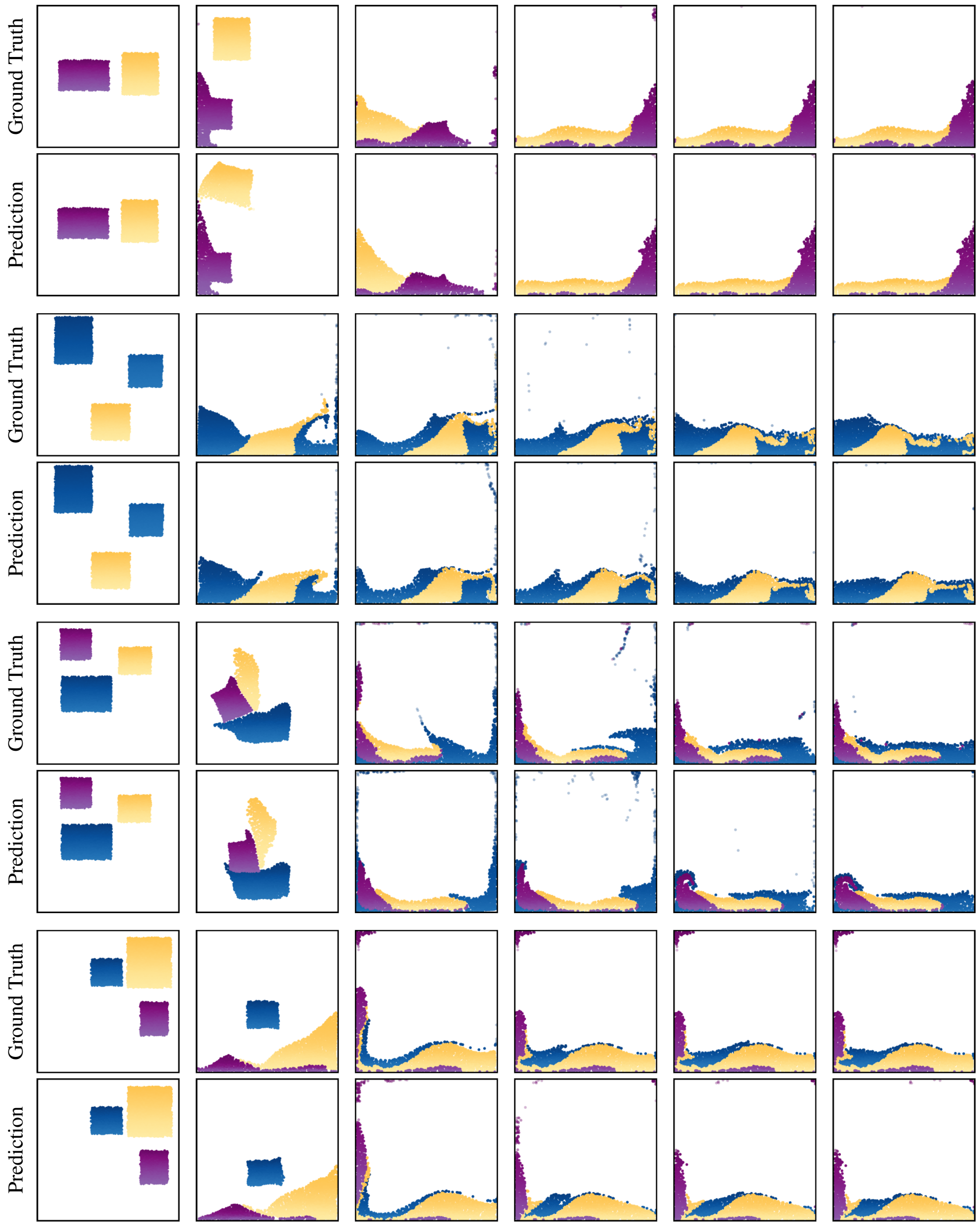

Figure 12. **Additional MULTIMATERIAL predicted trajectories.** Evenly spaced in time snapshots of predicted unrolled trajectories against ground truth. All trajectories are from the held-out test set. The first trajectory illustrates a rare failure where the shape of sand particles is not retained, even though all particles are supposed to maintain the same velocity while airborne, as they are thrown against the wall.

