Hybrid Neural-MPM for Accelerated and Controllable Fluid Simulations

Anonymous Authors[1]

Abstract

We propose **Hybrid Neural-MPM**, a neural physics framework for accelerated, controllable fluid simulation. While traditional physics-based methods offer high accuracy, they are often computationally prohibitive and prone to latency. Recent machine-learning approaches have reduced these costs, yet most still struggle to support the demands of interactive applications. To bridge this gap, we introduce a novel hybrid method that integrates *numerical simulation, neural physics, and generative control.* Our neural physics model balances low-latency performance with physical fidelity by employing a fallback safeguard to classical numerical solvers. Furthermore, we design a diffusion-based controller, trained via a reverse modeling strategy, to generate external dynamic force fields for intuitive fluid manipulation. Our system demonstrates improved performance across diverse 2D/3D scenarios, material types, and obstacle interactions, achieving accelerated simulations (11 $\sim$ 29% latency reduced) while enabling fluid control guided by user-friendly freehand sketches. We include video results in the supplement, and commit to releasing models and data upon acceptance.

1. Introduction

Modeling fluid behavior is essential for advancing diverse engineering fields, including entertainment (Stam, 2023), urban planning (Blocken & Stathopoulos, 2013), fashion design (Volino et al., 2005), and virtual reality (VR) (Solmaz & Van Gerven, 2022). Moreover, controllability, aiming to instruct movements and shapes of fluids, is also a very important attribute for volumetric effects, character animations, and fluid-solid coupling (Raveendran et al., 2012). Realizing compelling and interactive physics simulations in real-time has been the long-standing objective for years in order to deliver transformative user experiences.

Traditional simulation methods, though powerful, often demand significant implementation efforts and computational costs (Bridson, 2015). Recent neural physics and machine learning approaches present a promising path forward by learning from data, delivering transformative changes for use cases such as fluid interactions and animations (Sanchez-Gonzalez et al., 2020). However, fidelity and latency in these neural-based methods are not well-balanced. Moreover, most methods only focus on the accuracy of non-controllable applications, and their computational complexity still remains generally high for real-time scenarios (Brandstetter et al., 2022).

Motivated by the above challenges, we ask two scientific questions:

> ***Q1**: Can neural physics accelerate fluid simulations?*
> ***Q2**: Can neural physics and generative methods control fluids?*

To bridge the gap between passive simulation and controllable creation, we propose a novel paradigm: neural physics for accelerated and controllable simulations (Fig. 1). We contribute a cohesive framework that addresses the acceleration and interactivity by **proactively marrying the strengths of numerical simulation (high fidelity), neural physics (low latency), and generative control (interactivity).** Within this framework, neural physics enables significantly low-latency fluid simulation, while numerical simulation serves as an automated fidelity safeguard and fallback solution when dynamics become increasingly complex. To ensure the system is compatible with user-friendly, sketch-based generative control, we introduce a diffusion-based controller that generates external force fields to assist manipulations. We summarize our contributions below:

1. We improve the **error-latency trade-off** of fluid simulation. First, to accelerate neural physics, we seek to build our graph neural network at low spatiotemporal resolution without substantial degradation in simulation accuracy (Sec. 3.1.1). Second, to preserve simulation fidelity and avoid error accumulation during unrolling,

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the AI4Physics Workshop at the International Conference on Machine Learning (AI4Physics@ICML 2026). Do not distribute.

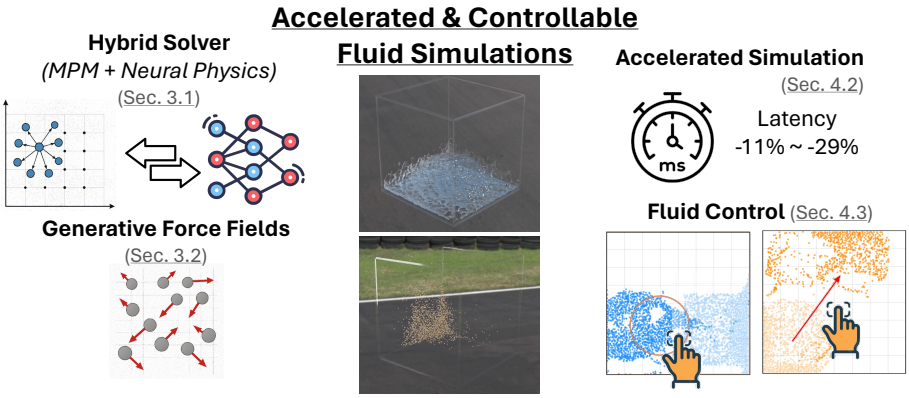


Figure 1. We target accelerated, controllable fluid simulations. Our hybrid solver integrates a numerical simulator and neural physics (Section 3.1), enabling accelerated simulation (Section 4.2). In addition, we generate external force fields (Section 3.2) to support users in controlling fluids interactively via freehand sketches (Section 4.3).

we make our neural physics hybrid with a safeguard condition and fallback mechanism to the classic MPM (Material Point Method) algorithm (Sec. 3.1.2).

2. We further aim to support **users' flexible freehand sketches** that specify desired trajectories or shapes of fluid particles to be controlled. To this end, our novel reverse simulation strategy enables the automated generation of realistic fluid control data (Sec. 3.2.2), which is used to train our diffusion-based generative controller (Sec. 3.2.3).

3. Across **diverse scenarios** (2D/3D, particle materials, rigid obstacles, see Table 2), our hybrid simulator can **accelerate simulations** ($11 \sim 29\%$ latency reduced) while maintaining low errors (Sec. 4.2), and can **control fluid particles to align with user sketches** (Sec. 4.3), paving the way for accelerated and controllable simulations.

2. Background

We introduce background and methodology, including acceleration and control, in Sec. 3. Our work builds on three key areas: fluid modeling and animation, fluid control, and controllable video generation. Due to page limits, we defer detailed discussion of related works to Appendix A.

2.1. Fluid Simulations with Material Point Method (MPM)

The Material Point Method (MPM) (Hu et al., 2019; 2020; 2021; Jiang et al., 2015) is a hybrid Eulerian-Lagrangian numerical technique for simulating complex interactions between solid and fluid materials, especially under large deformations and topological changes (snow, landslides, cloth, etc.). It extends the FLuids-Implicit-Particle (FLIP) (Brackbill & Ruppel, 1986) from Computational Fluid Dynamics (CFD) to solid mechanics by representing materials as a set of Lagrangian particles that carry mass, velocity ($\dot{\boldsymbol{p}}_{i,t}$), position ($\boldsymbol{p}_{i,t}$), and possible internal states. These particle quantities are first transferred to a background Eulerian grid using a particle-to-grid mapping (p2g). The equations of motion are then solved on this grid, after which updated values are mapped back to particles through grid-to-particle transfer (g2p). The particle positions ($\boldsymbol{p}$) are then advanced using the updated velocities ($\dot{\boldsymbol{p}}$), e.g., $\boldsymbol{p}_{i,t+1} = \boldsymbol{p}_{i,t} + \Delta t \cdot \dot{\boldsymbol{p}}_{i,t+1}$.

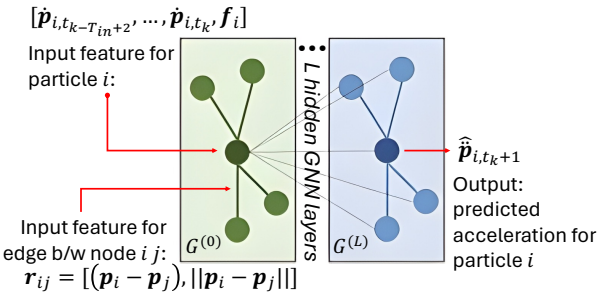


Figure 2. GNN as our neural physics simulator.

2.2. GNN-based Particle Simulations

We denote the state of particle i at time step t as $\boldsymbol{x}_{i,t}$ (position $\boldsymbol{p}$, velocity $\dot{\boldsymbol{p}}$, acceleration $\ddot{\boldsymbol{p}}$, etc.), and the state of N particles as $X_t = [\boldsymbol{x}_{1,t}, \ldots, \boldsymbol{x}_{N,t}]$. A *simulator* s maps T_{in} input states to causally consequent future states, and can iteratively compute $X_{t_{T_{\text{in}}+1}} = s(X_{t_1}, X_{t_2}, \cdots, X_{t_{T_{\text{in}}}})$ to simulate a rollout trajectory. Following (Sanchez-Gonzalez et al., 2020), our learnable simulator s_θ adopts a particle-based representation of the physical system, which can be viewed as message-passing via a graph neural network (GNN).

Input. Our neural physics simulator s_θ takes the input of particle i as: a sequence of 5 previous velocities (via finite differences from $T_{\text{in}} = 6$ previous locations), and features for materials (e.g., water, sand, rigid, boundary), i.e., $\boldsymbol{x}_{i,t_{k-T_{\text{in}}}:t_k} = [\dot{\boldsymbol{p}}_{i,t_{k-T_{\text{in}}+2}}, \ldots, \dot{\boldsymbol{p}}_{i,t_k}, \boldsymbol{f}_i]$ at time step t_k (Fig. 2).

GNN Design. We first build the initial graph $G^{(0)}$ by assigning a node to each particle and connecting particles as edges within a fixed "connectivity radius" R. The edge embeddings are learned from relative positional displacement and

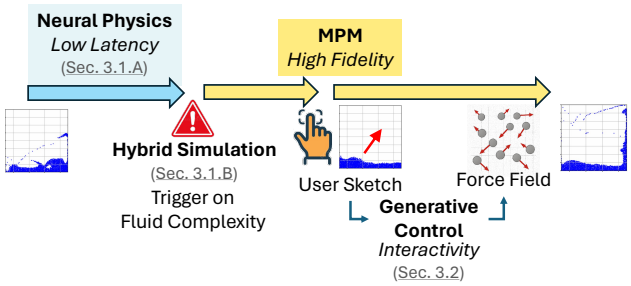


Figure 3. Method Overview. To achieve real-time simulations, we cut latency by learning neural physics at a coarse spatiotemporal resolution, while safeguarding fidelity by automatically falling back to an MPM solver when complex fluid phenomena arise (Sec. 3.1). For interactive control, we train a diffusion-based generative model that infers external force fields directly from user sketches (Sec. 3.2).

the magnitude $\boldsymbol{r}_{i,j} = [(\boldsymbol{p}_i - \boldsymbol{p}_j), \|\boldsymbol{p}_i - \boldsymbol{p}_j\|]$. Our neural physics consists of a stack of $L = 10$ GNN layers. The decoder predicts the per-particle acceleration, $\ddot{\boldsymbol{p}}_i$. The training loss is the particle-level $\mathrm{RMSE}_{\ddot{\boldsymbol{p}}} \equiv \frac{1}{N}\sum_{i=1}^{N} \frac{\|\hat{\ddot{\boldsymbol{p}}}_i - \ddot{\boldsymbol{p}}_i\|_2}{\|\ddot{\boldsymbol{p}}_i\|_2}$, where $\hat{\ddot{\boldsymbol{p}}}_i$ is the predicted acceleration from s_θ. The future position and velocity are updated using an Euler integrator. See Appendix B for further details.

3. Methods

Our method (Fig. 3) aims to accelerate fluid simulations (Sec. 3.1) with controllability (Sec. 3.2).

3.1. Hybrid Accelerated Fluid Simulation

In this work, we automate the hybridization between MPM and neural physics with a carefully studied trade-off (Fig. 5). Traditional numerical methods such as MPM provide high-fidelity simulations, but their computational cost is prohibitively high. In contrast, neural physics models can achieve significantly faster simulations by operating at low spatiotemporal resolution; however, this efficiency often comes at the cost of increased simulation errors.

This trade-off highlights our central motivation for building hybrid simulations. By primarily leveraging neural physics for efficient updates (Sec. 3.1.1) while incorporating a mechanism that falls back to MPM in challenging scenarios for simulation fidelity (Sec. 3.1.2), we can empirically ensure both efficiency and simulation quality. In this way, our simulator is designed as a hybrid system that effectively fuses the strengths of both approaches:

$$X_{t+1} = \begin{cases} \text{Neural Physics Update} & \text{if update is "good"} \\ \text{Fallback to MPM Update} & \text{otherwise.} \end{cases} \tag{1}$$

3.1.1. Learning Accelerated Neural Physics at Low Spatiotemporal Resolution.

We first aim to accelerate efficient updates by neural physics, and we choose to train our model at low spatiotemporal resolution. As shown in Fig. 4, we consider learning the neural physics on simulations with both a downsampled number of particles (ratio $r_p \in (0, 1)$) and also with a larger time step (i.e. coarser temporal discretization rate $r_t \in \mathbb{N}, r_t > 1$).

