Learning Particle Dynamics Subject to Rigid Body Manipulations Using Graph Neural Networks

Niteesh Midlagajni
Centre for Cognitive Science
Technical University of Darmstadt
niteesh.midlagajni@tu-darmstadt.de

Constantin A. Rothkopf
Centre for Cognitive Science
Technical University of Darmstadt
constantin.rothkopf@tu-darmstadt.de

Abstract

Simulating particle dynamics with high fidelity is crucial for solving real-world interaction and control tasks involving liquids in design, graphics, and robotics. Recently, data-driven approaches, particularly those based on graph neural networks (GNNs), have shown progress in tackling such problems. However, these approaches are often limited to learning fluid behavior in static free-fall environments or simple manipulation settings involving primitive objects, often overlooking complex interactions with dynamically moving kinematic rigid bodies. Here, we propose a GNN-based framework designed from the ground up to learn the dynamics of liquids under rigid body interactions and active manipulations, where particles are represented as graph nodes and particle-object collisions are handled using surface representations with the bounding volume hierarchy (BVH) algorithm. Our approach accurately captures fluid behavior in dynamic settings and can also function as a simulator in static free-fall environments. Despite being trained on single-object manipulation tasks, our model generalizes effectively to environments with novel objects and novel manipulation tasks. Finally, we show that the learned dynamics can be leveraged to solve control and manipulation tasks using gradient-based optimization methods.

1 Introduction

Our physical world is shaped by complex interactions between various forms of matter, from solid objects to flowing liquids. Accurately simulating these phenomena is essential across numerous disciplines, including science and engineering, and becomes increasingly critical as we advance toward a future with autonomous agents capable of reasoning, planning, and interacting with objects in novel environments. Moreover, understanding human intuitive physical reasoning and interactions in everyday tasks in the natural environment also requires the capability of simulating physical scenarios [1, 2]. Traditional physics simulators, relying on hand-crafted features and computationally intensive numerical methods, often struggle to scale to the complexity of real-world scenarios.

Recently, learning-based simulators have offered a viable alternative, demonstrating the ability to learn complex dynamics directly from data [3–5], improve simulation accuracy, and enable natural integration with gradient-based optimization for solving inverse problems [6, 7]. Graph neural networks (GNNs) [8, 9], in particular, have been successful in recent years in capturing the dynamics of diverse physical systems, including liquids, soft bodies, and rigid bodies [10–15]. While existing GNN-based simulators have shown impressive results in modeling particle dynamics, significant limitations remain. Many current approaches [10, 15] are restricted to simple collision scenarios with basic geometric shapes, hindering their applicability to real-world rigid body interactions. While recently, Mani et al. [14] improved boundary handling with complex rigid bodies via Signed Distance Fields (SDFs), the model is still constrained to free-fall simulations, where the rigid bodies remain fixed in place during simulation. Accurately modeling dynamic liquid interactions with moving objects (kinematic rigid bodies) is essential for applications in sequential decision-making tasks

Midlagajni et al., Learning Particle Dynamics Subject to Rigid Body Manipulations Using Graph Neural Networks. *Proceedings of the Fourth Learning on Graphs Conference (LoG 2025)*, PMLR 269, Arizona State University, Phoenix, USA, December 10–12, 2025.

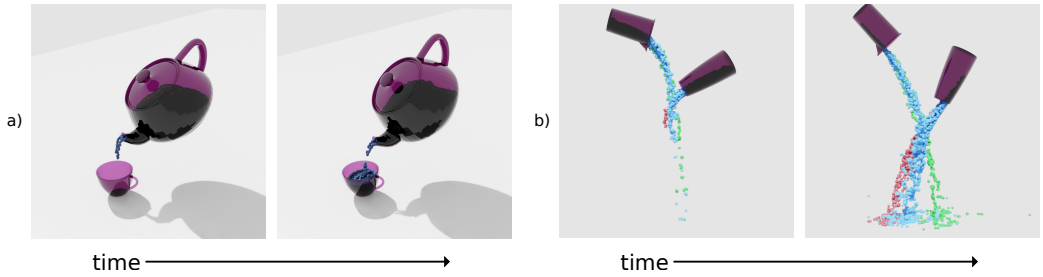

Figure 1: Rollouts from our proposed model on novel scenarios. (a) Successful simulation of pouring from the complex Utah teapot. (b) Simultaneous manipulation of two jugs, demonstrating multi-object control. The blue liquid streams, originating from each jug, collide and merge realistically. The red and green particles represent the predicted trajectories for each jug if simulated independently.

such as robotic control. A simulator designed for free-fall settings can address only design-oriented tasks (e.g., directing fluid flow [16]), and is insufficient for tasks requiring manipulation and control. Crucially, even with highly accurate collision handling, a static framework does not guarantee correct behavior when objects move, as dynamic interactions introduce additional complexities in fluid motion and response. Therefore, achieving accurate dynamic simulations requires careful design choices, including the appropriate GNN input representation and architectural modifications.

To address these challenges, we propose a unified GNN-based framework that models complex fluid dynamics with dynamically moving rigid bodies. Our architecture leverages a multi-graph representation where liquid particles and rigid bodies are represented as distinct node sets. Liquid–liquid interactions are determined by spatial proximity (particles are connected if they lie within a fixed radius), while liquid–object interactions are determined by particle–surface proximity computed efficiently using the BVH algorithm. We train our model on ground truth simulations of generic *pouring* and *sliding* tasks, collectively covering a broad range of liquid-object interactions under dynamic control. Our results show: 1) improved performance both qualitatively and quantitatively over existing GNN-based particle simulators; 2) strong generalisation to unseen object geometries and scene configurations; and 3) successful zero-shot transfer to novel manipulation tasks such as *stirring* and *scooping*. Finally, we demonstrate the utility of our model by integrating it into a gradient-based optimization pipeline to solve a pouring task using a Model Predictive Control (MPC) controller. Taken together, these results demonstrate that our proposed model extends the capabilities of GNN-based simulators to include complex interactions of liquids with kinematic rigid bodies.

2 Related Work

With the rise of deep learning and the increasing preference for end-to-end models, differentiable simulators [17] have become a compelling alternative to traditional physics-based simulators such as Bullet [18] and Mujoco [19] for simulating physics. Unlike conventional simulators, these analytical differentiable simulators enable automatic differentiation, allowing seamless integration with gradient-based optimization techniques [20–23]. Several frameworks have been developed for fluid dynamics, such as PhiFlow [24], which is framework-agnostic, JAX-Fluids [25], built on JAX, and FluidLab [26], which leverages Taichi [20]. Hybrid approaches have also emerged, combining analytical equations with neural networks to improve accuracy and generalization [27–29].

Purely learning-based simulators, in particular, Graph Neural Networks (GNNs) [10–15, 30–34] have gained significant traction due to their ability to model complex physical interactions. This success can be attributed to the inherent nature of many physical phenomena – the dynamics of rigid bodies, deformable bodies, fluids, and even their coupled interactions can often be described by local interactions between constituent entities. GNNs, with their strong relational inductive bias [9], provide a natural and effective framework for encoding this locality, allowing the model to learn the underlying physics from data. The seminal work by Sanchez-Gonzalez et al. [10] established several key architectural and representational choices that have influenced a number of subsequent works. These include using relative geometric features to define node and edge attributes, employing one-step

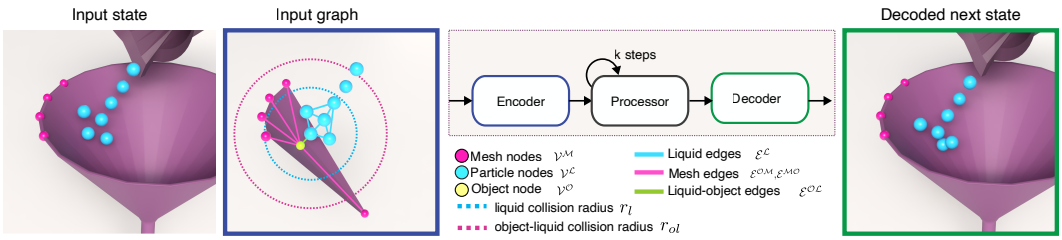

Figure 2: Overview of our network architecture and graph encoding scheme. The figure shows the graph connectivity of a liquid particle with its neighboring particles and a Cup object. Note that, for illustration purposes only, the closest point on the surface of the cup to the particle is shown as $\mathcal{V}^O$. In reality, the object node $\mathcal{V}^O$ is virtual and located at the object's center of mass.

prediction training instead of full rollout-based supervision, and utilizing message-passing GNNs [35, 36] as the core model. While this work primarily focused on particle-based fluid simulation, later studies successfully extended similar architectures to soft-body [11] and rigid-body [12, 30, 32–34] simulations.

