GROUNDING GRAPH NETWORK SIMULATORS USING PHYSICAL SENSOR OBSERVATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Physical simulations that accurately model reality are crucial for many engineering disciplines such as mechanical engineering and robotic motion planning. In recent years, learned Graph Network Simulators produced accurate mesh-based simulations while requiring only a fraction of the computational cost of traditional simulators. As these predictors have to simulate complex physical systems from only an initial state, they exhibit a high error accumulation for long-term predictions. In this work, we integrate sensory information to *ground* Graph Network Simulators on real world observations in the form of point clouds. The resulting model allows for accurate predictions over longer time horizons, even under uncertainties in the simulation, such as unknown material properties.

1 INTRODUCTION

Mesh-based simulation of complex physical systems lies at the heart of many fields in numerical science and engineering (Rao, 2017; Sabat & Kundu, 2021). Applications include structural mechanics (Zienkiewicz & Taylor, 2005; Stanova et al., 2015), electromagnetics (Jin, 2015; Xiao et al., 2022) and fluid dynamics (Zawawi et al., 2018; Long et al., 2021). Recent advances in deep learning have led to a number of learned methods for application in physics. These include Physical Reasoning (Battaglia et al., 2016; Mrowca et al., 2018) and learned simulators using data from a ground truth simulator. Here there are Convoluational Neural Network (CNN)-based methods, e.g., for fluid flow (Tompson et al., 2017; Chu & Thuerey, 2017; Ummenhofer et al., 2020; Kim et al., 2019; Xie et al., 2018) or aerodynamic flow (Guo et al., 2016; Zhang et al., 2018; Bhatnagar et al., 2019). Graph Network Simulators (GNSs) (Sanchez-Gonzalez et al., 2018; 2020; Pfaff et al., 2021) use Graph Neural Networks (GNNs) (Scarselli et al., 2009) to learn to predict the dynamics of a system from raw data by encoding the system state as a graph. They are widely used in particle-based simulations (Li et al., 2019; Sanchez-Gonzalez et al., 2020) and mesh-based simulations (Weng et al., 2021; Han et al., 2022; Fortunato et al., 2022; Allen et al., 2022). Yet, they assume perfect knowledge of the initial system state, making them ill-suited for applications like model-based control (Camacho & Alba, 2013; Schwenzer et al., 2021) and model-based reinforcement learning (Polydoros & Nalpantidis, 2017; Moerland et al., 2020). Simulation directly from observation comes with the benefit of requiring less expert knowledge for the design of the simulator and better applicability to real-world scenarios. Here, often point clouds are used as observations (Watters et al., 2017; Wang et al., 2019; Gomes et al., 2021; Park et al., 2021). Another recent work trains a simulator to align its simulations with real-world point cloud observations of deformable objects (Sundaresan et al., 2022).

In this work, we present Grounding Graph Network Simulators (GGNSs), a new class of GNS that can process sensory information as input to *ground* predictions in the observations of the scene. As the sensory information is not always available, our architecture is trained with imputed point cloud data. For inference, the model is used iteratively predicting the next system state, using point clouds whenever available to improve the simulation quality. As a practical example, consider a robot grasping a deformable object. For optimal planning of the grasp, the robot needs to model the state of the deformable object over time and predict the influence of interactions between object and gripper. Once the robot starts deforming the object, it may easily observe the deformations in the form of e.g., point clouds, which then be integrated into the state prediction. An example is given in Figure 1.

We evaluate GGNS on a suite of 2d and 3d deformation prediction tasks created in the Simulation Open Framework Architecture (SOFA) (Faure et al., 2012). Comparing our approach to an existing

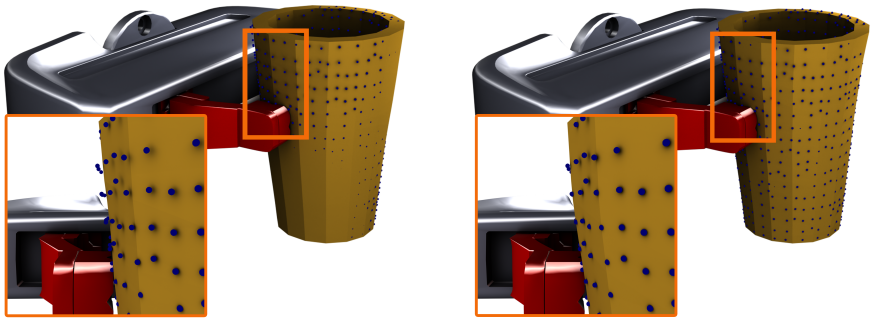


Figure 1: A robot's end-effector (grey, red) grasps a 3-dimensional deformable cavity. The robot maintains an internal simulated prediction of the cavity (orange) for two consecutive simulation steps (left, right). The true cavity state can infrequently be observed from point cloud data (blue), which the model can use to correct its prediction. We repeat the point cloud from the earlier simulation step in both images for clarity.