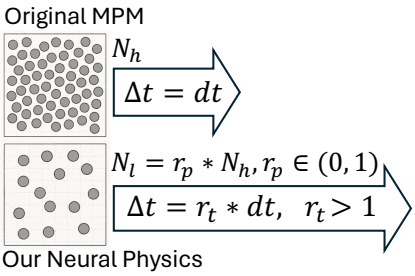


Figure 4. Our neural physics accelerates simulations by learning and inferring at low spatial (N_l num. particles) and temporal (Δt time steps) resolutions, with downsampling ratios as r_p, r_t.

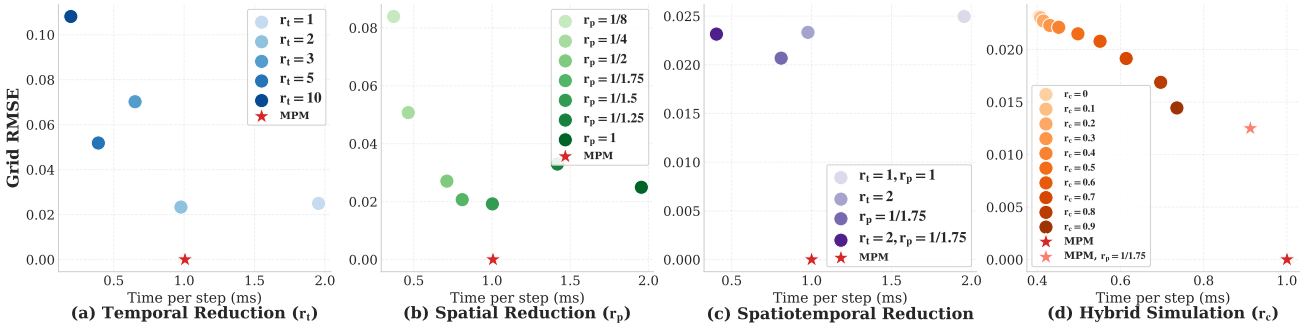
However, a key pitfall is that once the number of particles is downsampled (N_h particles are merged via clustering into N_l, see Appendix C), we will lose the particle-wise correspondence, i.e., $\hat{\ddot{\boldsymbol{p}}}_i$ ($i \in [1, N_l]$) and $\ddot{\boldsymbol{p}}_j$ ($j \in [1, N_h]$) cannot align in the particle-level $\mathrm{RMSE}_{\ddot{\boldsymbol{p}}}$ (Sec. 2.2). As a result, $\mathrm{RMSE}_{\ddot{\boldsymbol{p}}}$ can no longer quantify the simulation's fidelity to the ground truth of the original spatial resolution. Inspired by (Deng et al., 2023; Huang et al., 2021)[1], to mitigate this issue, we use a normalized grid-level $\mathrm{RMSE}_{\tilde{\boldsymbol{m}}} \equiv \frac{1}{N}\sum_{i=1}^{N} \frac{\|\hat{\tilde{\boldsymbol{m}}}_i - \tilde{\boldsymbol{m}}_i\|_2}{\|\tilde{\boldsymbol{m}}_i\|_2}$ as the evaluation metric, which essentially quantifies the mass distribution. $\tilde{\boldsymbol{m}}$ is the normalized grid mass ($\tilde{\boldsymbol{m}}_i = \frac{\boldsymbol{m}_i}{\sum_{i=1}^{N} \boldsymbol{m}_i}$) converted from particles to the grid via p2g, and $\hat{\tilde{\boldsymbol{m}}}$ is the prediction by s_θ. $\tilde{\boldsymbol{m}}_i$ and $\hat{\tilde{\boldsymbol{m}}}$ share the same grid size but can represent mass distributions from different resolutions (number of particles). A more detailed discussion regarding $\mathrm{RMSE}_{\tilde{\boldsymbol{m}}}$ can be found in Appendix E. During training, we continue to optimize the surrogate loss $\mathrm{RMSE}_{\ddot{\boldsymbol{p}}}$ at the low spatial resolution, thereby avoiding additional p2g operations.

In Fig. 5 (a-c), we can see that by tuning spatiotemporal downsampling ratios r_p, r_t, we can improve the trade-off between simulation errors and latency. Based on this ablation study, we will choose $r_p = 1/1.75$ and $r_t = 2$. With this configuration, on Water 2D, we can reduce the latency of the original neural physics ($r_p = r_t = 1$) by over 78.8% (from 1.954ms to 0.4048ms).

3.1.2. Hybrid Simulator with Safeguard

We now discuss how to actively interact our neural physics with MPM and fuse the strengths of both approaches.

[1] See Sec. 3.1 (page 4), paragraph "Goal and Reward" in (Huang et al., 2021); Sec. 6.1 "Validation of SSNF" in (Deng et al., 2023)



(a) Temporal Reduction (r_t) **(b) Spatial Reduction (r_p)** **(c) Spatiotemporal Reduction** **(d) Hybrid Simulation (r_c)**

Figure 5. Ablation studies of the trade-off between grid-level $\mathrm{RMSE}_{\tilde{\boldsymbol{m}}}$ vs. simulation latency. Left to right: temporal reduction r_t (train neural physics with larger time step Δt), spatial reduction r_p (train neural physics with reduced particles N_l), spatiotemporal reduction (combine $r_t = 2$ and $r_p = 1/1.75$), and hybrid with MPM (at $r_p = 1/1.75$) with different thresholds r_c. Scenario: Water 2D.

Fluid Complexity Measures. Intuitively, when the particle system evolves smoothly, the neural model generalizes well. In contrast, highly chaotic or abrupt behaviors, such as splashes, collisions, or multiphase interactions, often correspond to out-of-distribution (OOD) regimes, where learned models are more prone to error, suggesting that we should fall back to MPM.

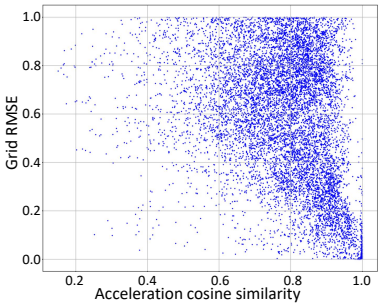


Figure 6. Strong correlation magnitude between "cosine similarity of particle accelerations over frames" vs. "simulation errors of neural physics" (Water 2D). Spearman correlation: -0.3902.

We thus trigger the fallback condition based on the complexity of the current fluid dynamics being simulated by neural physics. Our choice of fallback trigger is motivated by two key considerations. *First*, this trigger should faithfully indicate the **fluid complexity**. *Second*, the safeguard should be **computationally efficient**, since we need to densely monitor them during the simulation of neural physics.

Specifically, we consider the cosine similarity of per-particle acceleration over a window of history (window size as $\delta t = 10$ steps by default) in Eqn. 2.

In contrast, prior works commonly quantify fluid complexity or incompressibility by monitoring the divergence of particles' velocity (Gao et al., 2025). However, calculating velocity divergence relies on finite-difference approximations or expensive grid interpolations, which significantly increase simulation latency. Similarly, adopting a learned uncertainty estimation would introduce prohibitive computational overhead.

We show the strong correlation magnitude between this cosine similarity and the neural physics simulation error in Fig. 6, which indicates that whenever particles' accelerations start diverging, we should fall back to MPM.

Triggering MPM by Fluid Complexity. With our fluid complexity metric, we choose to trigger the fallback mechanism and switch to MPM by a threshold r_c, and $\rho_t = \frac{1}{N}\sum_{i=1}^{N} \cos(\ddot{\boldsymbol{p}}_{i,t-2\delta t:t-\delta t}, \ddot{\boldsymbol{p}}_{i,t-\delta t:t})$:

$$X_{t+1} = \begin{cases} \text{Neural Physics Update} & \text{if } \rho_t > r_c \\ \text{MPM Update} & \text{otherwise.} \end{cases} \tag{2}$$

Our fallback trigger is efficient: it only relies on the cosine similarity of per-particle acceleration vectors. It acts as an early-warning signal that strongly correlated with future rollout errors (Fig. 6), but requires no neighbor search or grid operations. In Table 1, when increasing our threshold r_c (i.e. MPM will be more frequently triggered), the simulation fidelity will be corrected (improved $\mathrm{RMSE}_{\tilde{\boldsymbol{m}}}$), with increased latency due to computations of MPM. Thus, we need to choose a threshold r_c such that we can improve our trade-off between $\mathrm{RMSE}_{\tilde{\boldsymbol{m}}}$ and latency. In Fig. 5 (d), we tune this threshold, and choose $r_c = 0.8$ to balance the improvements over $\mathrm{RMSE}_{\tilde{\boldsymbol{m}}}$ and latency.

We finalize our hybrid solver using this threshold. In Fig. 7, we demonstrate trajectories simulated by the original neural physics and our hybrid solver (same initial condition and number of steps T). Although the original neural physics ($r_p = r_t = 1$) shows lower rollout errors in the early stage (black curve, due to simulation at high resolution), it quickly accumulates long-term errors. In contrast, our fluid complexity (Eqn. 2) reliably predicts when neural rollouts become inaccurate, **triggers the fallback to MPM (yellow areas), suppress the error, and finish the simulation**

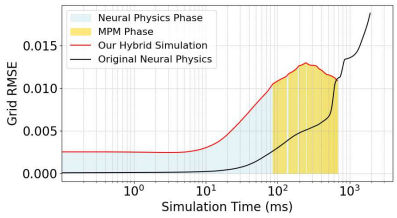


Figure 7. Simulation error trajectories (Water 2D). Simulating the same steps ($T = 1000$), our hybrid solver (red): 1) takes significantly less time (676.4ms) than the original neural physics (1931.1ms, black); 2) reduces the error (grid $\mathrm{RMSE}_{\tilde{\boldsymbol{m}}}$, 0.0109 vs. 0.0188) with our fallback mechanism (Eqn. 2, yellow), as the MPM solver corrects the drift accumulated by the neural solver.

Table 1. Grid $\mathrm{RMSE}_{\tilde{m}}$ vs. time per step with hybrid simulations triggered by different thresholds (Water 2D).

Threshold r_c	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Grid $\mathrm{RMSE}_{\tilde{m}}$	0.0232	0.0230	0.0227	0.0223	0.0221	0.0215	0.0208	0.0192	0.0169	0.0144
Time per step (ms)	0.4048	0.4081	0.4147	0.4301	0.4516	0.4977	0.5509	0.6137	0.6966	0.7356

much faster. Thus, our hybrid solver improves both rollout $\mathrm{RMSE}_{\tilde{m}}$ and latency.

3.2. Fluid Control: Generating Dynamic Force Fields

3.2.1. Motivations

Fluid control is essential in computer graphics, where liquid animations convey expressive, story-driven scenes and key visual ideas (Yan et al., 2020). Manual fluid control produces unnatural effects and forces artists into slow, trial-and-error workflows (Pan et al., 2013), underscoring the need for intuitive tools that shape visuals without complex physics. Yet achieving desired fluid appearances remains difficult: fluid dynamics are intrinsically chaotic, setup and tuning are tedious, and recording real fluid motion is expensive and hard to customize. In this paper, we consider the following use case: during simulation, a user draws a simple sketch as a control signal, and the fluid particles should follow it (Fig. 8 bottom panel), with the *artistic* manipulation of particles automatically designed by our system.

Why Generative Fluid Control via Diffusion Models? Diffusion-based generative models are a natural choice for fluid control because they combine strong conditional generation with temporal coherence and spatial flexibility. Recent works show that diffusion can precisely steer motions when conditioned on trajectories or region cues in video (Wang et al., 2024b; Yang et al., 2024; Zhang et al., 2024) and predict spatiotemporal physics dynamics with high fidelity (Dong et al., 2025; Hu et al., 2024; Kohl et al., 2023; Zhou et al., 2024). This suggests that **diffusion models can map sketch-like intent to physically meaningful velocity or force fields over time.** In fact, recent works have already learned to synthesize fluid velocities with deep generators (Chu et al., 2021; Yan et al., 2020). In contrast, traditional control pipelines often rely on classic offline optimization methods (McNamara et al., 2004; Pan et al., 2013; Schoentgen et al., 2020), which require bespoke assets and iterative heuristics that are computationally prohibitive for real-time interaction and broad generalization. Our work targets fluid control and interpretation of user-friendly, free-hand sketches, which represent high-level "intent" rather than precise target states, and instantaneously generate a physically plausible, dynamic force field.