In physical simulation, accurately modeling collisions is essential. In GNN-based simulators, this is typically addressed through node-based neighborhood interactions. This approach has proven effective for simulating liquid-liquid [10] and deformable body [11] interactions. However, it struggles when dealing with rigid-body collisions, as mesh vertices are often sparse and fail to capture fine-grained object geometry [12–14]. To address these limitations, Allen et al. [12] proposed handling collisions in the mesh-face space instead of the vertex space, leading to more robust rigid-body interactions. Another widely adopted approach is the use of SDFs for collision handling. SDFs implicitly represent surface geometry by encoding the distance of any point in space to the closest surface on a mesh. This allows for efficient collision detection by evaluating the SDF at query points. Rubanova et al. [13] used learned SDFs to simulate rigid body dynamics while Mani et al. [14] used SDF formulation to learn fluid dynamics in a static free-fall environment, representing all rigid surfaces in the environment as a single union of SDFs. These methods significantly reduce the computational overhead compared to node-based collision handling, offering a scalable alternative for complex physical interactions.

So far, the aforementioned GNN-based simulators have either been designed for static free-fall environments, restricting their use to design optimization tasks [14, 16], or have supported manipulation only in limited settings, typically involving simple rigid bodies like cuboids [10, 15] or spheres [11]. In contrast, our model extends beyond these constraints by generalizing to arbitrary scenes and complex manipulation tasks, marking a significant step toward a fully learned, general-purpose differentiable simulator capable of handling diverse physical interactions.

3 Method

Our learned simulator predicts the dynamics of liquid particles interacting with kinematic rigid bodies under active manipulations. Given the initial state of the liquid particles X^0 in terms of their positions, and external 6-DOF control inputs at every time step $U^{(0)}, ..., U^{(T-1)}$ for all the rigid bodies in the scene, the GNN simulator iteratively applies the learned model $X^{(t+1)} = s_\theta(X^t, U^t)$ and produces the rollout trajectory of particles as $X^{(1)}, ..., X^{(T)}$. The rigid bodies in our framework can either be *kinematic* objects, which can be subjected to external control, or *stationary* objects, that remain fixed throughout the simulation. At timestep t, each *kinematic* object can be transformed to a new 6-DOF pose under the transformation $\mathcal{T}_t$. For *stationary* objects, pose remains fixed throughout the simulation.

Our graph network follows the Encode-Process-Decode architecture introduced in [10]. The state of the world is first encoded into a graph where node features are derived from the current state, and edge connections are determined based on factors such as spatial proximity for potential collisions and the underlying geometry of objects. Subsequently, both node and edge features are embedded into a latent space via MLPs. The final encoded graph is passed through P message-passing layers to

generate a sequence of P latent graphs. Finally, a decoder, which is also implemented as an MLP, extracts the updated dynamics from task-relevant nodes of the final latent graph.

3.1 Graph Network

We define the graph as $\mathcal{G} = (\mathcal{V}^L, \mathcal{V}^O, \mathcal{V}^M, \mathcal{E}^L, \mathcal{E}^{OL}, \mathcal{E}^{OM}, \mathcal{E}^{MO})$. In the input graph representation, $\mathcal{V}^L = \{p_i\}_{i=1...N}$ corresponds to the set of N liquid particles. $\mathcal{V}^O = \{o_i\}_{i=1...Q}$ represents the set of Q rigid-bodies in the scene. $\mathcal{V}^M = \{m_{ik}\}_{i=1...Q,k=1...K_i}$ represents the set of surface vertices of all the objects in the scene. The surface vertices are defined such that when an object o_i is transformed to a new 6-DOF pose at time t, the surface vertices m_{ik} belonging to the object also move under the same transformation. Figure 2 gives an overview of our architecture.

Graph Connectivity in our model is handled using four sets of edges. Following [12, 13], we connect bi-directional edges between object node o_i and its corresponding surface nodes $\{m_{ik}\}$, forming edge sets $\mathcal{E}^{OM}$ (object-to-mesh) and $\mathcal{E}^{MO}$ (mesh-to-object). This facilitates the propagation of object motion to the particles through the surface nodes. Interaction between liquid particles is captured using the edge set $\mathcal{E}^L$. We connect edges between liquid particles if the distance between them is less than the liquid-liquid connectivity radius r_l. This promotes local interactions of particles, and we choose radius r_l to ensure each particle has roughly 10-20 neighbors.

Crucially, to model the interactions between fluid particles and rigid bodies, we introduce the $\mathcal{E}^{OL}$ edge set. Similar to liquid-liquid interactions, the dynamics of fluid particles are primarily affected when they are near a rigid body's surface. Notably, this interaction can occur at any arbitrary point on the object's surface, not just at mesh vertices. To achieve this, we define a function $\mathbf{C}(p'_j, o_i)$ based on the Boundary Vector Hierarchy (BVH) algorithm [37]. This function takes the mesh geometry of object o_i and a query particle p'_j as input, and efficiently computes the closest point c'_{ij} on the surface of the mesh to the query particle. Since this function operates on positions in the object's local frame, we first transform the liquid particle position p_j into this frame using the inverse transformation $\mathcal{T}_t^{-1}(p_j)$ based on the object's current transformation $\mathcal{T}_t$. The closest point is then transformed back to the world coordinate frame:

$$c_{ij} = \mathcal{T}_t(\mathbf{C}(\mathcal{T}_t^{-1}(p_j), o_i)) \tag{1}$$

If the distance d between the particle p_j and the transformed closest point c_{ij} is less than the liquid-object connectivity radius, r_{ol}, we connect an edge between the particle node and the object node. This approach is similar to [13], where learned SDFs for collision detection between rigid bodies were used. We use a different connectivity radius r_{ol} for liquid-object interactions since our ground truth simulator maintains some separation distance between the object surface and the particles to enforce its boundary condition.

Node features Following the approach of [10–14], we use a history of velocity information as the main node feature. Specifically, we use the finite-difference estimates of velocity from the previous five timesteps. We calculate these velocity features differently depending on the node type. For liquid particle nodes $\mathcal{V}^L$ and mesh surface nodes $\mathcal{V}^M$, we calculate them based on their positions over the past five timesteps. For object nodes $\mathcal{V}^O$, which represent the rigid bodies, we derive velocity information from the object's 6-DOF pose at each timestep. This 6-DOF pose information is crucial for the network to learn the rotational dynamics of the objects, in addition to their translational motion. Additionally, we also add features to distinguish between different types of object nodes $\mathcal{V}^O$. Our simulation includes *kinematic* objects (which are controlled) and *stationary* objects (which remain fixed). To represent this, we use a one-hot encoding to indicate its type. This one-hot encoding is also extended to the mesh nodes $\mathcal{V}^M$ since each mesh node belongs to a specific object.