GNS (Pfaff et al., 2021), we find that adding sensory information in the form of point clouds to our model greatly improves the simulation quality for all tasks.

2 Grounding Graph Network Simulator

Let $\mathcal{G} = (\mathbf{V}, \mathbf{E}, \mathbf{X_V}, \mathbf{X_E})$ be a directed graph with nodes $\mathbf{V}$, edges $\mathbf{E} \subseteq \mathbf{V} \times \mathbf{V}$, node features $\mathbf{X_V} : \mathbf{V} \to \mathbb{R}^{d_\mathbf{V}}$ of dimension $d_\mathbf{V}$ and edge features $\mathbf{X_E} : \mathbf{E} \to \mathbb{R}^{d_\mathbf{E}}$ of dimension $d_\mathbf{E}$. A Message Passing Network (MPN) (Gilmer et al., 2017; Battaglia et al., 2018) is a GNN consisting of L *Message Passing Blocks* that receives a graph $\mathcal{G}$ as input and outputs a learned representation for nodes and edges. Each block l computes updated features for all nodes $v \in \mathbf{V}$ and edges $e \in \mathbf{E}$ as

$$\mathbf{x}_e^{l+1} = f_\mathbf{E}^l(\mathbf{x}_v^l, \mathbf{x}_u^l, \mathbf{x}_e^l), \text{ with } e = (u, v) \quad \text{and} \quad \mathbf{x}_v^{l+1} = f_\mathbf{V}^l(\mathbf{x}_v^l, \bigoplus_{\{e=(v,u)\in\mathbf{E}\}} \mathbf{x}_e^{l+1}),$$

where $\mathbf{x}_v^0$ and $\mathbf{x}_e^0$ are embeddings of the node and edge features of $\mathcal{G}$, $\oplus$ is a permutation-invariant aggregation function and each $f_\cdot^l$ is a learned function, e.g., a Multilayer Perceptron (MLP).

Graph Network Simulators. GNSs first encode the system state $\mathcal{S}$ in a graph $\mathcal{G}$. For mesh-based simulations, the graph is naturally given by the mesh $\mathcal{M}$. For the features, encoding purely *relative* properties such as relative distances and velocities per edge rather than absolute positions per node have been shown to greatly improve generalization (Sanchez-Gonzalez et al., 2020). The encoded graph $\mathcal{G}$ is then used as input for a learned MPN, which computes latent representations x_v^L for each node $v \in \mathbf{V}$. These latent representations are then interpreted as derivatives of dynamic quantities, which a simple forward Euler integrator uses to compute the updated system state $\mathcal{S}'$. GNSs are trained on a node-wise next-step Mean Squared Error (MSE) objective, i.e., they minimize the 1-step prediction error of predicted system state to a given ground truth. During inference, full rollouts can be generated by iteratively repeating the above-mentioned steps, using the updated dynamics of one step as the input for the next. Here, the model does not predict the movement of fixed entities such as e.g., a collider, which is instead assumed to be known. Due to this iterative dependence on previous outputs, the model is prone to error accumulation. To alleviate this problem, noise is applied to the dynamic variables of the system for each training step (Pfaff et al., 2021).

Grounding Graph Network Simulator. Our approach extends the existing GNS framework to naturally and efficiently integrate auxiliary point cloud data whenever available. This auxiliary information *grounds* the predictions of the model in an observation of the true system state. Figure 2 illustrates an overview of our approach. More details on the GNN part of the method is found in Appendix A. In order to utilize point-based data in addition to meshes we first have to transfer both into a common graph. Following previous work (Sanchez-Gonzalez et al., 2020), we do this by creating a neighborhood graph based on spatial proximity. The different node and edge types are one-hot encoded into their respective features to allow the model to differentiate between them.