A core challenge in sketch-based fluid control is the lack of paired training data: creating massive datasets that pair an abstract "user sketch" with the corresponding "optimal spatiotemporal force field" is analytically intractable! We bypass this data bottleneck by introducing a novel reverse-simulation strategy. By programmatically generating training data from passive forward trajectories, we formulate the abstract "intent-to-physics" problem as a solvable conditional generation task.

3.2.2. Data Generation via Reversed Simulation.

The key to training a generative model for fluid control is to automatically collect training data. Specifically, we have two highly nontrivial sub-tasks:

1) Design a large number of diverse scenarios of fluid particles with artistic control effects (i.e. fluid particles move along a desired direction or fill a pre-defined shape, in an organized instead of a chaotic manner); 2) Solve a spatiotemporal external force field that will be applied to the particles, such that the artistic control effect can be fulfilled driven by the composition of gravity, particle interactions, and the proposed force field.

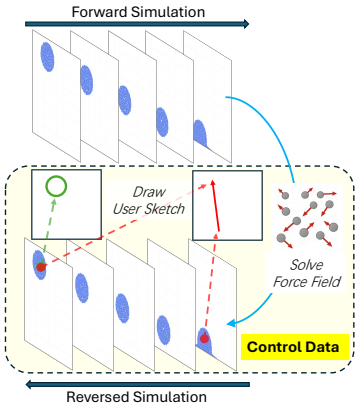


Figure 8. We prepare training data for generative control via solving external force fields that can reverse a forward simulation, and also user sketches (arrow, ellipse) that depict movements or target shapes of particles (details in Appendix C).

We address these challenges with a reverse simulation strategy. The core idea is to **solve the required force fields** that can **reverse the fluid dynamics** of artistic effects. We have the following steps:

1) Forward Simulation. We randomly simulate a trajectory of fluid dynamics $\boldsymbol{X} = (X_1, X_2, \cdots, X_{T_{\text{ctr}}})$, with different initial conditions (positions or velocities of particles).

2) Reversed Simulation. We iteratively solve the required acceleration[2] $\boldsymbol{a}_t$ that can restore *positions* of each fluid particle reversely, from $X_{T_{\text{ctr}}}$ to X_1:

$$(\boldsymbol{a}_t + \boldsymbol{g})\Delta t = \frac{\boldsymbol{p}_{t-1} - \boldsymbol{p}_t}{\Delta t} - \dot{\boldsymbol{p}}_t, \boldsymbol{a}_t = \frac{(\boldsymbol{p}_{t-1} - \boldsymbol{p}_t) - \dot{\boldsymbol{p}}_t \Delta t}{(\Delta t)^2} - \boldsymbol{g}. \tag{3}$$

[2]Equivalently, the force field if all particles have the same constant mass

Equation 3 stems from the discretized second-order difference equation of motion, and provides a physically interpretable approximation of the acceleration needed to move from $\boldsymbol{p}_t$ to $\boldsymbol{p}_{t-1}$, subtracting out the known gravitational acceleration $\boldsymbol{g}$. Our target force field can be non-linear (see Fig. 13 in our Appendix for a simulation example). Moreover, in Fig. 16, the cases where fluid shapes change during control indicate that our diffusion model is trained to predict these non-linear force fields.

3) Generation of Control Sketches. Finally, based on $\boldsymbol{X}$, we generate the user's sketch that depicts the general movements of particles. We support both directional arrows for movement guidance and one-stroke freehand oval shapes to indicate target regions, as shown in Fig. 8. See our Appendix C for details of implementing freehand arrows and oval shapes. Note that in 3D scenarios, we use the arrow width to indicate depth (Pan et al., 2013).

For simplicity, we will by default control the fluid particles for 100 MPM steps ($T_{\text{ctr}} = 100$). That means all our control trajectories will have 100 steps. While it is possible to employ dynamic neural architectures (Yu et al., 2018) to adaptively adjust the number of MPM steps for this control based on the control complexity, we leave it as future work.

3.2.3. Diffusion-based Fluid ControlNet.

Conditioned on previous particle trajectories and a user's sketch, our diffusion-based Fluid ControlNet will predict a dynamic force field ($\boldsymbol{a}$ in Equation 3). We control fluid particles by applying this predicted force field atop MPM, since MPM can explicitly take external forces and update the velocity advection, intentionally bypassing the neural physics solver. This separation ensures both physical fidelity and system robustness. Our neural physics simulator is trained to approximate passive physical dynamics; it has never observed the massive, artistically-driven external forces generated by the user's sketch. Applying these unphysical control forces to the GNN would represent a severe OOD shift, triggering immediate instability and a spike in fluid complexity that would force an MPM fallback regardless. Conversely, the MPM solver acts as a reliable, high-fidelity physical sink that robustly integrates external force fields while conserving momentum and adhering to physical laws.

Our design follows standard practice in controllable generation (Wang et al., 2024b; Zhang et al., 2023). We accept user sketches and extract features for spatially grounded conditioning, enabling free-form arrows, regions, and shapes.

We show our architecture design in Fig. 9. Specifically: ① Our diffusion-based Fluid ControlNet shares the same backbone and input particle features as our neural physics (Sec. 2.2). ② Parallel to the backbone, we extract the embeddings of the user's sketch input using a convolutional neural network (CNN) and concatenate them with the diffusion timestep embeddings to guide the generation process. We also embed the current control time step into a latent space and integrate it into the initial noise. ③ The training target will be the ground truth force fields we simulate in Sec. 3.2.2. ④ During training, noise is added to the ground-truth force (or acceleration) field, which is then used as input. The diffusion model's inputs are: (1) the previous 6 positions and velocities, (2) the noisy force field (Gaussian noise during inference), and (3) the embedding of the control timestep. ⑤ The output of our Fluid ControlNet is an external force field that will be applied to particles during MPM simulations. ⑥ Along the MPM simulation, our Fluid ControlNet will unroll the subsequent temporal force fields.

See Appendix C for details of the architecture of our Fluid ControlNet.

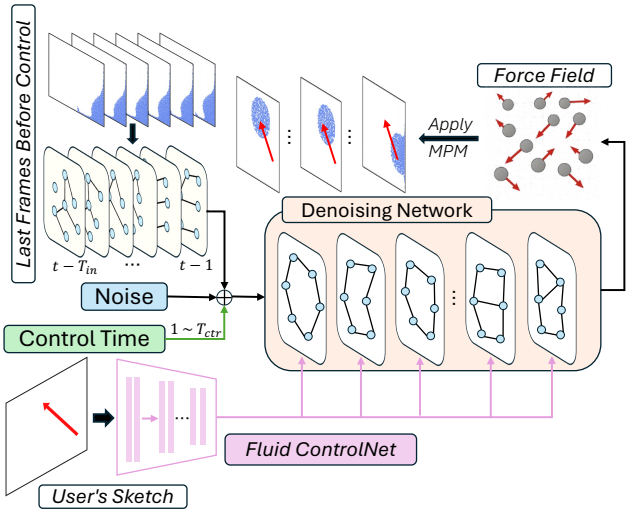


Figure 9. Architecture design of our Fluid ControlNet. The diffusion-based model takes past particle states, initial noise, and control time as inputs. Conditioned on features extracted from a user's sketch, the denoising network predicts an external force field that is subsequently applied within the MPM simulation to guide fluid dynamics.

4. Experiments

For different simulation scenarios, we train separate neural physics models and Fluid ControlNet. This design choice follows prior work (Sanchez-Gonzalez et al., 2020) where a dedicated model

Table 2. Datasets. N_h: Max number of particles at the original spatial resolution. T: total time steps. M: number of simulation trajectories.

Domain	N_h	T	M
Water (2D)	4k	1k	1k
WaterRamps (2D)	3.3k	600	1k
Sand (2D)	4k	320	1k
SandRamps (2D)	3.3k	400	1k
Water (3D)	4k	800	1k
Sand (3D)	4k	350	1k
Water-Sand (2D)	4k	500	1k

is also trained per scene to better capture scene-specific dynamics.

4.1. Settings

Physical Domains and Simulations. To build our hybrid simulator, we prepare our own ground truth simulations[3] with the Taichi package (Hu et al., 2019; 2020; 2021) on GPUs, with settings closely aligned with (Sanchez-Gonzalez et al., 2020). Different simulation scenarios are summarized in Table 2. We include diverse initial conditions (position, velocity) and numbers of particles. We fix our grid size as 128×128 for 2D and $64 \times 64 \times 64$ for 3D, and use time step $dt = 2.5$ms in simulations. See Appendix C.2 for details of physical simulation parameters.

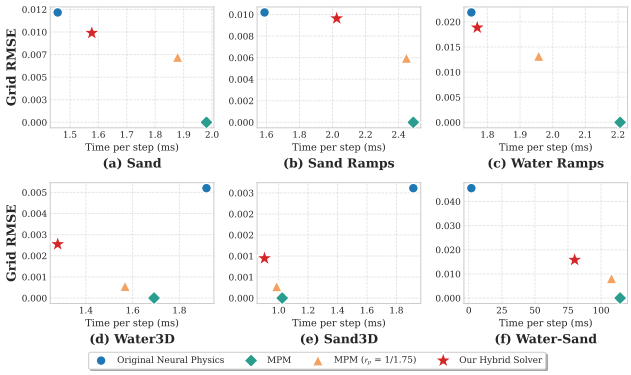


Figure 10. Trade-off between simulation error (grid $\mathrm{RMSE}_{\tilde{m}}$) and latency, comparing different methods. (a) Sand (2D); (b) SandRamps (2D); (c) WaterRamps (2D); (d) Water (3D); (e) Sand (3D); (f) Water-Sand (2D). Overall, our hybrid solver achieves a balanced trade-off between RMSE and simulation latency, outperforming both neural physics and MPM.

Evaluation. To report quantitative results, we evaluated our models by computing **rollout** metrics on held-out test trajectories, drawn from the same distribution of initial conditions used for training. As discussed in Sec. 3.1.1, we use grid-level $\mathrm{RMSE}_{\tilde{m}}$ to compare predictions at lower spatial resolution with the original ground truth.

4.2. Fluid Simulation Acceleration

Our hybrid simulator can consistently achieve accelerated fluid simulations with preserved simulation fidelity across both 2D and 3D cases. We show the trade-off between simulation error and latency in Fig. 10, where we compare our hybrid solver with the original neural physics ($r_p = r_t = 1$) (Sanchez-Gonzalez et al., 2020), MPM (Hu et al., 2019; 2020; 2021), and another MPM that also simulates at low spatial resolution ($r_p = 1/1.75$). On 2D scenarios, our hybrid solver consistently balances the neural physics and MPM, achieving both reduced simulation latency and preserved simulation errors. For example, on multiple materials (Water-Sand 2D), our hybrid solver can accelerate MPM from 0.114s per frame to 0.08s, with a 29.8% reduction. On 3D, the neural physics at $r_p = r_t = 1$ is extremely slow, whereas our hybrid solver improves both latency and errors. For example, on Sand 3D, we reduce the latency of MPM by 11.8%, from 1.02ms to 0.90ms.

We also compare with neural physics methods on WATER2D and SAND2D (Table 3). On WATER2D, our hybrid solver achieves the lowest RMSE of 0.0186, significantly outperforming GIOROM (0.0804), NeuralMPM (0.0829), and GNS (0.0263). On SAND2D, our method again achieves the lowest RMSE of 0.0116. Notably, our training cost in GPU hours remains on par with the GNS baseline, indicating that improved accuracy comes without additional overhead. Note that this comparison is limited to passive simulation, as methods like GIOROM and NeuralMPM do not support user-controllable actions, a key feature of our framework. For comparisons with additional methods, see Appendix F.

Table 3. Comparison of grid $\mathrm{RMSE}_{\tilde{m}}$ and GPU hours on Water2D and Sand2D.

Dataset	Metric	GNS	GIOROM	NeuralMPM	Our Hybrid Solver
Water2D	$\mathrm{RMSE}_{\tilde{m}}$	0.0263	0.0804	0.0829	**0.0186**
	GPU Hours	17.27h	28.37h	17.72h	**17.27h**
Sand2D	$\mathrm{RMSE}_{\tilde{m}}$	0.0125	0.2175	0.0785	**0.0116**
	GPU Hours	17.94h	20.43h	**15.62h**	17.94h

4.3. Generative Fluid Control

We show visualizations of our generative fluid control in Fig. 11. We compare with a baseline[4], where particles are controlled with a spatiotemporal constant force field, with the force magnitude and orientation solved by moving particles from $X_{T_{\text{ctr}}}$ to X_1.