Edge features For the edge sets $\mathcal{E}^L$ (liquid-liquid), $\mathcal{E}^{MO}$ (mesh-to-object), and $\mathcal{E}^{OM}$ (object-to-mesh), we adopt the relative encoding technique [10]. The edge features consist of the relative displacement vector between two connected nodes and the scalar distance between them. The calculation of these relative features differs slightly depending on the edge type. For $\mathcal{E}^L$, the relative position and distance are computed between the positions of the two connected liquid particles. For $\mathcal{E}^{MO}$ and $\mathcal{E}^{OM}$ edges, we calculate them based on the mesh node position and the center of mass position of the corresponding object, derived from the object node's 6-DOF pose. For the liquid-object edge set $\mathcal{E}^{OL}$, which is critical for modeling particle collisions with surfaces, we use a specialized feature set. For an edge connected between particle p_j and object o_i, the feature vector is

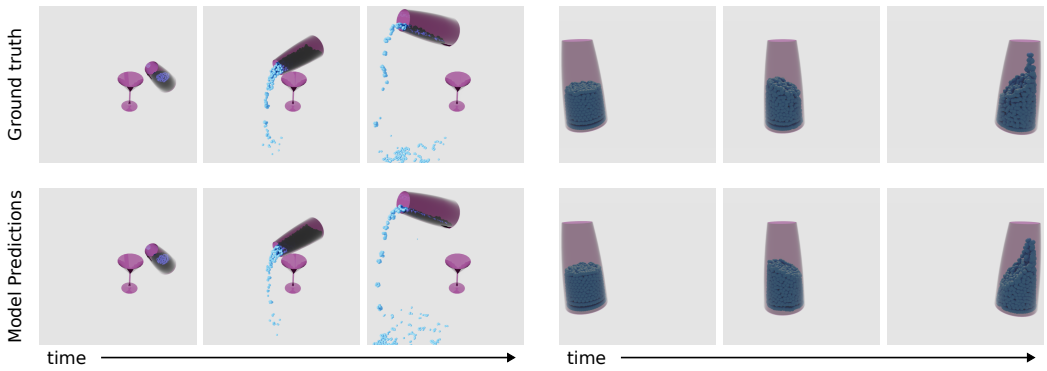

Figure 3: Example rollouts of our *Full-model* on the held-out test set.

defined as $e_{ij}^{ol} = [c_{ij} - p_j, c_{ij} - p_{o_i}, \|c_{ij} - p_j\|, \|c_{ij} - p_{o_i}\|]$. Here, c_{ij} represents the closest point on the surface of object o_i to the particle p_j, and p_{o_i} represents the center of mass of object o_i. These features together give information about the spatial relationship between the connected particle and object, using the closest point on the object's surface c_{ij}, as a reference. The resulting input graph is encoded into separate MLPs for each node type and edge type before passing to the processor block.

Processor The processor stage consists of P identical message-passing steps, each with its own learned parameters. These blocks iteratively update the embedded node and edge features using the following update equations:

$$e_{ij}'^{L} = f^{LL}(e_{ij}^{L}, \mathbf{v}_i^{L}, \mathbf{v}_j^{L}) \tag{2}$$
$$e_{ij}'^{OL} = f^{OL}(e_{ij}^{OL}, \mathbf{v}_i^{O}, \mathbf{v}_j^{L}) \tag{3}$$
$$e_{ij}'^{MO} = f^{MO}(e_{ij}^{MO}, \mathbf{v}_i^{M}, \mathbf{v}_j^{O}) \tag{4}$$
$$e_{ij}'^{OM} = f^{OM}(e_{ij}^{OM}, \mathbf{v}_i^{O}, \mathbf{v}_j^{M}) \tag{5}$$
$$\mathbf{v}_i'^{M} = f^{M}(\mathbf{v}_i^{M}, \textstyle\sum_j e_{ij}'^{OM}) \tag{6}$$
$$\mathbf{v}_i'^{O} = f^{O}(\mathbf{v}_i^{O}, \textstyle\sum_j e_{ij}'^{MO}) \tag{7}$$
$$\mathbf{v}_i'^{L} = f^{L}(\mathbf{v}_i^{L}, \textstyle\sum_j e_{ij}'^{L}, \textstyle\sum_j e_{ij}'^{OL}) \tag{8}$$

where f^{LL}, f^{OL}, f^{MO}, f^{OM}, f^{M}, f^{O}, f^{L} are each implemented as MLPs.

Decoder The processor updates the latent representations of all the node types. However, since our goal is to predict particle dynamics under the influence of rigid body manipulations, the decoder, implemented as an MLP, only decodes latent features of liquid nodes $\mathcal{V}^L$. We train the model using the $L2$ loss on one-step acceleration predictions of the liquid particles. Additional details on the network architecture and training procedure are provided in Appendix A.1.

4 Experiments

4.1 Ground Truth Simulator

We use NVIDIA Isaac Sim [38] as our ground truth simulator. Isaac Sim is a robotics simulation platform that also supports particle simulation based on the Position Based Dynamics (PBD) [39] method. To investigate the ability of our system to generalize from a small set of training data, our primary dataset consisted of a scene with a collection of liquid particles and three rigid bodies: a jug (*kinematic*), a cup (*stationary*), and a floor (*stationary*). We adjusted the properties of particles to approximate those of water. Each simulation episode was initialized as follows: the cup was placed at a random location, the floor was positioned underneath it, and the jug was placed above the floor, containing the liquid particles. To enable the network to learn liquid dynamics under a wide range of jug motions, as well as the effects of liquid collisions with stationary objects (cup and floor), we create three types of simulation scenarios:

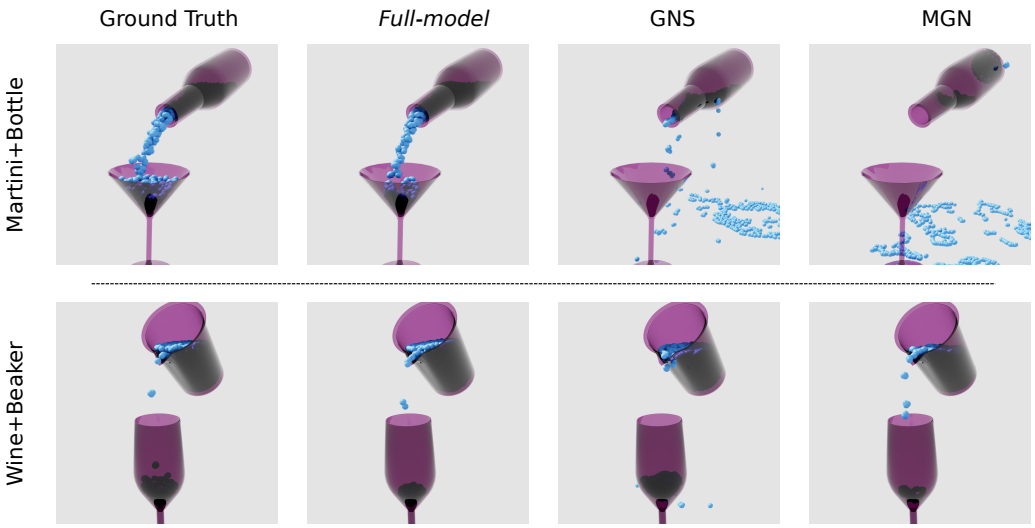

Figure 4: Comparison with the baseline models. GNS and MGN struggle to handle particle dynamics with novel geometry, while our method generalises without any additional training.