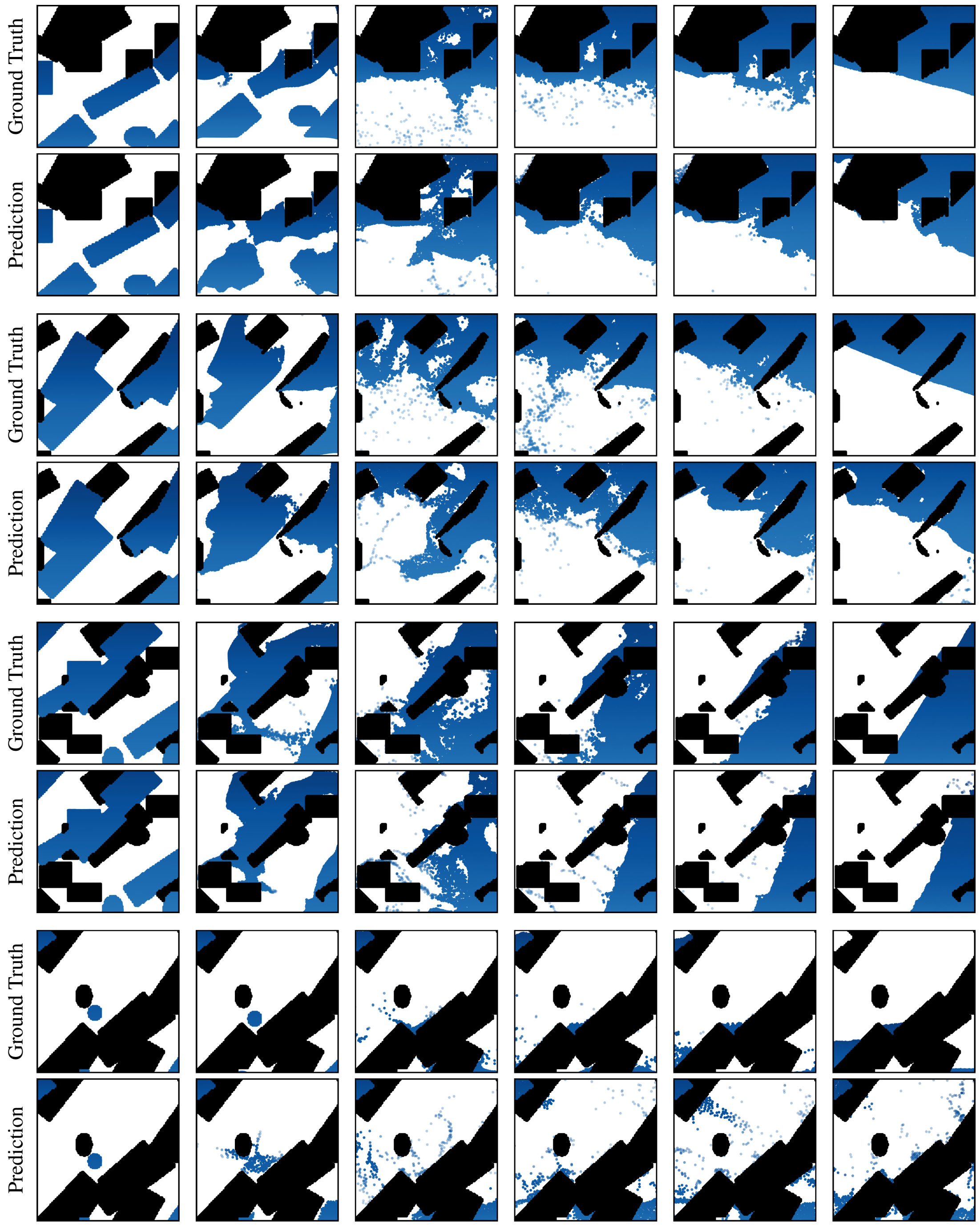

Figure 13. **Additional WBC-SPH predicted trajectories.** Snapshots of predicted trajectories against ground truth. All trajectories come from the held-out test set. To better show the differences of these longer sequences, we select the following timesteps not even in time: $t \in \{0, 100, 300, 500, 700, 3199\}$. In the last trajectory, NeuralMPM struggles to follow the gravity direction and breaks down over time.