User Study. We conducted a user study where 25 participants evaluated 17 conditioned samples per method on two criteria: Control Fidelity (sketch alignment) and Physical Realism (natural fluid dynamics); see Appendix D for details. As shown in Table 4, our method consistently outperformed the baseline. Participants strongly preferred our method for Physical Realism (72.0%), as the baseline's

[3] 1) Prior works (Kumar & Vantassel, 2022; Sanchez-Gonzalez et al., 2020) released only simulation trajectories, not the original simulators. Without the original simulator, exact trajectory reproduction and apple-to-apple comparisons are inherently impossible; 2) Prior works evaluate only within the training distribution, and none establish a standardized protocol for systematic distribution shift testing (Li et al., 2018; Sanchez-Gonzalez et al., 2020; Toshev et al., 2024); 3) The scale of our number of particles is comparable to 1k–4.3k particles used in prior works (Sanchez-Gonzalez et al., 2020).

[4] Prior fluid control methods (Chu et al., 2021; Pan et al., 2013; Schoentgen et al., 2020; Yan et al., 2020) either rely on costly optimization requiring precise keyframes rather than intuitive sketches, or use sketch-based interfaces with significant complexity limitations; neither standardized initial conditions nor their simulators were publicly released.

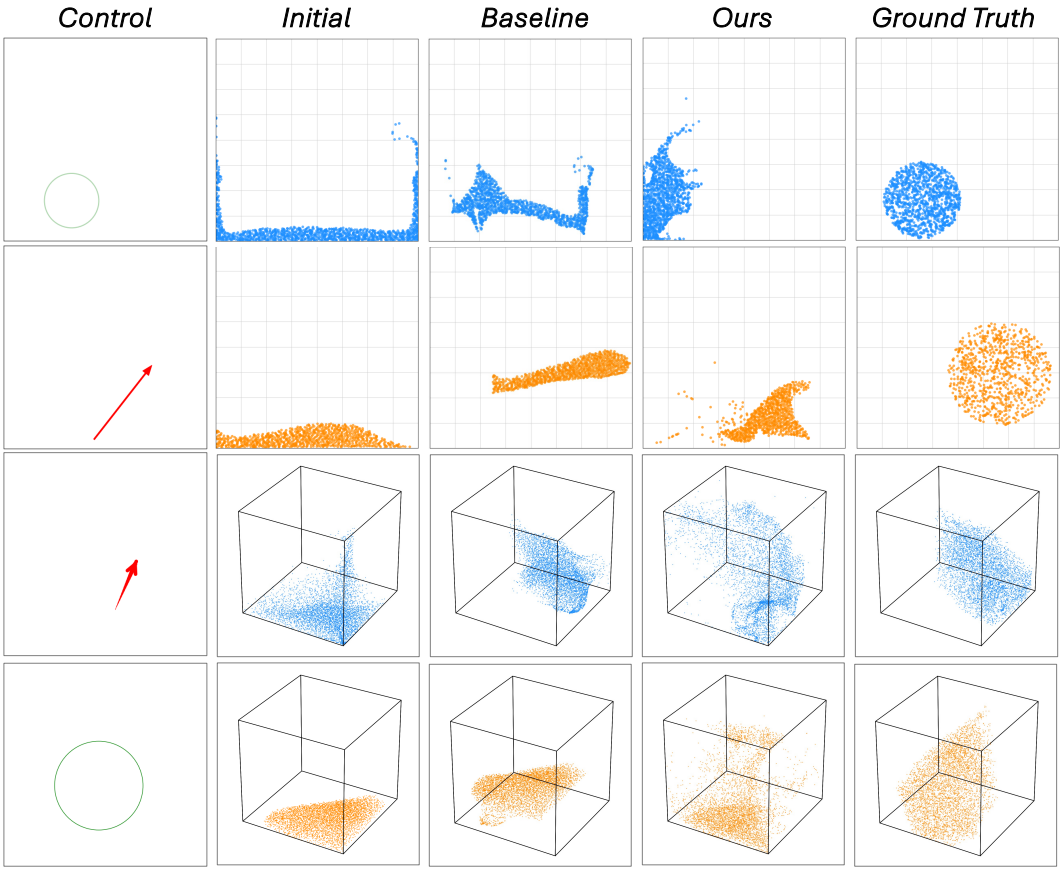


Figure 11. Visualization of fluid control. Top to bottom: Water (2D), Sand (2D), Water (3D), Sand (3D).

constant force fields produce stiff, unnatural motion. This advantage in dynamic, natural-looking physics drove an overall preference of 63.1% for our diffusion-based Fluid ControlNet.

Table 4. User Study Results. Participant preference rates comparing our method against the baseline across evaluated criteria. The remaining percentages represent cases where participants indicated a tie.

Method	Control Fidelity	Physical Realism	Overall
Baseline	43.1%	24.3%	33.7%
Ours	**54.2%**	**72.0%**	**63.1%**

We also provide quantitative results; see Appendix F.3 for more details. In sum, we can see that our diffusion-based Fluid ControlNet can move particles to better align with the user sketches.

4.4. Complete Results: Hybrid Simulation with Fluid Control

Finally, we present the result from our complete pipeline in Fig. 12. Particles are first simulated by our hybrid solver, where we start with the neural physics (at low spatiotemporal resolution), and switch to MPM once the fluid complex is high. Then, a user draws a sketch to control, and our diffusion-based Fluid ControlNet takes both this sketch and recent particle states as inputs, and generates external force fields to control particles.

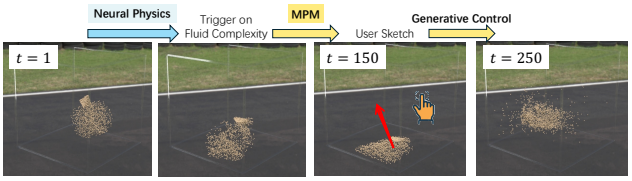


Figure 12. Complete results: hybrid simulation + fluid control. We start the simulation with our neural physics, which is then triggered to MPM. At $t = 150$, a user presents the control sketch.

5. Conclusion

We introduced a hybrid neural physics framework combining learned graph-based neural simulators with a classical MPM fallback, achieving robust, low-latency fluid dynamics without sacrificing accuracy. A diffusion-based generative controller trained via reverse modeling enables intuitive sketch-driven fluid control. Experiments across 2D and 3D domains confirm that our approach accelerates simulation while producing controllable, physically plausible results, advancing artist-friendly fluid simulation for graphics, design, and virtual environments. Main limitations: (1) the control step T_{ctl} is fixed at 100 rather than adaptive to scenario difficulty; and (2) low-resolution neural physics inference introduces errors. Potential remedies, an unrolling-aware diffusion controller and a super-resolution correction model, are left for future work.

References

Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Blocken, B. and Stathopoulos, T. Cfd simulation of pedestrian-level wind conditions around buildings: Past achievements and prospects. *Journal of Wind Engineering and Industrial Aerodynamics*, 121:138–145, 2013.

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

Brandstetter, J., Worrall, D., and Welling, M. Message passing neural pde solvers. *arXiv preprint arXiv:2202.03376*, 2022.

Bridson, R. *Fluid simulation for computer graphics*. AK Peters/CRC Press, 2015.

Chen, J., Fang, H.-r., and Saad, Y. Fast approximate kNN graph construction for high dimensional data via recursive lanczos bisection. *Journal of Machine Learning Research*, 10(Sep):1989–2012, 2009.

Chen, Y., Levin, D., and Langlois, T. Fluid control with laplacian eigenfunctions. In *ACM SIGGRAPH 2024 Conference Papers*, pp. 1–11, 2024.

Chu, M., Thuerey, N., Seidel, H.-P., Theobalt, C., and Zayer, R. Learning meaningful controls for fluids. *ACM Transactions on Graphics (TOG)*, 40(4):1–13, 2021.

Deng, Y., Yu, H.-X., Zhang, D., Wu, J., and Zhu, B. Fluid simulation on neural flow maps. *ACM Transactions on Graphics (TOG)*, 42(6):1–21, 2023.

Dong, W., Moses, C., and Li, K. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th International Conference on World Wide Web*, pp. 577–586, 2011.

Dong, X., Chen, C., and Wu, J.-L. Data-driven stochastic closure modeling via conditional diffusion model and neural operator. *Journal of Computational Physics*, 534: 114005, 2025.

Gao, Y., Yu, H.-X., Zhu, B., and Wu, J. Fluidnexus: 3d fluid reconstruction and prediction from a single video. *arXiv preprint arXiv:2503.04720*, 2025.

Guan, S., Deng, H., Wang, Y., and Yang, X. Neurofluid: Fluid dynamics grounding with particle-driven neural radiance fields. In *International conference on machine learning*, pp. 7919–7929. PMLR, 2022.

Guo, Y., Yang, C., Rao, A., Agrawala, M., Lin, D., and Dai, B. Sparsectrl: Adding sparse controls to text-to-video diffusion models. In *European Conference on Computer Vision*, pp. 330–348. Springer, 2024.

Han, J., Huang, W., Ma, H., Li, J., Tenenbaum, J., and Gan, C. Learning physical dynamics with subequivariant graph neural networks. *Advances in Neural Information Processing Systems*, 35:26256–26268, 2022.

He, H., Xu, Y., Guo, Y., Wetzstein, G., Dai, B., Li, H., and Yang, C. Cameractrl: Enabling camera control for text-to-video generation. *arXiv preprint arXiv:2404.02101*, 2024.

Hu, P., Wang, R., Zheng, X., Zhang, T., Feng, H., Feng, R., Wei, L., Wang, Y., Ma, Z.-M., and Wu, T. Wavelet diffusion neural operator. *arXiv preprint arXiv:2412.04833*, 2024.

Hu, Y., Li, T.-M., Anderson, L., Ragan-Kelley, J., and Durand, F. Taichi: a language for high-performance computation on spatially sparse data structures. *ACM Transactions on Graphics (TOG)*, 38(6):201, 2019.

Hu, Y., Anderson, L., Li, T.-M., Sun, Q., Carr, N., Ragan-Kelley, J., and Durand, F. Difftaichi: Differentiable programming for physical simulation. *ICLR*, 2020.

Hu, Y., Liu, J., Yang, X., Xu, M., Kuang, Y., Xu, W., Dai, Q., Freeman, W. T., and Durand, F. Quantaichi: A compiler for quantized simulations. *ACM Transactions on Graphics (TOG)*, 40(4), 2021.

Huang, Z., Hu, Y., Du, T., Zhou, S., Su, H., Tenenbaum, J. B., and Gan, C. Plasticinelab: A soft-body manipulation benchmark with differentiable physics. *arXiv preprint arXiv:2104.03311*, 2021.

Jain, P., Qu, Z., Chen, P. Y., and Stein, O. Neural monte carlo fluid simulation. In *ACM SIGGRAPH 2024 Conference Papers*, pp. 1–11, 2024.

Jiang, C., Schroeder, C., Selle, A., Teran, J., and Stomakhin, A. The affine particle-in-cell method. *ACM Transactions on Graphics (TOG)*, 34(4):1–10, 2015.

Jin, H. and Xie, H. Sketch-based fluid video generation using motion-guided diffusion models in still landscape images. In *Proceedings of the Special Interest Group on Computer Graphics and Interactive Techniques Conference Posters*, pp. 1–2, 2025.

Kohl, G., Chen, L.-W., and Thuerey, N. Benchmarking autoregressive conditional diffusion models for turbulent flow simulation. *arXiv preprint arXiv:2309.01745*, 2023.

Kumar, K. and Choi, Y. Accelerating particle and fluid simulations with differentiable graph networks for solving forward and inverse problems. In *Proceedings of the SC'23 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis*, pp. 60–65, 2023.

Kumar, K. and Vantassel, J. Gns: A generalizable graph neural network-based simulator for particulate and fluid modeling. *arXiv preprint arXiv:2211.10228*, 2022.

Li, J., Gao, Y., Dai, J., Li, S., Hao, A., and Qin, H. Mpmnet: A data-driven mpm framework for dynamic fluid-solid interaction. *IEEE Transactions on Visualization and Computer Graphics*, 2023a.

Li, X., Qiao, Y.-L., Chen, P. Y., Jatavallabhula, K. M., Lin, M., Jiang, C., and Gan, C. Pac-nerf: Physics augmented continuum neural radiance fields for geometry-agnostic system identification. *arXiv preprint arXiv:2303.05512*, 2023b.

Li, Y., Wu, J., Tedrake, R., Tenenbaum, J. B., and Torralba, A. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. *arXiv preprint arXiv:1810.01566*, 2018.