- **Translation-Motion-Sim**: The jug undergoes purely translational motion, with a random velocity along the x, y, or z axis at each timestep. This scenario teaches the network how linear motion of a container affects the enclosed liquid.
- **Rotation-Motion-Sim**: The jug remains positionally fixed but rotates with a random angular velocity around a random axis. In half the cases, the rotation is configured to pour liquid into the cup, and in the other half, the liquid spills on the floor. This helps the model learn the effects of rotational motion on the liquid, as well as collisions with the stationary objects.
- **Full-Body-Motion-Sim**: The jug undergoes combined translational and rotational motion. It moves upwards along a specified direction in the xy-plane while simultaneously rotating with a random angular velocity. This motion is designed to simulate a pouring action and allows the network to learn complex fluid dynamics resulting from 6-DOF motion.

To train the network to handle abrupt changes in liquid motion, we introduce random noise to the jug's motion in 30% of the trials. We use a *Martini* cup and *Vase* jug to generate the dataset for training our model. Our training and test sets contain 1200 and 120 simulations(each with 415 timesteps), respectively.

Generalization Test Sets To evaluate our model's generalization capabilities, we created additional *Rotation-Motion-Sim* simulations using two novel jug shapes, *WineBottle* and *Beaker*, and a *Wine* cup. Our training dataset used the *Vase* jug, which has a relatively simple geometry. In contrast, the *WineBottle* has a wide base that tapers sharply to a narrow opening. The *Beaker* has a spout at the rim, channeling the liquid into a narrower stream. We also used a *Wine* cup, which features a curved geometry, unlike the conical shape of the *Martini* cup used during training.

4.2 Evaluation

Baseline models We compare our model against recent GNN-based particle simulators that support kinematic rigid body manipulation, specifically Graph Network Simulator (GNS) [10] and MeshGraphNet [11]. Both models handle liquid-object interaction using node-based collision. GNS uses a single edge set to model both liquid-liquid and liquid-object interactions. In contrast, MeshGraphNet separates liquid-liquid and liquid-object interactions into two distinct edge sets. We evaluate two variants of MeshGraphNet based on how the liquid-object edge set is defined. In the first variant (MGN), it includes only liquid-object edges. In the second variant (MGN*), mesh connectivity edges are included alongside the liquid-object edges. All 3 models use a single node set to represent both particle and mesh nodes. As a result, the 6-DOF control inputs for manipulating a rigid body are incorporated into the node features of all the nodes in the graph. This implicit representation of

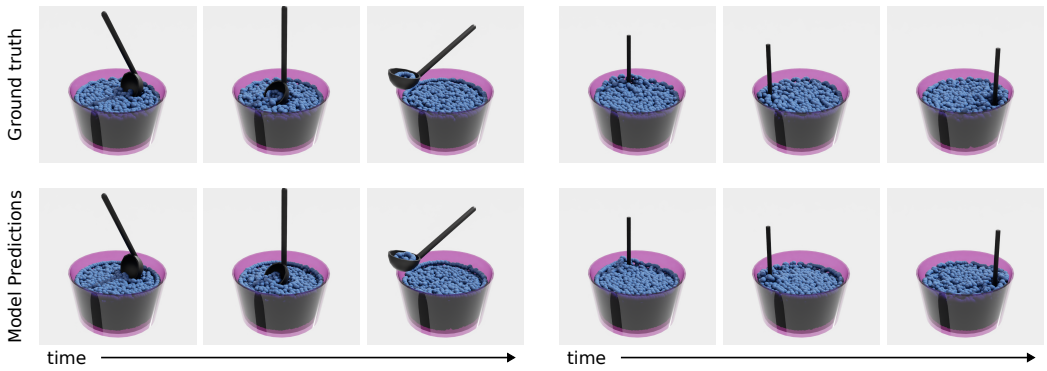

Figure 5: Rollouts of our *Full-model* showing successful *Scooping* (left) and *Stirring* (right) manipulation tasks

control inputs limits these models to manipulating only one rigid object per simulation. We use the same neighborhood radii as in our model: r_l for liquid-liquid collision and r_{ol} for liquid-object collision. See Appendix A.2 for complete details of the baseline models.

Ablation We hypothesized that for our task, the mesh vertex nodes $\mathcal{V}^M$ might be less critical for learning liquid-object interaction, and that the direct interaction between $\mathcal{V}^L$ and $\mathcal{V}^O$ via our accurate collision handling method might suffice. While the mesh nodes $\mathcal{V}^M$ do contribute to updating latent states of object nodes $\mathcal{V}^O$ within the processor block, their direct influence on the liquid nodes $\mathcal{V}^L$ might be limited. To test this, we created an ablated version of our model by removing the mesh vertex nodes $\mathcal{V}^M$ and the corresponding edge sets $\mathcal{E}^{OM}$ and $\mathcal{E}^{MO}$. This results in a simplified input graph: $\mathcal{G} = (\mathcal{V}^L, \mathcal{V}^O, \mathcal{E}^L, \mathcal{E}^{OL})$. The processor update equations are reduced to Equations (2), (3), and (8), with Equation (7) simplifying to $\mathbf{v}_i'^O = \mathbf{v}_i^O$ (meaning the object node features are not updated within the processor block). This ablated model has a lower memory footprint during both training and inference. In the rest of the paper, we refer to our complete network as *Full-model* and to this reduced version as *Ablated-model*. The baseline models and the *Ablated-model* were trained using the same dataset as our proposed model.

Metric For quantitative comparison, we use the mean Chamfer distance over the entire rollout trajectory between the ground truth and the model prediction. Chamfer distance is more suitable for evaluating point cloud data than root mean squared error (RMSE), as it emphasizes overall distributional similarity rather than exact one-to-one correspondences between particles.

4.3 Results

Our graph network model reliably predicts particle dynamics under external manipulations of rigid bodies. We first evaluate the *Full-model*'s long-term rollout performance on the test set by comparing the predicted rollouts with the ground truth simulator, qualitatively assessing the model's ability to maintain realistic motion over extended periods. During rollouts, the ground truth 6-DOF control inputs for the jug are provided to the network at each timestep. Figure 3 presents qualitative results for our *Full-model*, showing rollout snapshots from two challenging scenarios. In the *sliding* task (Figure 3 (right) from the *Rotation-Motion-Sim* test set), when the externally controlled jug comes to an abrupt stop, our model accurately handles both the boundary collisions and the conservation of momentum of the liquid particles. In the *full-motion* task (Figure 3 (left) from the *Full-Body-Motion-Sim*), the jug is manipulated across all six degrees of freedom. Our network successfully tracks the intricate dynamics throughout the rollout. Notably, on these test sets, both the *Ablated-model* and the GNS baseline exhibit qualitatively similar performance to the *Full-model*, whereas the MGN and MGN* models struggle to maintain consistency over the entire rollout.

Generalization A key advantage of our model is its ability to generalize to novel object shapes. Figure 4 compares rollouts from our *Full-model*, GNS, and MGN on the Wine-Beaker and Martini-Bottle test sets. Despite never encountering these shapes during training, our model accurately simulates the liquid dynamics, closely matching the ground truth behavior. Importantly, our *Ablated-model* also performs comparably well, suggesting that both models effectively learn to represent

Table 1: Quantitative comparison of *Full-model*, *Ablated-model*, and the baseline models using mean Chamfer Distance (**lower is better**) with respect to ground truth simulations. N represents the number of particles. All simulations had a rollout length of 415 timesteps.