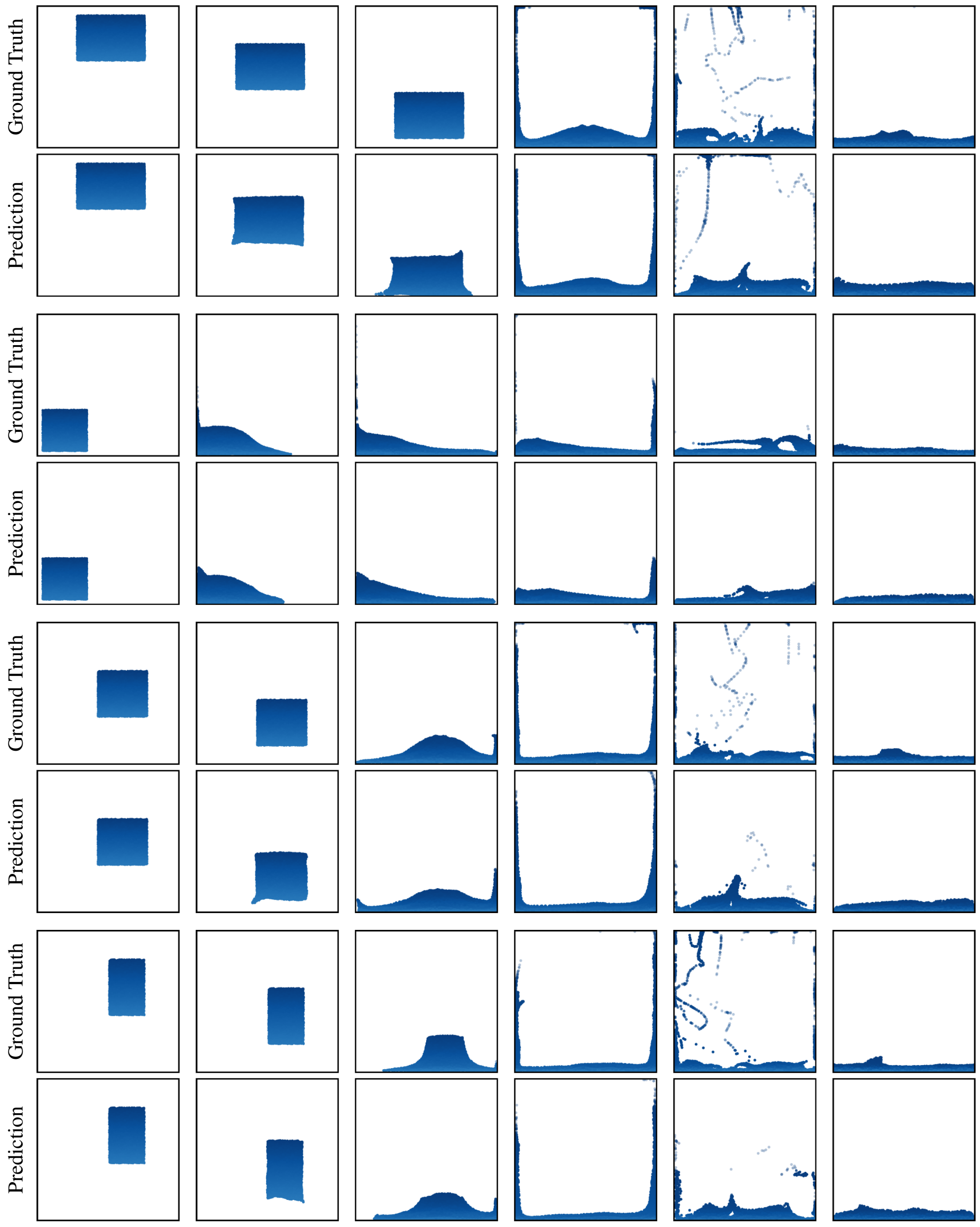

Figure 14. **Additional WaterDrop-XL predicted generalization trajectories.** Snapshots of predicted trajectories emulated using a model trained solely on WaterRamps, against ground truth. All trajectories come from the held-out test set of WaterDrop-XL. To better show the differences of these longer sequences, we select the following timesteps not even in time: $t \in \{0, 75, 125, 200, 400, 999\}$. We can observe that the generalizing model struggles to retain the shape of water while it's falling.