Li, Y., Sun, Y., Ma, P., Sifakis, E., Du, T., Zhu, B., and Matusik, W. Neuralfluid: Nueral fluidic system design and control with differentiable simulation. *Advances in Neural Information Processing Systems*, 37:84944–84967, 2024.

Li, Z. and Farimani, A. B. Graph neural network-accelerated lagrangian fluid simulation. *Computers & Graphics*, 103: 201–211, 2022.

Ma, P., Chen, P. Y., Deng, B., Tenenbaum, J. B., Du, T., Gan, C., and Matusik, W. Learning neural constitutive laws from motion observations for generalizable pde dynamics. In *International Conference on Machine Learning*, pp. 23279–23300. PMLR, 2023.

McNamara, A., Treuille, A., Popović, Z., and Stam, J. Fluid control using the adjoint method. *ACM Transactions On Graphics (TOG)*, 23(3):449–456, 2004.

Pan, Z., Huang, J., Tong, Y., Zheng, C., and Bao, H. Interactive localized liquid motion editing. *ACM Transactions on Graphics (TOG)*, 32(6):1–10, 2013.

Peebles, W. and Xie, S. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 4195–4205, 2023.

Pfaff, T., Fortunato, M., Sanchez-Gonzalez, A., and Battaglia, P. Learning mesh-based simulation with graph networks. In *International conference on learning representations*, 2020.

Raissi, M., Perdikaris, P., and Karniadakis, G. E. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019.

Raveendran, K., Thuerey, N., Wojtan, C. J., and Turk, G. Controlling liquids using meshes. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2012.

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

Schoentgen, A., Poulin, P., Darles, E., and Meseure, P. Particle-based liquid control using animation templates. In *Computer Graphics Forum*, volume 39, pp. 79–88. Wiley Online Library, 2020.

Sharabi, O. R., Lewin, S., and Louppe, G. A neural material point method for particle-based emulation. *arXiv preprint arXiv:2408.15753*, 2024.

Solmaz, S. and Van Gerven, T. Interactive cfd simulations with virtual reality to support learning in mixing. *Computers & Chemical Engineering*, 156:107570, 2022.

Stam, J. Stable fluids. In *Seminal Graphics Papers: Pushing the Boundaries, Volume 2*, pp. 779–786. 2023.

Tang, J., Liu, J., Zhang, M., and Mei, Q. Visualizing large-scale and high-dimensional data. In *Proceedings of the 25th International Conference on World Wide Web*, pp. 287–297, 2016.

Toshev, A. P., Erbesdobler, J. A., Adams, N. A., and Brandstetter, J. Neural sph: Improved neural modeling of lagrangian fluid dynamics. *arXiv preprint arXiv:2402.06275*, 2024.

Viswanath, H., Chang, Y., Panas, A., Berner, J., Chen, P. Y., and Bera, A. Reduced-order neural operators: Learning lagrangian dynamics on highly sparse graphs. *arXiv preprint arXiv:2407.03925*, 2024.

Volino, P., Cordier, F., and Magnenat-Thalmann, N. From early virtual garment simulation to interactive fashion design. *Computer-aided design*, 37(6):593–608, 2005.

Wandel, N., Weinmann, M., Neidlin, M., and Klein, R. Spline-pinn: Approaching pdes without data using fast, physics-informed hermite-spline cnns. In *Proceedings of*

the AAAI conference on artificial intelligence, volume 36, pp. 8529–8538, 2022.

Wang, J., Zhang, Y., Zou, J., Zeng, Y., Wei, G., Yuan, L., and Li, H. Boximator: Generating rich and controllable motions for video synthesis. *arXiv preprint arXiv:2402.01566*, 2024a.

Wang, Z., Yuan, Z., Wang, X., Li, Y., Chen, T., Xia, M., Luo, P., and Shan, Y. Motionctrl: A unified and flexible motion controller for video generation. In *ACM SIGGRAPH 2024 Conference Papers*, pp. 1–11, 2024b.

Winchenbach, R. and Thuerey, N. Symmetric basis convolutions for learning lagrangian fluid mechanics. *arXiv preprint arXiv:2403.16680*, 2024.

Xu, D., Nie, W., Liu, C., Liu, S., Kautz, J., Wang, Z., and Vahdat, A. Camco: Camera-controllable 3d-consistent image-to-video generation. *arXiv preprint arXiv:2406.02509*, 2024.

Yan, G., Chen, Z., Yang, J., and Wang, H. Interactive liquid splash modeling by user sketches. *ACM Transactions on Graphics (TOG)*, 39(6):1–13, 2020.

Yang, S., Hou, L., Huang, H., Ma, C., Wan, P., Zhang, D., Chen, X., and Liao, J. Direct-a-video: Customized video generation with user-directed camera movement and object motion. In *ACM SIGGRAPH 2024 Conference Papers*, pp. 1–12, 2024.

Yin, S., Wu, C., Liang, J., Shi, J., Li, H., Ming, G., and Duan, N. Dragnuwa: Fine-grained control in video generation by integrating text, image, and trajectory. *arXiv preprint arXiv:2308.08089*, 2023.

Yu, J., Yang, L., Xu, N., Yang, J., and Huang, T. Slimmable neural networks. *arXiv preprint arXiv:1812.08928*, 2018.

Zhang, L., Rao, A., and Agrawala, M. Adding conditional control to text-to-image diffusion models. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 3836–3847, 2023.

Zhang, Z., Liao, J., Li, M., Dai, Z., Qiu, B., Zhu, S., Qin, L., and Wang, W. Tora: Trajectory-oriented diffusion transformer for video generation. *arXiv preprint arXiv:2407.21705*, 2024.

Zhao, R., Gu, Y., Wu, J. Z., Zhang, D. J., Liu, J.-W., Wu, W., Keppo, J., and Shou, M. Z. Motiondirector: Motion customization of text-to-video diffusion models. In *European Conference on Computer Vision*, pp. 273–290. Springer, 2024.

Zhou, A., Li, Z., Schneier, M., Buchanan Jr, J. R., and Farimani, A. B. Text2pde: Latent diffusion models for accessible physics simulation. *arXiv preprint arXiv:2410.01153*, 2024.

A. Details of Related Works

Fluid Modeling and Animation Learning-based fluid simulators have progressed from graph-based models to hybrid, physics-informed approaches (Jain et al., 2024; Ma et al., 2023; Viswanath et al., 2024; Winchenbach & Thuerey, 2024). DPI-Net (Li et al., 2018) introduced dynamic interaction graphs with hierarchical message passing to model interactions across particles. This was unified in GNS (Kumar & Choi, 2023; Kumar & Vantassel, 2022; Pfaff et al., 2020; Sanchez-Gonzalez et al., 2020), enabling generalized simulation of fluids, solids, and deformables. (Raissi et al., 2019) introduced physics-informed neural networks (PINNs), training neural networks to respect physical laws described by nonlinear PDEs. Building on this, (Wandel et al., 2022) combines PINNs with CNNs via Hermite-spline kernels, enabling training without precomputed data and fast inference that generalizes to unseen domains. Hybrid solvers like MPMNet (Li et al., 2023a) and NeuralMPM (Sharabi et al., 2024) adopt the Material Point Method for scalability. Neural SPH (Toshev et al., 2024) integrates SPH priors to stabilize rollouts, while NeuroFluid (Guan et al., 2022) combines learned dynamics and rendering from videos. These advances balance physical accuracy with accelerated performance. PAC-NeRF (Li et al., 2023b) focused on inverse problems (system identification and geometry reconstruction from video) with Neural Radiance Fields for visual reconstruction. (Kumar & Choi, 2023) leverages differentiable GNS to accelerate particulate and fluid simulations for forward and inverse problems. (Li & Farimani, 2022) similarly applies graph networks to accelerate Lagrangian fluid simulation. In our work, we propose a hybrid approach that combines neural and numerical methods to enable accelerated and high-fidelity fluid simulation.

Fluid Control Recent work in fluid control aims to make simulation more intuitive and accessible. Traditional methods using space-time optimization were costly and hard to tune. Yan et al.(Yan et al., 2020) addressed this with a sketch-based system using conditional GANs to generate liquid splashes. Pan et al.(Pan et al., 2013) enabled control through sketching and mesh dragging. Chu et al.(Chu et al., 2021) used GANs to infer fluid motion from static fields with semantically controllable features. Schoentgen et al.(Schoentgen et al., 2020) introduced reusable templates for particle-based animations. These approaches shift toward flexible, artist-friendly tools. We tackle the case where only a freehand sketch is given, and the generative controller is tasked with producing the intended artistic fluid behavior. Crucially, our work fundamentally departs from classic optimization-based control methods, such as the adjoint method (McNamara et al., 2004). While such methods share the objective of directing fluid motion to match predefined keyframes, they are inherently offline and require solving computationally prohibitive, iterative optimization problems. In contrast, our generative controller interprets user-friendly freehand sketches to instantaneously predict dynamic force fields, enabling an accelerated control loop.

Controllable Video Generation Controllable video generation has advanced rapidly with diffusion models, especially in disentangling motion control (Chen et al., 2024; Jin & Xie, 2025; Li et al., 2024). DragNUWA (Yin et al., 2023) enabled trajectory-based editing, while MotionCtrl (Wang et al., 2024b) and Direct-a-Video (Yang et al., 2024) decoupled camera and object motion. CameraCtrl (He et al., 2024) and CamCo (Xu et al., 2024) refined camera control using geometric cues. MotionDirector (Zhao et al., 2024) and Boximator (Wang et al., 2024a) allowed user-customized motion, and SparseCtrl (Guo et al., 2024) enabled sparse, entity-level conditioning. Tora (Zhang et al., 2024) unified text, image, and trajectory inputs for physics-aware generation. Inspired by these approaches, we leverage forward simulations and compute control forces via reversed simulation.

B. Details of Neural Physics Simulator

B.1. Particle Simulations as Message-Passing on a Graph

We denote the state of a particle i at time step t as $\boldsymbol{x}_{i,t} \in \mathbb{R}^D$, and the collective state of N particles as $X_t = [\boldsymbol{x}_{1,t}, \ldots, \boldsymbol{x}_{N,t}] \in \mathbb{R}^{N\times D}$. Applying physical dynamics over multiple timesteps yields a trajectory of particle states, $\boldsymbol{X}_{t_1:t_{T_{\text{in}}}} = [X_{t_1}, X_{t_2}, \cdots, X_{t_{T_{\text{in}}}}] \in \mathbb{R}^{T_{\text{in}}\times N\times D}$. In essence, the simulator $s : \mathbb{R}^{T_{\text{in}}\times N\times D} \to \mathbb{R}^{N\times d}$ ($d = 2$ or 3 for 2D/3D) leverages the current T_{in} particle states as input to predict their future motion, capturing the underlying dynamics using methods ranging from simple Euler integration to advanced numerical or data-driven techniques. If a simulator is learnable, it can be represented as s_θ, a parameterized function approximator. The simulator then iteratively computes future states, such as $\tilde{X}_{t_{T_{\text{in}}+1}} = s(\tilde{X}_{t_1}, \tilde{X}_{t_2}, \cdots, \tilde{X}_{t_{T_{\text{in}}}})$, where each newly predicted state is appended to simulate a rollout trajectory over time.

Our learnable simulator s_θ represents the physical system as interacting particles, where dynamics emerge from exchanges of energy and momentum with neighbors. To ensure robust simulation quality, s_θ must generalize across diverse interaction

patterns and physical scenarios. This particle-based approach naturally maps to message passing on a graph, with particles as nodes and pairwise interactions as edges, making graph neural networks (GNNs) a suitable modeling choice.

B.2. Details of Graph-based Neural Physics

Following (Sanchez-Gonzalez et al., 2020), we implement our neural physics with GNN, and use standard nearest neighbor algorithms (Chen et al., 2009; Dong et al., 2011; Tang et al., 2016) to construct the graph.

Input. In our learnable simulator s_θ, the input state vector for each particle i at time step t_k includes a sequence of 5 previous velocities (via finite differences from $T_{\text{in}} = 6$ previous locations), and static features representing material properties (e.g., water, sand, rigid, boundary particle). In practice, only the position vectors $\boldsymbol{p}_i$ are stored in our datasets; the velocities $\dot{\boldsymbol{p}}_i$ and accelerations $\ddot{\boldsymbol{p}}_i$ are computed on the fly using finite differences when needed. Formally, the node feature is defined as

$$\boldsymbol{x}_{i,t_{k-T_{\text{in}}}:t_k} = [\dot{\boldsymbol{p}}_{i,t_{k-T_{\text{in}}+2}}, \ldots, \dot{\boldsymbol{p}}_{i,t_k}, \boldsymbol{f}_i] \in \mathbb{R}^D,$$

where $\boldsymbol{f}_i$ denotes the concatenated material-specific features and scene boundary indicators. Specifically, the dimension of the encoded node feature vector is $D = 30$ for 2D simulations (5 2-dim velocities by finite differences, i.e., $5 \times 2 = 10$; 4 distances from the boundary; 16-dim embedding for the particle type), or $D = 37$ for 3D simulations (5 3-dim velocities by finite differences, i.e. $5 \times 3 = 15$; 6 distances from the boundary; 16-dim embedding for the particle type). See Fig. 2 for an illustration. It is important to note that in our Fluid ControlNet (Sec. 3.2.3), the input feature dimension D will increase by 16, where we embed the current control timestep into the latent space with another 2-layer MLP with SiLU activation.