Simulation domain	N	*Full-model*	*Ablated-model*	GNS	MGN	MGN*
Translation-Motion-Sim	1K	0.057±0.01	**0.055±0.01**	0.063±0.02	0.092±0.06	0.089±0.04
Rotation-Motion-Sim	1K	**0.086±0.03**	0.099±0.05	0.099±0.04	0.178±0.18	0.163±0.07
Full-Body-Motion-Sim	1K	**0.138±0.07**	0.165±0.09	0.184±0.09	0.493±0.37	0.236±0.14
Wine-Beaker	1K	**0.091±0.02**	0.111±0.035	0.156±0.01	0.141±0.04	0.347±0.06
Martini-Bottle	1K	0.077±0.01	**0.069±0.01**	2.02±0.15	1.317±0.15	2.261±0.17

local surface geometry through our liquid-object collision handling. In contrast, the GNS and MGN baselines exhibit substantial limitations. With the *Beaker*, GNS shows pronounced tunneling effects, with particles incorrectly passing through the spout walls. While MGN is able to channel particles through the spout, it still suffers from tunneling toward the end of the rollout. With the *WineBottle*, both GNS and MGN struggle to simulate the liquid flow as the particles approach the narrow neck, resulting in significant particle tunneling. This difficulty stems from their reliance on mesh vertices alone to handle liquid-object collisions, which fails to capture finer geometric details of these novel shapes. Importantly, this is despite increasing the mesh vertex density of the objects by 2-5 times to facilitate better shape information. MGN* performs worse than both GNS and MGN across all evaluated scenarios. Table 1 summarizes the quantitative results for each model, while Figure 6 shows the step-wise error over the trajectory. Our *Full-model* consistently achieves the best scores across most test scenarios. Additional snapshots can be found in Appendix A.4.

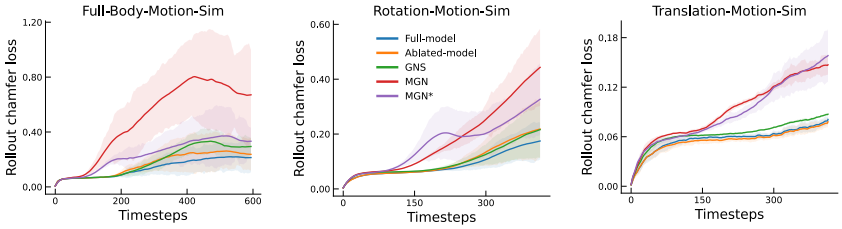

Figure 6: Mean Chamfer loss curves over rollout on the three test sets for all models. Our model achieves the lowest error; shaded regions show variability as median absolute deviation.

We further validated the generalization capabilities of our *Full-model* by simulating liquid pouring from the classic Utah teapot, which features intricate geometry. As shown in Figure 1, the *Full-model* (chamfer loss of 0.078) successfully simulates realistic liquid dynamics even for this challenging shape. The *Ablated-model* (chamfer loss of 0.301), however, exhibited some leakage with the Utah teapot, suggesting a potential limitation in handling highly complex geometries. The difference in performance may be due to the unavailability of object node $\mathcal{V}^O$ updates during the message-passing process in the *Ablated-model*.

In practice, since both the *Full-model* and *Ablated-model* offer similar performance in most scenarios, the two can be seen as complementary. The *Ablated-model* is suitable for environments with rigid objects of relatively simple shapes, as it reduces computational overhead and memory footprint, while the *Full-model* is preferable for scenes containing highly intricate geometries.

4.4 Novel Manipulation Scenarios

Having demonstrated in the previous section that our network generalises to novel objects, we now investigate our model's ability to handle unseen manipulation tasks. We focus on *Stirring* and

Scooping tasks, two common liquid manipulation tasks found in kitchen settings that are significantly different from the pouring motions used during training. We created a new simulation environment consisting of a large pot filled with liquid particles (2.1K particles). In the *Stirring* task, a stick is controlled to stir the liquid. In the *Scooping* scenario, a ladle is manipulated to scoop liquid from the pot. We compare the performance of our *Full-model* against ground truth simulations generated by Isaac Sim. Figure 5 illustrates the trajectories of these two manipulation tasks performed by our *Full-model* along with the ground truth. In both the manipulation tasks, our model successfully simulates the complex liquid dynamics. For the *Stirring task* (chamfer loss of 0.0514), the network accurately models the pushing forces exerted by the stick on the liquid particles as it rotates. For the *Scooping task* (chamfer loss of 0.0593), the model captures both the scooping action and the subsequent containment and transport of the liquid within the ladle. Additionally, we also showcase our model's ability to simultaneously control multiple objects. We created a two-jug pouring scenario, where each jug is independently controlled to pour liquid. Figure 1 shows the snapshots of the result from our *Full-model* (chamfer loss of 0.257), where it realistically simulates the interaction of liquid streams from two sources. Taken together, these results show that our learned simulator is indeed a versatile general-purpose simulator for modeling liquid dynamics. It accurately captures complex interactions between liquids and rigid bodies of arbitrary shapes undergoing diverse manipulations.

4.5 Control on learned dynamics

A key advantage of a fully differentiable environment is the ability to solve tasks using gradient-based optimization techniques. To showcase the utility of our model, we designed a pouring task where the objective is to pour a specified percentage of the cup's total volume from the jug into the cup. In the initial state, the jug is placed in an upright position next to the cup, such that rotating the jug around the x-axis will cause liquid to pour into the cup. We use an MPC controller with the control variable being the jug's rotational acceleration angle around the x-axis. The control inputs are initialized randomly. Following a task formulation similar to [27], we implement a two-stage cost function optimized using Adam [40]. In the first stage, the cost is the $L2$ distance between all liquid particles and a target point on the rim of the jug closest to the cup. This encourages the jug to rotate towards the cup to let the liquid flow into the cup. Once the target fill level is reached, the second stage activates. This stage uses a regularization term on the jug's rotation angle, encouraging it to return to its initial upright position.

Figure 7 shows snapshots of the optimized trajectory for a target fill level of 30%. Despite the random initialization of control inputs, the optimizer successfully finds a control sequence that achieves the desired fill level. We tested the controller with four different target fill levels, corresponding to 10%, 25%, 50%, and 70% of the cup's volume, and observed achieved fill levels of 10%, 44%, 72%, and 94%, respectively. For higher fill targets, there is a larger deviation from the desired level, likely due to the dynamics of the two-stage control: at higher fill targets, the jug rotates more in the first stage. When the second stage activates and the jug begins to return to its upright position, residual liquid continues to pour, leading to overfilling.

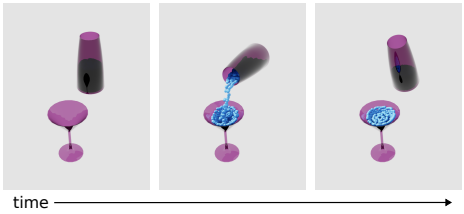

Figure 7: Optimal trajectory snapshots from our MPC controller for a 30% fill target.

4.6 Computational Complexity and Inference Times

For liquid-object collisions, our BVH-based handling reduces complexity compared to node-based baselines. BVH is relatively inexpensive, with a complexity of $\mathcal{O}(\log(K_i))$ per query, where K_i is the number of vertices in the mesh. In our method, each particle is connected to at most one edge per object, yielding an $\mathcal{O}(NQ)$ complexity for N liquid particles and Q objects. In node-based collision, such as in GNS, a particle may connect to many vertices, resulting in $\mathcal{O}(NK)$ complexity, where K is the total number of object mesh vertices. Since the number of objects is significantly lower than the total number of node vertices($Q << K$), a substantial speedup is possible.

We calculated the rollout inference times per step ($dt = 1/60$) for all our models and baselines on the test set (evaluated on a workstation with Nvidia 4090 GPU and 32-core CPU) and report them in Table 2. Indeed, the performance is twice as fast as the fastest baseline methods. However, it is

Table 2: Comparison of Inference Times across all models.