To obtain more informative edge features $\boldsymbol{r}_{i,j}$, we use the relative positional displacement between a pair of adjacent particles i and j, along with its magnitude:

$$\boldsymbol{r}_{i,j} = [(\boldsymbol{p}_i - \boldsymbol{p}_j), \|\boldsymbol{p}_i - \boldsymbol{p}_j\|].$$

Edges are added between particles that lie within a predefined *connectivity radius* $R = 0.015$, which captures local particle interactions. R is kept constant for all 2D scenarios. In different 3D scenarios, a larger radius can be used to accommodate higher-resolution environments. Although R is fixed in simulations, edges in the graph are still dynamically updated by comparing the current particle-wise distances to R. For full details of these input and target features, we refer readers to (Sanchez-Gonzalez et al., 2020).

The $\textsc{Encoder} : \mathbb{R}^{N \times D} \to \mathcal{G}$ embeds particle-based states, it can be formulated as: $G^{(0)} = (\boldsymbol{V}^{(0)}, \boldsymbol{E}^{(0)}) = \textsc{Encoder}(\boldsymbol{X}, \boldsymbol{r}_{i,j})$ The node embeddings $\boldsymbol{V}^{(0)} = \textsc{Encoder}_V(\boldsymbol{X})$ are learned functions of the particles' states. The edge embeddings, $\boldsymbol{E}^{(0)}_{i,j} = \textsc{Encoder}_E(\boldsymbol{r}_{i,j})$, are learned functions of the pairwise properties of the corresponding particles. We implement $\textsc{Encoder}_V$ and $\textsc{Encoder}_E$ as multilayer perceptrons (MLP), which encode node features and edge features into the latent vectors, $\boldsymbol{V}_i$ and $\boldsymbol{E}_{i,j}$, of size 128.

The $\textsc{Processor} : \mathcal{G} \to \mathcal{G}$ computes interactions among nodes through L steps of learned message passing and outputs the final graph, $G^{(L)} = \textsc{Processor}(G^{(0)})$. Message passing enables information propagation among particles. Our $\textsc{Processor}$ consists of a stack of $L = 10$ GNN layers, each using separate (non-shared) MLPs for updating node and edge features, along with residual connections between the input and output latent attributes of both nodes and edges. For the Fluid ControlNet setting, an additional MLP layer is used to encode the diffusion timestep and control image features; see Appendix C.3 for details.

The $\textsc{Decoder} : \mathcal{G} \to \mathbb{R}^{N \times d}$ extracts dynamics information (of the future state) from the nodes of the final latent graph, $\hat{X} = \textsc{Decoder}(\boldsymbol{V}^{(L)})$. Our $\textsc{Decoder}$ is an MLP that outputs accelerations $\ddot{\boldsymbol{p}}_i$. The future position and velocity are updated using an Euler integrator.

All MLPs in $\textsc{Processor}$ have two hidden layers with ReLU, followed by an output layer without activation, with a width of 128. All MLPs are followed by a LayerNorm (Ba et al., 2016).

C. Implementations

C.1. Latency Measurements

Latency of Neural Physics. We utilize the `TensorRT` library to convert the `PyTorch` model into an `ONNX` model to accelerate model inference and align it with the acceleration of MPM on the `Taichi` kernel. However, since `TensorRT`

does not support the `aggregation` operation in GNNs (i.e., aggregating information from edges to adjacent nodes), when measuring the latency, we approximate the time cost of this `aggregation` operation with a matrix multiplication between an adjacency matrix $A \in \mathbb{R}^{N \times N}$ (where N denotes the number of nodes, i.e. particles), and node features ($\boldsymbol{o}$), such that the aggregation becomes $A \cdot \boldsymbol{o}$. All reported latency measurements are based on the median number of nodes across different scenarios in our test datasets.

Latency of Taichi. To enable a fair comparison under MPM simulation setting, we applied a matching latency reduction strategy to the `Taichi` implementation by skipping non-essential overhead. Specifically, we excluded the time spent on initializing the MPM state (initial positions and velocities of particles) and the cost of initializing the `Taichi` kernel at the beginning of the simulation. As a result, our comparison focuses solely on the runtime per simulation step after the `CUDA` or `Taichi` kernel has been initialized.

C.2. Physical Simulation Parameters

We detail the physical parameters used in our MPM simulations. For Sand materials, we set the density (ρ) to 400. The elastic properties are governed by a Young's Modulus (E) of 4.0×10^5 and a Poisson's Ratio (ν) of 0.25, from which the Lamé parameters (μ, λ) are derived. The friction coefficient (μ_b) is set to 1.0, and the Drucker-Prager hardening model parameters are configured as $h_0 = 35$, $h_1 = 9$, $h_2 = 0.2$, and $h_3 = 10$. For Water materials, we use a density (ρ) of 1.0 and a Young's Modulus (E) of 400 to govern incompressibility. Gravity $\boldsymbol{g}$ is set to 9.8 across all environments.

C.3. Design of Fluid ControlNet

In our Fluid ControlNet, the control signal $\mathcal{C} \in \mathbb{R}^{H \times W \times 3}$ is encoded using our Fluid ControlNet. The encoded embedding is then injected into the graph-based diffusion model to guide the generation of the external field of accelerations. The Fluid ControlNet consists of 8 convolutional layers and 3 downsampling operations. It extracts multi-scale features from the control signal $\mathcal{C}$, projects each scale to a different dimensional space, and then concatenates the projected features into control embedding representation of dimension size 44. The resulting embedding is then integrated into the PROCESSOR module of the graph-based diffusion model. Notably, to better condition the diffusion process on the control signal, we draw inspiration from DiT (Peebles & Xie, 2023) and concatenate the embedding of the control signal to the diffusion time step embedding. This design choice ensures that the control condition is effectively incorporated at each diffusion step, thereby generating high-fidelity acceleration fields that can align fluid particles to the target motion or shape.

C.4. Training

Following (Sanchez-Gonzalez et al., 2020), we normalize the input velocity to the GNNs, and apply random noises to input positions ($\boldsymbol{p}_{t_1:t_{T_{\text{in}}}}$) during training. For both neural physics and Fluid ControlNet, we train with the Adam optimizer and a learning rate at 1×10^{-4} with exponential decay. Our training batch size is 1, and we train for 2 million gradient descent steps.

Table 5. Training Costs (GPU hours) across different scenarios.

GPU Hours	Water (2D)	Sand (2D)	Water (3D)	Sand (3D)
Neural Physics (Sec. 3.1)	17.27h	17.94h	19.71h	19.67h
Fluid ControlNet (Sec. 3.2)	69.87h	76.36h	184.12h	151.03h

We include our training costs in Table 5. Neural physics requires approximately one day on a single NVIDIA 4090 GPU. For the Fluid ControlNet, training takes around three days for 2D scenarios on a single NVIDIA 4090 and six days for 3D scenarios on a single NVIDIA A40. We train both neural physics and Fluid ControlNet with the particle-level RMSE loss on predicted accelerations $\text{RMSE}_{\ddot{\boldsymbol{p}}} \equiv \frac{1}{N}\sum_{i=1}^{N} \frac{\left\|\hat{\ddot{\boldsymbol{p}}}_i - \ddot{\boldsymbol{p}}_i\right\|_2}{\left\|\ddot{\boldsymbol{p}}_i\right\|_2}$, which was defined in Sec. 2.2.

C.5. Generating Users' Freehand Sketches (Arrows and Oval Shapes)

Arrows are computed by connecting the centroid ($\bar{\boldsymbol{p}} = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{p}_i$) of fluid particles at $t = 1$ ($\bar{\boldsymbol{p}}_1$) and $t = T_{\text{ctr}}$ ($\bar{\boldsymbol{p}}_{T_{\text{ctr}}}$). Based on the mean displacement vector $\Delta\bar{\boldsymbol{p}} = \bar{\boldsymbol{p}}_1 - \bar{\boldsymbol{p}}_{T_{\text{ctr}}}$, we derive the arrow length $\|\Delta\bar{\boldsymbol{p}}\|$ and orientation $\theta = \tan^{-1}(\Delta\bar{\boldsymbol{p}}_y / \Delta\bar{\boldsymbol{p}}_x)$.

In 3D, we use the arrow width to indicate depth (Pan et al., 2013). A multi-segment arrow with varying line width is implemented as n-segment polyline with width modulation, where each segment's width w_i ($i \in [1, n]$) is $w_i = w_{\min} + (w_{\max} - w_{\min}) \cdot \frac{\Delta\bar{\boldsymbol{p}}_{z,i} - \Delta\bar{\boldsymbol{p}}_{z,\min}}{\Delta\bar{\boldsymbol{p}}_{z,\max} - \Delta\bar{\boldsymbol{p}}_{z,\min}}$. The arrowhead adopts perspective-correct scaling.

For 2D oval sketches, shapes of particles at $t = T_{\text{ctr}}$ are represented as elliptical outlines centered at $\bar{\boldsymbol{p}}_{T_{\text{ctr}}}$, with radii corresponding to $\pm 2\boldsymbol{\sigma}$, where $\boldsymbol{\sigma}$ is the standard deviation of particle positions along each principal axis. This statistically-grounded ellipse captures approximately 95% particles' positions while being visually simple. Meanwhile, 2D oval-shaped control sketches can indeed be ambiguous in 3D, since it is infeasible to depict 3D volumes with a simple one-stroke 2D sketch. While 2D sketches provide a direct mapping, 3D control involves inherent depth ambiguity. Our model leverages arrow thickness as a depth proxy, but true 3D-aware sketching remains a frontier for future generative physics.

C.6. Enforcing Smoothness on Target Accelerations

We observe that ground truth accelerations solved by Equation 3 are typically complicated (see the temporal-wise cosine similarity in Fig. 13), which will be challenging to learn. We thus further enforce a certain level of smoothness of the acceleration across temporal steps:

$$\boldsymbol{a}_{t,\text{smooth}} = \boldsymbol{a}_t - \lambda \cdot \exp\left(-\beta \cdot \frac{\boldsymbol{a}_t \cdot \boldsymbol{a}_{t+1}}{\|\boldsymbol{a}_t\| \cdot \|\boldsymbol{a}_{t+1}\|}\right) \cdot (\boldsymbol{a}_t - \boldsymbol{a}_{t+1}) \tag{4}$$

Essentially, Equation 4 enforces decoupled smoothness over the magnitude and the orientation of accelerations over temporal steps. We choose $\lambda = 0.1$ and $\beta = 2$ in our work.

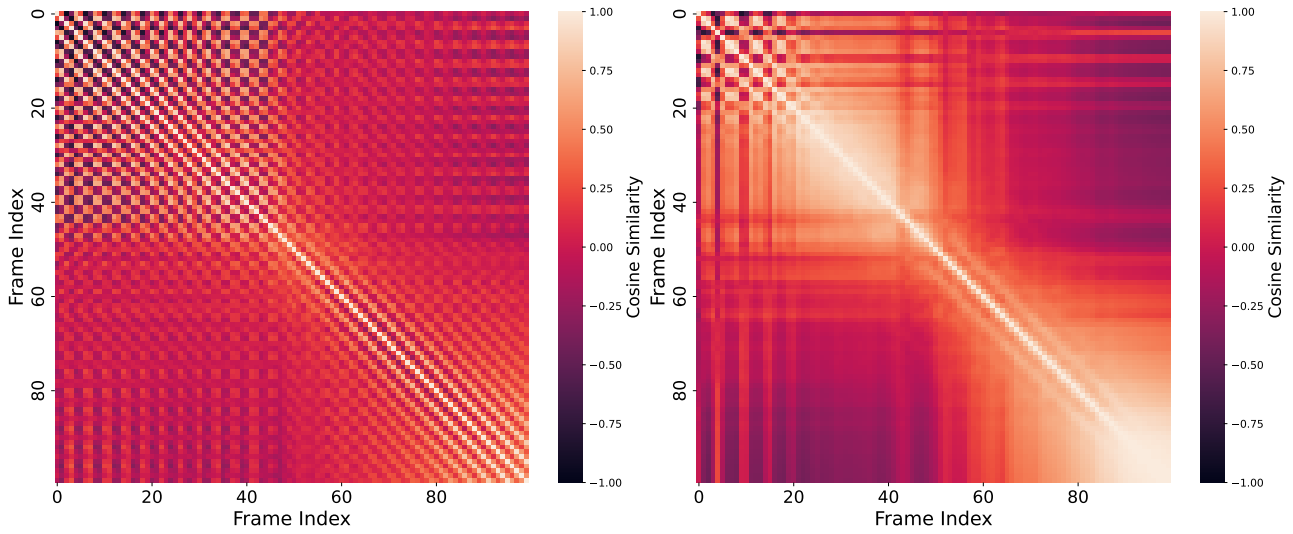


Figure 13. Step-wise correlations of ground-truth accelerations for fluid control. Left: before enforcing smoothness; Right: after enforcing smoothness.

D. User Study Details and Discussion

D.1. Discussion on Evaluating Physical Realism

Evaluating "Physical Realism" in art-directable fluid control raises a subtle conceptual issue. When a fluid is guided to form an unnatural configuration, such as a floating arrow or a prescribed oval, the motion is driven by external forces that would not normally arise in natural environments. Since sufficiently flexible external forces can in principle produce arbitrary particle trajectories, it may appear unclear what it means to assess the "realism" of such simulations.

In this work and user study, Physical Realism does not refer to whether the overall event would naturally occur. Instead, it measures whether **the intrinsic material behavior of the fluid is preserved during the guided transition**. Even in highly stylized visual effects where fluids are manipulated by artificial forces, the material itself is still expected to behave as a fluid rather than as a rigid body or a deformable solid. The key question is therefore whether the internal dynamics remain consistent with plausible fluid behavior.

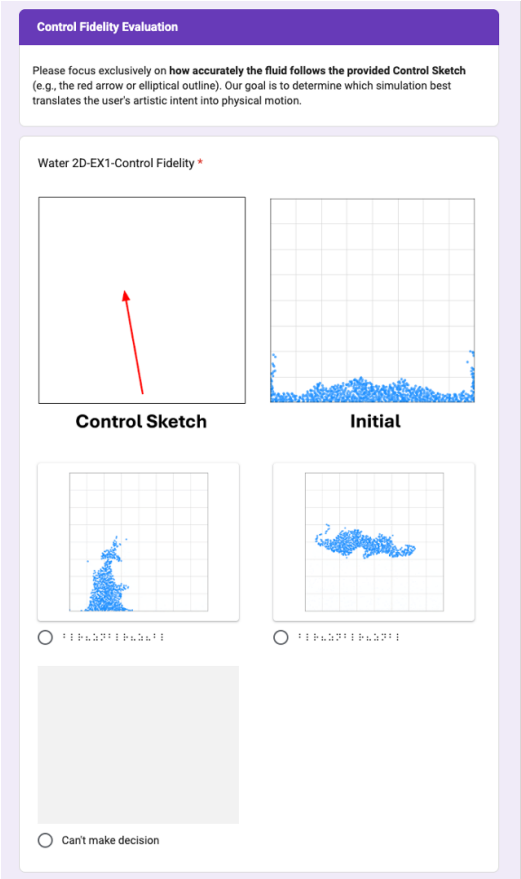


Figure 14. Screenshot of the Google Forms interface used in our user study. Participants first see the initial condition and control sketch, followed by two randomized and anonymous simulation results. A "Can't make decision" option is provided for ambiguous cases.

Participants were instructed to judge realism based on the relative motion and interaction of particles. A simulation is considered more physically realistic if it exhibits continuous flow, natural deformation, coherent splashing or scattering, and consistent volume preservation. Incompressibility for water and granular characteristics for sand should remain visually apparent. In contrast, simulations are penalized if they introduce kinematic artifacts such as stiff, synchronized motion resembling rigid translation, abrupt collapses, unnatural clustering, or noticeable discontinuities.

This separation between macroscopic trajectory and microscopic dynamics is central to our evaluation. The global motion toward a target shape and position is assessed under Control Fidelity. Physical Realism instead evaluates whether the internal evolution of the material remains consistent with fluid behavior throughout that motion. This distinction allows us to determine whether a method can satisfy artistic intent while preserving perceptually plausible dynamics.

D.2. Questionnaire Design and Interface

We conducted a user study using Google Forms. We recruited 25 participants to evaluate 17 conditioned samples. To ensure a comprehensive and rigorous evaluation, these test cases were carefully curated to cover a diverse range of physical scenarios, including both 2D and 3D environments with different material properties (specifically, WATER2D, SAND2D, WATER3D, and SAND3D).

To prevent biased responses, the presentation order of the two simulated videos was randomized for each question. Additionally, to account for user confidence and avoid forced choices in ambiguous cases, we included a "Can't make decision" (tie) option.

The survey began with the following introduction and general instructions:

We are conducting a user study to evaluate a framework for controllable fluid simulation. All data collected is

anonymous and will be used solely for academic research.

*For each case, you will see a **Control Sketch** representing the user's intent (e.g., an arrow for movement or an oval for a target position) and an **Initial Condition**. You are then presented with 2 simulated animations: **2 anonymous Videos**.*

This survey is divided into 2 sections, each focusing on a specific evaluation metric:

- **Section 1 Control Fidelity:** *Evaluates how well the animation aligns with the motion or shape intended by the user's sketch.*
- **Section 2 Physical Realism:** *Evaluates which animation looks more like real-world fluid (water or sand), maintains volume, and avoids visual artifacts.*

Important Note: *If you find that the 2 anonymous Videos are very similar (or both fail to achieve reasonable quality), and you cannot make a clear decision, please select the "Can't make decision" option.*

Depending on the section, participants were given specific criteria to base their evaluations on. For the **Control Fidelity** section, users were instructed:

Please focus exclusively on how accurately the fluid follows the provided Control Sketch (e.g., the red arrow or elliptical outline). Our goal is to determine which simulation best translates the user's artistic intent into physical motion.

For the **Physical Realism** section, to align with our theoretical definition in Appendix D.1, users were instructed:

Please focus on the natural behavior and physical plausibility of the material (water or sand). A high-quality simulation should look like the real world, maintaining volume and exhibiting realistic dynamics.

A screenshot demonstrating the Google Forms survey interface is provided in Fig. 14.

E. More Explanation on Grid-RMSE

E.1. Explanation of Grid-RMSE

Grid-RMSE (RMSE_m) is a normalized error metric used to evaluate the difference between the predicted mass grid and the ground-truth mass grid in simulations. It is defined as the normalized grid-level $\text{RMSE}_{\tilde{\boldsymbol{m}}} \equiv \frac{1}{N}\sum_{i=1}^{N}\frac{\left\|\hat{\tilde{\boldsymbol{m}}}_i - \tilde{\boldsymbol{m}}_i\right\|_2}{\left\|\tilde{\boldsymbol{m}}_i\right\|_2}$, which essentially quantifies the mass distribution. $\tilde{\boldsymbol{m}}$ is the normalized grid mass ($\tilde{\boldsymbol{m}}_i = \frac{\boldsymbol{m}_i}{\sum_{i=1}^{N}\boldsymbol{m}_i}$).

E.2. The Effectiveness of Grid-RMSE

Why we choose grid-RMSE as the metric? Standard particle-level RMSE is intractable in low-resolution settings due to significant differences in the number of particles. Moreover, performing upsampling or downsampling will break the particle alignments: the predicted and ground-truth particles will no longer be one-to-one mappings, leaving particle-RMSE infeasible.

Instead, evaluating the normalized mass distribution on the grid offers a more stable and meaningful approximation of the overall fluid shape. Essentially, it quantifies the IoU (intersection-over-union) of the predicted and target fluids.

To further validate the reliability of this metric, we have added an additional experiment based on Fig. 5(d), where we compute RMSE directly at the original resolution ($r_p = 1$). As shown in Table 6, results from particle RMSE and grid-level RMSE_m are consistent, which supports the validity and robustness of using mass RMSE in our setting.

We study whether we can address the particle misalignment issue by downsampling the reference MPM simulator to $r_p = 1/1.75$, ensuring both methods operate on identical sparse particle distributions. Under this setting, we compare two metrics: particle RMSE (our method vs. MPM at $r_p = 1/1.75$) and grid-level mass RMSE. As shown in Table 7, both metrics exhibit consistent trends across various r_c values, providing additional validation for the reliability of mass-based RMSE in our experimental framework.

Table 6. Comparison of RMSE and mass-based RMSE (RMSE_m) under different r_c values for the WATER2D setup ($r_t = 2, r_p = 1$).

	$r_c = 0$	$r_c = 0.3$	$r_c = 0.5$	$r_c = 0.6$	$r_c = 0.7$	$r_c = 0.8$	$r_c = 0.9$
Particle RMSE	0.2780	0.2741	0.2619	0.2528	0.2437	0.2112	0.1844
RMSE_m	0.0238	0.0231	0.0225	0.0223	0.0208	0.0175	0.0132

Table 7. Comparison of Particle RMSE and mass-based RMSE (RMSE_m) under different r_c values for the WATER2D setup ($r_t = 2, r_p = 1/1.75$).

	$r_c = 0$	$r_c = 0.3$	$r_c = 0.5$	$r_c = 0.6$	$r_c = 0.7$	$r_c = 0.8$	$r_c = 0.9$
Particle RMSE	0.2755	0.2610	0.2550	0.2603	0.2375	0.2176	0.1619
RMSE_m	0.0232	0.0223	0.0214	0.0207	0.0192	0.0169	0.0144

We also need to emphasize that "applying G2P to obtain particle acceleration at high particle resolution" requires introducing new particles. Refer to line88: `for p in x: # grid to particle (G2P)` in `mpm128.py` from the taichi-dev GitHub repository. This means that if no new particles are introduced, the number of particles in x will remain unchanged, causing misalignment between simulation particles at different resolutions.

To implement the approach of "applying G2P to obtain particle acceleration at high particle resolution," we explored two methods for introducing new particles:

1. randomly seeding new particles, and

2. using a learned point-cloud upsampler (from the pointcloud-upsampling GitHub repository).

Our experiments revealed that both methods yield excessively high particle RMSE values (Table 8). A relative RMSE greater than 1 clearly indicates that upsampling during G2P severely disrupts particle alignment and is not a viable solution.

Table 8. Particle RMSE comparison for different upsampling methods in WATER2D setup ($r_t = 2, r_p = 1/1.75$).

Upsampling Method	**Random Upsampling**	**Point Cloud Upsampling**
Particle RMSE	1.3153	1.2335

However, if we do not upsample and directly compare GNN with the ground-truth MPM with the same number of particles (i.e. particles are always aligned), the particle RMSE is much smaller. (0.2755 vs. 0.2780). This indicates that misaligned particles introduced by upsampling dominate the metric and lead to misleading evaluations. In contrast, our grid-level mass RMSE is computed directly on the simulation output, without any heuristic postprocessing or resampling, and remains stable across all tested resolutions. For this reason, we consider it a more reliable proxy for assessing cross-resolution fidelity.

F. More Results

F.1. Comparison with Previous Neural Physics Methods.

We have listed and compared numerous related works in Table 9, highlighting key differences between their approaches and ours, such as the use of misaligned input modalities. Among them, GIOROM (Viswanath et al., 2024) and NeuralMPM (Sharabi et al., 2024) are the most closely related to our work, as they also perform validations based on GNS (Sanchez-Gonzalez et al., 2020).

F.2. Grid $\mathrm{RMSE}_{\tilde{m}}$ of Fluid Simulations over Random Seeds

To ensure a fairer comparison, we conducted experiments using three different random seeds. The results, as shown in Table 10, demonstrate that our hybrid solver consistently outperforms the original neural physics across all datasets.

Table 9. Comparison of recent neural physics methods.

Dataset	Simulation Type	Method	Input	Control	Physics Correction
GNS (Kumar & Choi, 2023; Kumar & Vantassel, 2022; Li et al., 2018; Pfaff et al., 2020; Sanchez-Gonzalez et al., 2020)	Neural Simulator	GNN	Particle States	✗	✗
MPMNet (Li et al., 2023a)	Hybrid Simulator	MPM + ConvLSTM	Pressure Fields	✗	✓
NeuralMPM (Sharabi et al., 2024)	Neural Simulator	Voxelized CNN	Voxelized Grid	✗	✗
SGNN (Han et al., 2022)	Neural Simulator	Subequivariant GNN	Particle States	✗	✗
NCLaw (Ma et al., 2023)	Hybrid Simulator	Neural Constitutive Model + PDE	Deformation Gradient	✗	✓
GIOROM (Viswanath et al., 2024)	Neural Simulator	GNN + Neural Fields	Velocity Fields	✗	✗
Ours	Hybrid Simulator	GNN + MPM + Diffusion	Particle States + User Action	✓	✓

Table 10. Grid $\mathrm{RMSE}_{\tilde{m}}$ of fluid simulations on different scenarios, over three random runs.