Method	Time per step (ms)
Full-model	25.05
Ablated-model	21.34
GNS	75.51
MGN	50.32
MGN*	49.58
Isaac-Sim (equiv.)	10.20

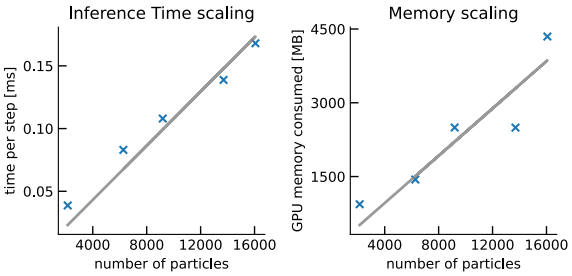

Figure 8: Inference time and memory consumption plotted against the number of particles for our model.

still slower than Isaac-sim. This gap arises from CPU-bound neighborhood computations, a known limitation in these GNN-based methods, as noted in the GNS (supplementary).

For the overall system, our method scales linearly with the number of particles $\mathcal{O}(N)$. We tested this on the stirring environment by varying the number of particles between 2k and 16k particles and report the memory consumption and inference time per step in Figure 8. The results confirm that our model scales linearly with the number of particles.

5 Discussion

We presented a unified GNN-based framework for learning particle-based fluid dynamics under rigid body manipulations. The key insights driving our approach are (1) separating liquid-liquid and liquid-object interactions, (2) using accurate surface-based collision handling for liquid-object interactions, (3) representing rigid bodies and fluid particles with separate node sets, making the architecture modular and extensible. These design choices enable our model to simulate fluid behavior in complex environments, setting it apart from prior work.

While promising, our current framework is limited in its ability to simulate fluids with diverse physical properties, such as viscosity. Future work will focus on incorporating richer representations of material properties within the network architecture. Additionally, our framework is currently designed exclusively for learning fluid dynamics and, as such, cannot handle rigid body dynamics when interacting with particles. For instance, it cannot simulate a scenario where flowing water displaces or carries a rigid object. However, given our modular graph definition, it is possible to extend our framework to handle this two-way coupling. By introducing trainable non-kinematic rigid-body nodes ($\mathcal{V}^R$) alongside liquid nodes ($\mathcal{V}^L$), and defining explicit liquid–object ($\mathcal{E}^{LO}$) and object–object ($\mathcal{E}^{OO}$) edges for bidirectional interactions, the framework could learn coupled fluid–rigid dynamics. We consider this a natural extension for future work.

Acknowledgments

This research was supported by the European Research Council (ERC; Consolidator Award 'ACTOR'-project number ERC-CoG-101045783), by the Hessian Ministry of Higher Education, Research, Science and the Arts with its LOEWE research priority program 'WhiteBox', and by 'The Adaptive Mind', funded by the Excellence Program of the Hessian Ministry of Higher Education, Science, Research and Art.

References

[1] Christopher J Bates, Ilker Yildirim, Joshua B Tenenbaum, and Peter Battaglia. Modeling human intuitions about liquid flow with particle-based simulation. *PLoS computational biology*, 15(7): e1007210, 2019. 1

[2] Peter W Battaglia, Jessica B Hamrick, and Joshua B Tenenbaum. Simulation as an engine of physical scene understanding. *Proceedings of the National Academy of Sciences*, 110(45): 18327–18332, 2013. 1

[3] Siyu He, Yin Li, Yu Feng, Shirley Ho, Siamak Ravanbakhsh, Wei Chen, and Barnabás Póczos. Learning to predict the cosmological structure formation. *Proceedings of the National Academy of Sciences*, 116(28):13825–13832, 2019. 1

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

[5] L'ubor Ladickỳ, SoHyeon Jeong, Barbara Solenthaler, Marc Pollefeys, and Markus Gross. Data-driven fluid simulations using regression forests. *ACM Transactions on Graphics (TOG)*, 34(6):1–9, 2015. 1

[6] Adithya Challapalli, Dhrumil Patel, and Gouqiang Li. Inverse machine learning framework for optimizing lightweight metamaterials. *Materials & Design*, 208:109937, 2021. 1

[7] Xingyu Lin, Zhiao Huang, Yunzhu Li, Joshua B Tenenbaum, David Held, and Chuang Gan. Diffskill: Skill abstraction from differentiable physics for deformable object manipulations with tools. *arXiv preprint arXiv:2203.17275*, 2022. 1

[8] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008. 1

[9] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018. 1, 2

[10] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks. In *International conference on machine learning*, pages 8459–8468. PMLR, 2020. 1, 2, 3, 4, 6, 14

[11] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W Battaglia. Learning mesh-based simulation with graph networks. *arXiv preprint arXiv:2010.03409*, 2020. 3, 6

[12] Kelsey R Allen, Yulia Rubanova, Tatiana Lopez-Guevara, William Whitney, Alvaro Sanchez-Gonzalez, Peter Battaglia, and Tobias Pfaff. Learning rigid dynamics with face interaction graph networks. *arXiv preprint arXiv:2212.03574*, 2022. 3, 4

[13] Yulia Rubanova, Tatiana Lopez-Guevara, Kelsey R Allen, William F Whitney, Kimberly Stachenfeld, and Tobias Pfaff. Learning rigid-body simulators over implicit shapes for large-scale scenes and vision. *arXiv preprint arXiv:2405.14045*, 2024. 3, 4

[14] Arjun Mani, Ishaan Preetam Chandratreya, Elliot Creager, Carl Vondrick, and Richard Zemel. Surfsup: Learning fluid simulation for novel surfaces. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14225–14235, 2023. 1, 3, 4

[15] Yunzhu Li, Jiajun Wu, Russ Tedrake, Joshua B Tenenbaum, and Antonio Torralba. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. *arXiv preprint arXiv:1810.01566*, 2018. 1, 2, 3

[16] Kelsey Allen, Tatiana Lopez-Guevara, Kimberly L Stachenfeld, Alvaro Sanchez Gonzalez, Peter Battaglia, Jessica B Hamrick, and Tobias Pfaff. Inverse design for fluid-structure interactions using graph network simulators. *Advances in Neural Information Processing Systems*, 35: 13759–13774, 2022. 2, 3

[17] Rhys Newbury, Jack Collins, Kerry He, Jiahe Pan, Ingmar Posner, David Howard, and Akansel Cosgun. A review of differentiable simulators. *IEEE Access*, 2024. 2

[18] Erwin Coumans. Bullet physics simulation. In *ACM SIGGRAPH 2015 Courses*, page 1. 2015. 2

[19] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012. 2

[20] Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Frédo Durand. Difftaichi: Differentiable programming for physical simulation. *arXiv preprint arXiv:1910.00935*, 2019. 2

[21] C Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax–a differentiable physics engine for large scale rigid body simulation. *arXiv preprint arXiv:2106.13281*, 2021.

[22] Samuel S. Schoenholz and Ekin D. Cubuk. Jax m.d. a framework for differentiable physics. In *Advances in Neural Information Processing Systems*, volume 33. Curran Associates, Inc., 2020.

[23] Genesis Authors. Genesis: A Universal and Generative Physics Engine for Robotics and Beyond. https://github.com/Genesis-Embodied-AI/Genesis, December 2024. URL https://github.com/Genesis-Embodied-AI/Genesis. 2

[24] Philipp Holl, Vladlen Koltun, Kiwon Um, and Nils Thuerey. phiflow: A differentiable pde solving framework for deep learning via physical simulations. In *NeurIPS workshop*, volume 2, 2020. 2

[25] Deniz A. Bezgin, Aaron B. Buhendwa, and Nikolaus A. Adams. Jax-fluids: A fully-differentiable high-order computational fluid dynamics solver for compressible two-phase flows. *Computer Physics Communications*, 282:108527, 1 2023. ISSN 00104655. doi: 10.1016/j.cpc.2022.108527. URL https://linkinghub.elsevier.com/retrieve/pii/S0010465522002466. 2

[26] Zhou Xian, Bo Zhu, Zhenjia Xu, Hsiao-Yu Tung, Antonio Torralba, Katerina Fragkiadaki, and Chuang Gan. Fluidlab: A differentiable environment for benchmarking complex fluid manipulation. In *International Conference on Learning Representations*, 2023. 2

[27] Connor Schenck and Dieter Fox. Spnets: Differentiable fluid dynamics for deep neural networks. In *Conference on Robot Learning*, pages 317–335. PMLR, 2018. 2, 9

[28] Benjamin Ummenhofer, Lukas Prantl, Nils Thuerey, and Vladlen Koltun. Lagrangian fluid simulation with continuous convolutions. In *International Conference on Learning Representations*, 2020.

[29] Yifei Li, Yuchen Sun, Pingchuan Ma, Eftychios Sifakis, Tao Du, Bo Zhu, and Wojciech Matusik. Neuralfluid: Nueral fluidic system design and control with differentiable simulation. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=LLsOmvJbBm. 2

[30] Kelsey R Allen, Tatiana Lopez Guevara, Yulia Rubanova, Kim Stachenfeld, Alvaro Sanchez-Gonzalez, Peter Battaglia, and Tobias Pfaff. Graph network simulators can learn discontinuous, rigid contact dynamics. In *Conference on Robot Learning*, pages 1157–1167. PMLR, 2023. 2, 3

[31] Andreas Mayr, Sebastian Lehner, Arno Mayrhofer, Christoph Kloss, Sepp Hochreiter, and Johannes Brandstetter. Boundary graph neural networks for 3d simulations. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(8):9099–9107, Jun. 2023. doi: 10.1609/aaai.v37i8.26092. URL https://ojs.aaai.org/index.php/AAAI/article/view/26092.

[32] Wenbing Huang, Jiaqi Han, Yu Rong, Tingyang Xu, Fuchun Sun, and Junzhou Huang. Equivariant graph mechanics networks with constraints. *arXiv preprint arXiv:2203.06442*, 2022. 3

[33] Vıctor Garcia Satorras, Emiel Hoogeboom, and Max Welling. E (n) equivariant graph neural networks. In *International conference on machine learning*, pages 9323–9332. PMLR, 2021.

[34] Jiaqi Han, Wenbing Huang, Hengbo Ma, Jiachen Li, Josh Tenenbaum, and Chuang Gan. Learning physical dynamics with subequivariant graph neural networks. *Advances in Neural Information Processing Systems*, 35:26256–26268, 2022. 2, 3

[35] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017. 3

[36] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. *Advances in neural information processing systems*, 29, 2016. 3

[37] James H Clark. Hierarchical geometric models for visible surface algorithms. *Communications of the ACM*, 19(10):547–554, 1976. 4

[38] NVIDIA. "Isaac Sim - Robotics Simulation and Synthetic Data Generation". `https://developer.nvidia.com/isaac-sim`. accessed on 2024-31-05. 5

[39] Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. Position based dynamics. *Journal of Visual Communication and Image Representation*, 18(2):109–118, 2007. 5

[40] Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 9, 14

[41] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. 14

A Appendix

A.1 Architecture and training

All the MLPs in our architecture contain 2 hidden layers of 128 units each with LayerNorm [41] and an output layer of 128 units. Only in the case of the decoder, the output layer has 3 units to decode the 3D acceleration of particles. We chose $P = 10$ as the number of processor blocks.

It has been shown that perturbing the inputs with noise makes the GNN models more robust to noisy inputs [10]. This is particularly helpful when generating rollouts where the network is fed with its own noisy, previous predictions as input. We choose a noise value $\mathcal{N}(0, \sigma = 0.00067)$. Additionally, all the inputs and outputs were normalised.

We use Adam [40] to optimize the $L2$ loss on one-step acceleration predictions of the liquid particles with an exponentially decaying learning rate from 1e-4 to 1e-6. The predicted particle accelerations are then integrated to produce the next-step positions using the finite-difference method.

The models were trained on a single Nvidia 4090 GPU up to a maximum of $2M$ gradient steps with a batch size of 20. The training period lasted approximately 10 days. The source code and visualizations are available here.[1]

A.2 Baseline model details

GNS baseline has a graph of the form $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, while MeshGraphNet (MGN and MGN*) models use a multigraph of the form $\mathcal{G} = (\mathcal{V}, \mathcal{E}^L, \mathcal{E}^M)$. For all three baselines, the node set $\mathcal{V}$ contains both liquid particles and rigid bodies. To represent a rigid body, its mesh vertices are used as nodes in a graph. To distinguish between particle nodes and mesh nodes, one-hot encoding is incorporated into the node features. Unlike our model, GNS does not explicitly represent the floor as a separate object; instead, the distance from each particle to the floor is included as part of each particle's node features. We use the same floor representation for MeshGraphNets as well.

For the edge sets in MeshGraphNet baselines, $\mathcal{E}^L$ captures liquid–liquid interactions (within neighborhood radius r_l) and corresponds to $\mathcal{E}^L$ from our model. The $\mathcal{E}^M$ edge set represents liquid–object interactions, where an edge is connected between a liquid particle and a mesh vertex node if they are within the collision radius r_{ol}. The MGN* model additionally includes edges between surface nodes of the rigid bodies in the $\mathcal{E}^M$ set. In the GNS baseline, $\mathcal{E}$ edge set contains both liquid-liquid and liquid-object edges.

The architecture of the individual MLPs in the baseline models, as well as the training procedure, normalisation, and noise injection, remains the same as in our model training.

Since collision with particles is handled at the mesh vertex level in the baseline models, a low vertex count negatively impacts performance. To ensure a fair comparison, we oversampled all objects used in the baseline evaluations. Table 3 summarizes the number of nodes and edges used in both our models and the baseline models.

Table 3: Number of vertices and edges of each object used in our models and baselines.

Object	Our models		Baseline models	
	# Nodes	# Edges	# Nodes	# Edges
Vase jug	776	3264	1684	8640
Beaker jug	540	3264	1994	10728
WineBottle jug	713	2880	2261	10944
Martini cup	491	1800	909	4080
Wine cup	903	4032	2780	14208

Connectivity radius r_l was chosen such that each liquid particle interacts with approximately 10-20 particles. To determine this value, we computed the average number of neighbors for each particle in a subset of our main dataset (Martini cup + Vase jug) across various values of r_l. Figure 9(a) illustrates these results, based on which we selected $r_l = 0.12$.

[1] https://github.com/RothkopfLab/fluid-manip-gnn

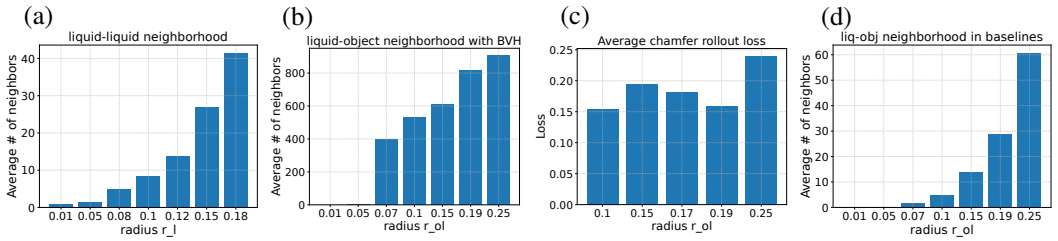

Figure 9: Effects of varying r_l and r_{ol} on edge connectivity and model performance.

r_{ol} defines the neighborhood radius for liquid-object interactions. In Isaac Sim, a small separation is maintained between liquid particles and object surfaces to enforce boundary conditions during collision handling. As a result, particles do not interact with objects exactly at their surface. To determine the minimum value of r_{ol} at which the object surface falls within the particle interaction range, we used the same dataset as for r_l and ran our BVH-based collision algorithm. Figure 9(b) shows the average number of particles in contact with object surfaces for different values of r_{ol}. For values of r_{ol} below 0.05, the object surface lies outside the liquid-object interaction radius.