$\mathrm{RMSE}_{\tilde{m}}$	Water (2D)	Sand (2D)	SandRamps (2D)	WaterRamps (2D)	Water (3D)	Sand (3D)	Water-Sand (2D)
Neural Physics	0.0263 (1.15e-6)	0.0125 (2.59e-7)	0.0101 (3.23e-8)	0.0229 (2.09e-6)	0.0048 (6.58e-7)	0.0025 (2.11e-8)	0.0441 (3.51e-6)
Our Hybrid Solver	0.0186 (8.17e-6)	0.0116 (6.88e-8)	0.0096 (1.00e-9)	0.0171 (3.16e-6)	0.0022 (1.77e-8)	0.0013 (1.08e-7)	0.0149 (2.38e-6)

F.3. Quantitative Results of Fluid Control

To complement the qualitative visualizations and the user study presented in the main text, we quantitatively evaluate the control fidelity of our method against the baseline. We measure this by calculating the grid-level $\mathrm{RMSE}_{\tilde{m}}$ between the ground truth and the prediction at the final time step of the control window. This metric is chosen because our primary control objective is the accurate recovery of the target shape or spatial distribution of the fluid at the end of the simulation.

Table 11. Grid $\mathrm{RMSE}_{\tilde{m}}$ between ground truth and predictions at the final time step during fluid control.

Method	Water (2D)	Sand (2D)	Water (3D)	Sand (3D)
Baseline	0.0908	0.1151	0.0019	0.0022
Ours	**0.0802**	**0.0924**	**0.0013**	**0.0019**

As shown in Table 11, our diffusion-based Fluid ControlNet achieves consistently lower RMSE values across both 2D and 3D scenarios for different materials (Water and Sand). The baseline method relies on solving a spatiotemporal constant force field to linearly transport particles, which struggles to accurately form complex shapes due to chaotic internal particle collisions. In contrast, our learned dynamic force fields successfully guide the particles to better align with the user's sketch-based targets while preserving the underlying fluid mechanics.

F.4. Latency of Fluid ControlNet

For accelerated performance, we report diffusion model inference time across scenes, as shown in Table 12. The latency is measured after model compilation and kernel warm-up, ensuring that initialization overhead is excluded. Further details of the latency measurement methodology are provided in Appendix C.1.

Table 12. Latency of our Fluid ControlNet.

Dataset	Water-2D	Sand-2D	Water-3D	Sand-3D
Latency (ms)	18.714	20.724	20.316	27.026

F.5. More Visualizations

Fluid Simulations. Fig. 15 presents the visualizations of all models discussed throughout the paper. Here, we show a comparison of intermediate frames from a single trajectory. It is evident that, due to the hybrid design of our hybrid solver, our method produces visual results that are more similar to MPM ($r_p = 1/1.75$) simulations. Since MPM ($r_p = 1/1.75$) is highly consistent with MPM (ground truth), the outputs of our Hybrid solver also align better with MPM compared to the original neural physics. This demonstrates that our approach effectively balances computational efficiency and accuracy.

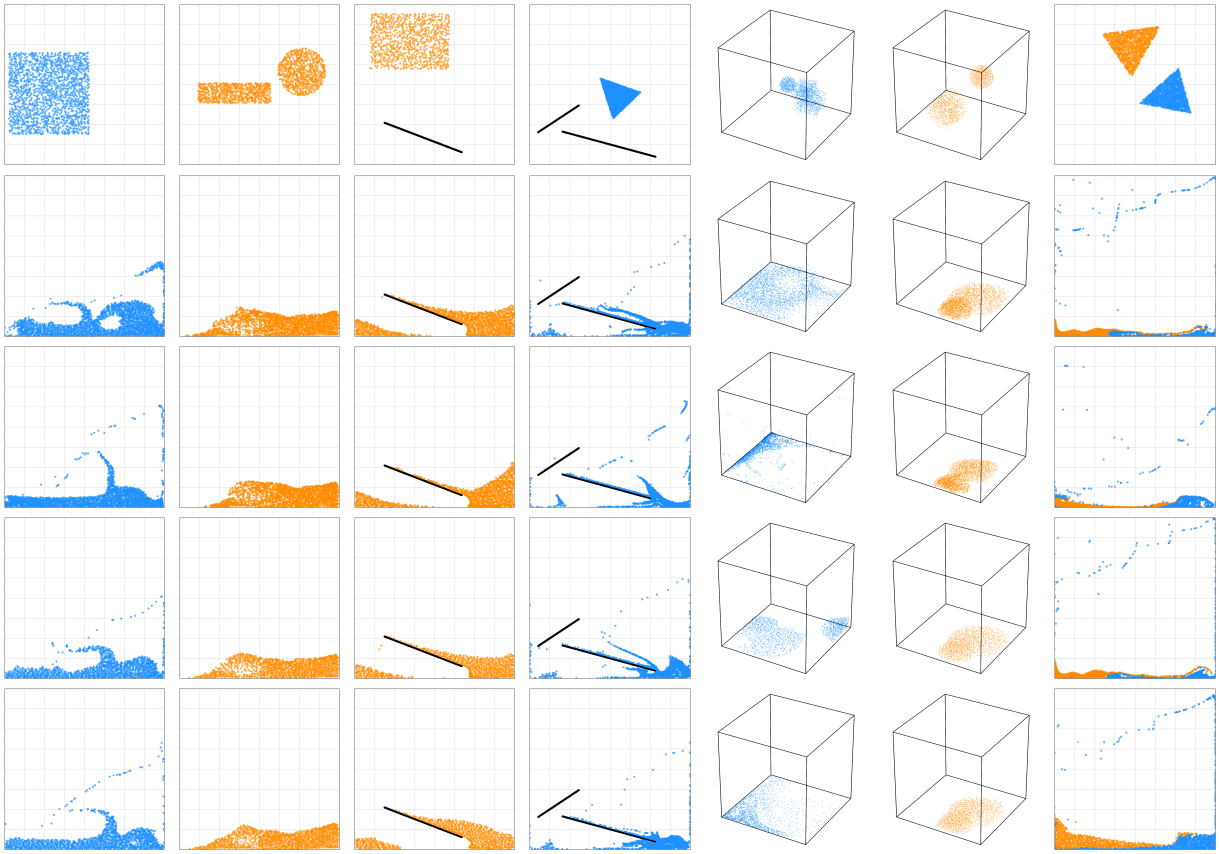


Figure 15. Visualizations of fluid simulations by different methods, over different scenarios. From left to right: Water (2D), Sand (2D), SandRamps (2D), WaterRamps (2D), Water (3D), Sand (3D), Water-Sand (2D). From top to bottom: Initial, MPM (ground truth), Original Neural Physics, MPM ($r_p = 1/1.75$), Our Hybrid Solver.

More Visualizations of Fluid Control. Fig. 16 presents additional visualizations of generative fluid control across a variety of tasks, both 2D and 3D control signals. We can see that our approach consistently generates physically plausible and visually accurate outcomes that align closely with the target controls across all fluid types and dimensions, demonstrating strong control capability. These results further confirm the effectiveness of our method in achieving both visually appealing and physically plausible fluid control.

Causes of Diffusion Failure in Some 3D Scenes. Our diffusion-based controller performs robustly in most 2D scenarios, where the spatial relationship between the user sketch and fluid particles is straightforward. However, in 3D simulations, we observe a slight performance drop, mainly due to two challenges: 1) Lack of explicit depth information: User sketches are

inherently 2D, making it difficult to encode depth cues in 3D space. While we use arrow thickness as a heuristic (Section 3.2.3, Appendix B), this may not effectively convey depth to the model. 2) Spatial misalignment: 2D sketches, such as ovals, can be ambiguous when projected into 3D. This mismatch may degrade the learned force field during diffusion, reducing control fidelity.

Despite these issues, the generated force fields still guide the fluid in the intended direction, and performance remains qualitatively acceptable. For high-precision or depth-sensitive 3D control, future work could explore 3D-aware sketching or explicit 3D conditioning.

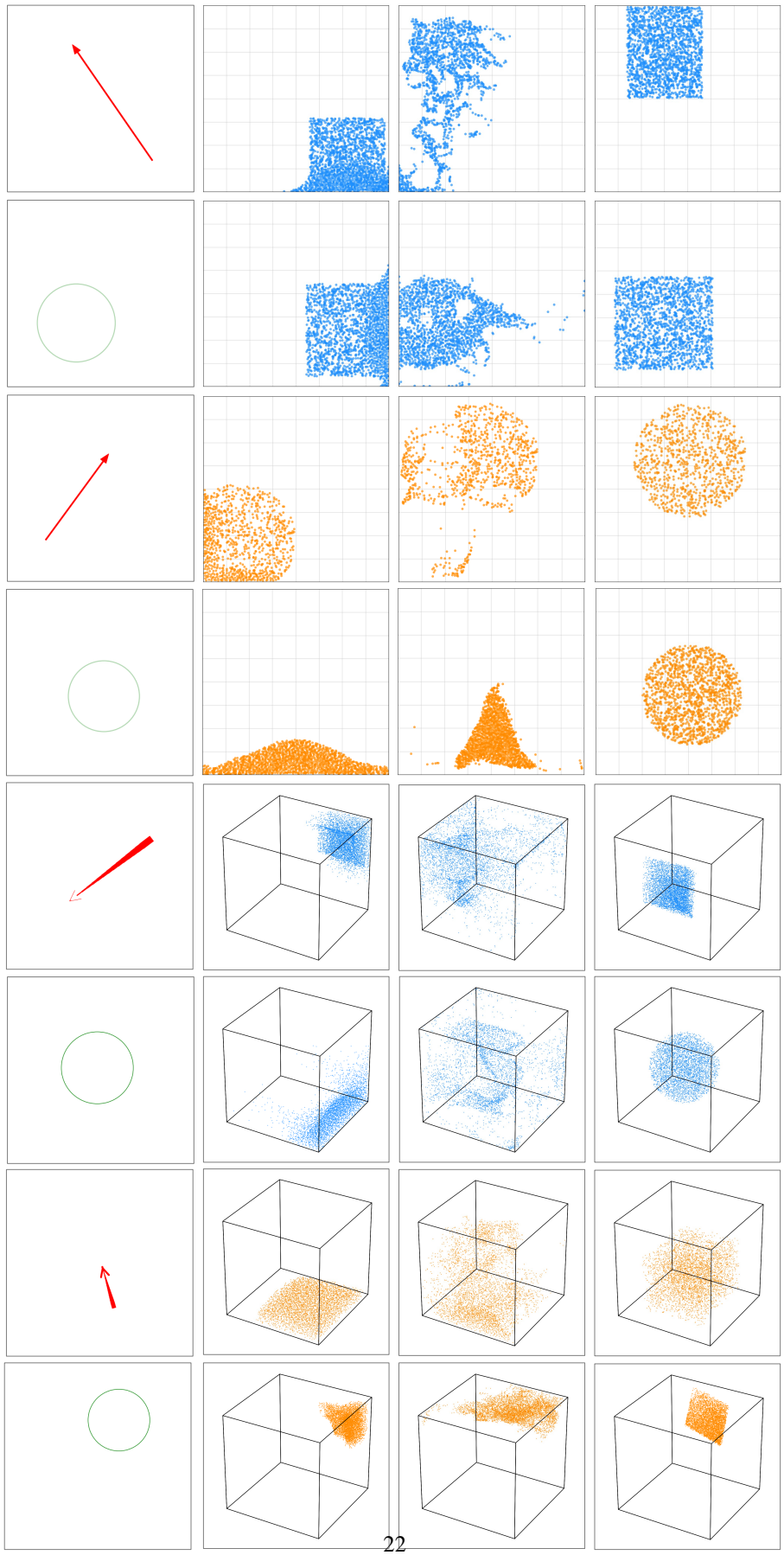


Figure 16. More visualization of generative fluid control. From top to bottom: Water2D, Sand2D, Water3D, and Sand3D, each with two types of control signals (arrows for motion direction, and oval shapes for target spatial positions). From left to right: control signal, initial, ours, ground truth.