To identify the appropriate value for r_{ol}, we trained our *Full-model* across different r_{ol} values and observed that performance remained largely consistent between 0.1 and 0.2 (see Figure 9(c)). For the baseline models, which use node-based collision detection, we ran a similar test as for r_l (Figure 9(d)). We ultimately selected $r_{ol} = 0.19$ for all experiments, as it ensures that each liquid particle is connected to approximately 20-30 object mesh nodes in the baseline models, allowing a fairer comparison with our models.

A.3 Failure cases

We tested our model on two scenarios where it fails or performs suboptimally.

Tunneling occurs when object velocities are significantly higher than those seen during training. GNN-based models operate on a fixed time step (dt=1), and hence, if the object pose changes significantly between steps, the object can jump across particles between steps, resulting in missed collisions ("tunneling"). We tested this on the Martini-Vase environment (identical to our training environment) by making the Vase jug rotate rapidly. We observe the tunneling effect at the beginning of the simulation (see video in our project repository). In Classical simulators, this tunneling effect is typically mitigated by using sub-stepping and continuous collision detection (CCD) methods. In learned GNN-based simulators, it is not clear how such remedies can be integrated.

Another case where our model fails is when the liquid motion is caused by shear stress and centrifugal effects. For example, when a cylindrical container with liquid inside spins about its axis, the liquid also rotates along with the container. This behavior cannot be captured by our model, as our liquid–object connectivity is distance-based. When the container rotates around its axis, particle–surface distances remain constant, so the model does not perceive tangential motion. To test this, we created a cement-mixer environment in which liquid particles are placed inside a rotating drum. Our model fails to reproduce the expected rotation of the liquid (see video in our project repository).

A.4 Additional Figures

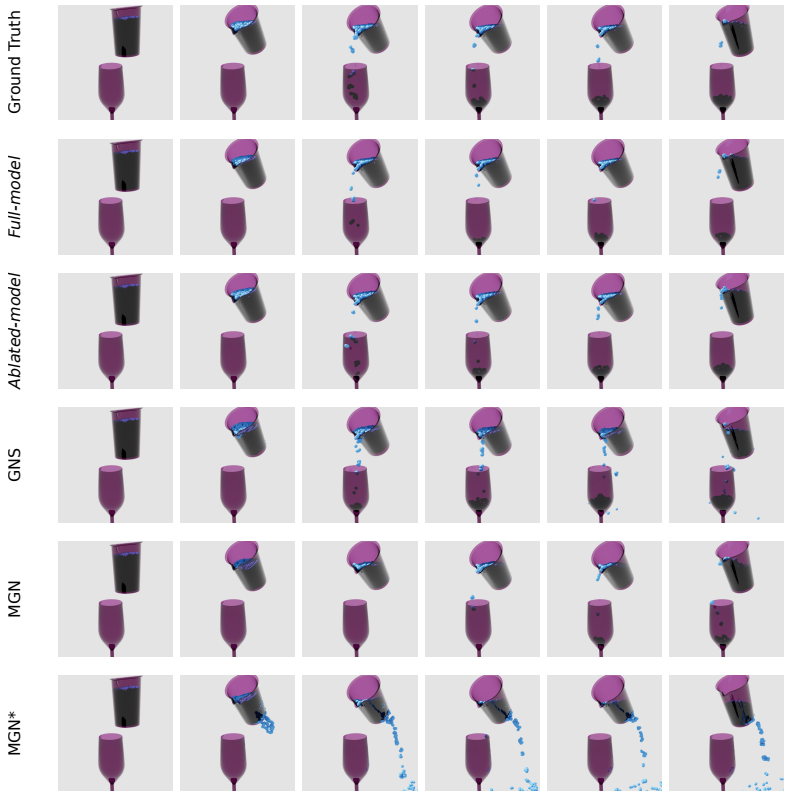

Figure 10: Visualisation of rollouts of all models on a Wine+Beaker example

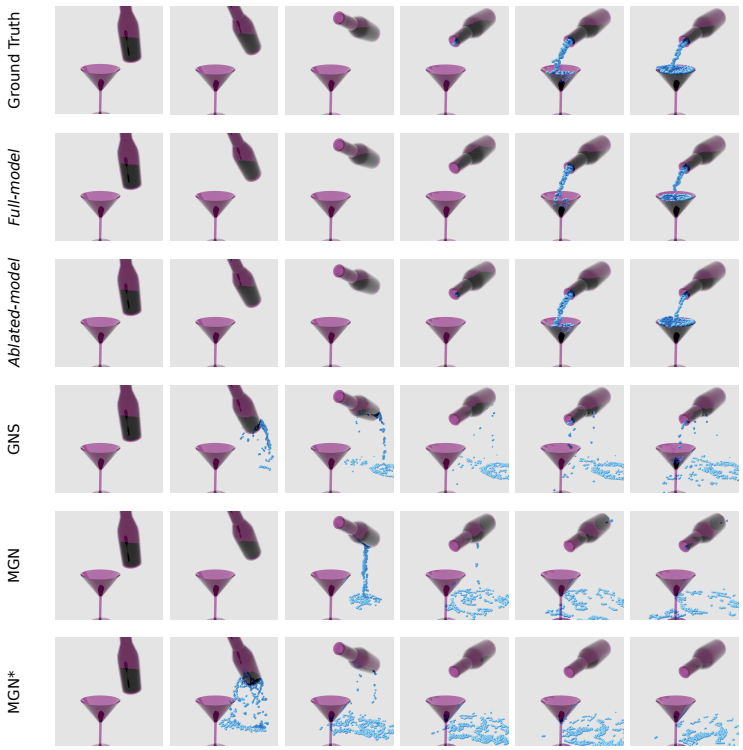

Figure 11: Visualisation of rollouts of all models on a Martini+Bottle example

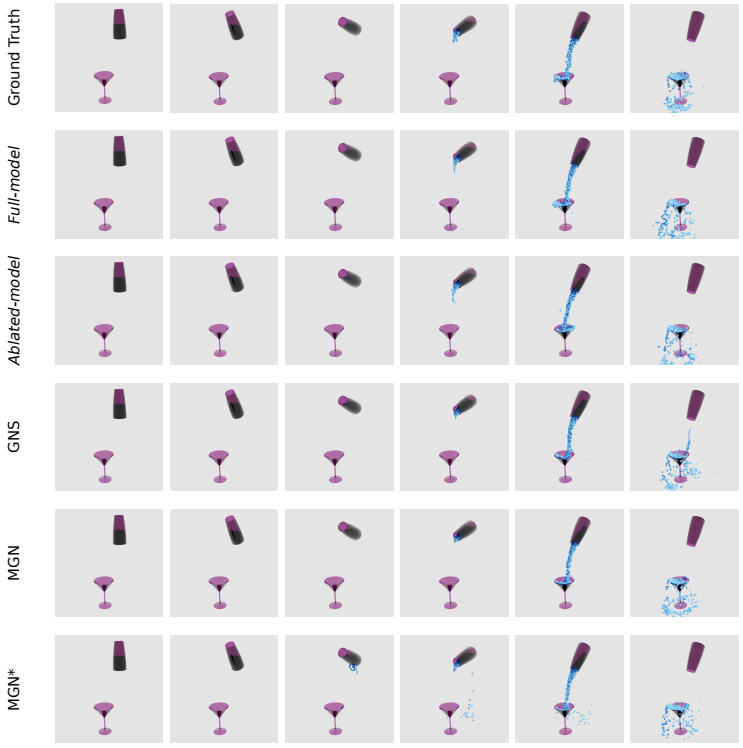

Figure 12: Visualisation of rollouts of all models on a Martini+Vase example