For most realistic applications, point clouds are typically available at a much lower frequency than the simulation. We adapt our model to this constraint by employing an imputation-based training scheme, i.e., for each time step the model receives point clouds only with a probability of $p = 0.5$. Intuitively,

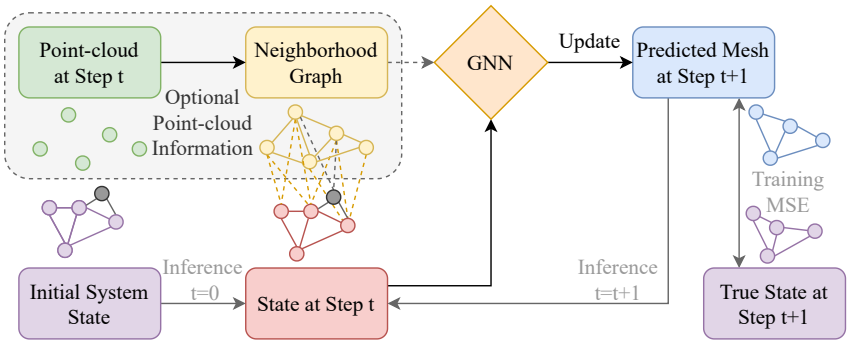


Figure 2: Schematic of GGNS . Given a system state $\mathcal{S}_t$ (red), boundary conditions (gray) and optional point cloud observations (dashed box), a GNN (orange) predicts how the system $\mathcal{S}_{t+1}$ will look like at the next step (blue). When provided, the point cloud (green) is transformed into a neighborhood graph (yellow) in the same coordinate system as the mesh, connecting each point in the point cloud to the nearest mesh nodes. During inference, the model iteratively predicts updates from a potentially incomplete initial system state (purple), using additional point cloud observations when available.

this allows each system node $v \in \mathbf{V}$ to utilize the additional information of close-by points of a point cloud when available, while forcing it to still make meaningful predictions if no additional information is available. An example can be seen in Figure 1. Here, $\mathcal{S}$ consists of a predicted mesh and a gripper. The mismatch between the predicted mesh and the point cloud of the true object indicates the prediction error. The model uses this information to correct the current state estimate.

3 Experiments

We evaluate GGNS on complex 2d and 3d object deformation prediction tasks, where the true system state is given by a triangular surface mesh of a deformable object with rigid boundary conditions and a triangular surface mesh of a rigid collider. The point clouds are generated by raycasting using virtual cameras arranged around the scene. For more details, see the Appendix B. We assume that, while the initial mesh of the object is known, its material properties are not. We model these unknown properties via the Poisson's ratio (Lim, 2015) $-1 < \nu < 0.5$, which is a scalar value describing the ratio of contraction ($\nu < 0$) or expansion ($\nu > 0$) under compression (Mazaev et al., 2020). An overview of the training and network hyperparameters can be found in Appendix E.

Evaluation Metrics. We evaluate the performance of all trained models on 10 different seeds per experiment. We report the means and standard deviations of the different runs, where, for each run, we average the results over all available steps of a trajectory and over all trajectories in the test set. For all experiments, we report the *full rollout loss*, where the model starts with the initial state $\mathcal{S}_0$ and predicts the states up to a final state $\mathcal{S}_T$. Here, we provide a point cloud to the model every $k \geq 1$ steps and resort to mesh-only prediction otherwise.

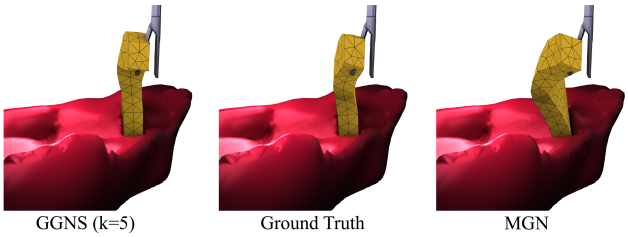
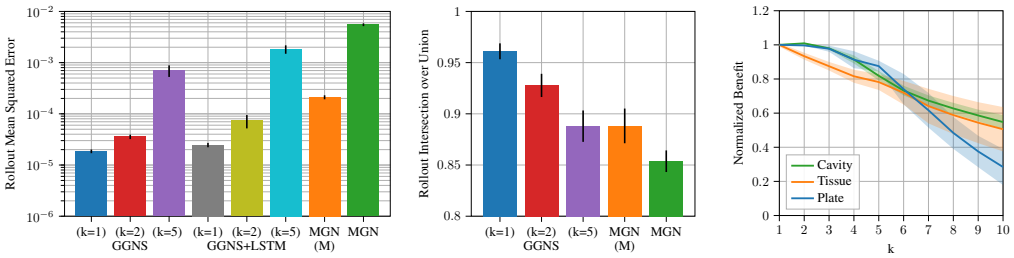


Figure 3: Test trajectory of the Tissue Manipulation task at time step $t = 70$ for GGNS (left), the ground truth simulation (middle) and MeshGraphNet (MGN) (right). While MGN exhibits a large prediction error, GGNS is able to utilize the point cloud information to stay close to the ground truth.

Baselines. We compare to MGN, a state-of-the-art GNS, which utilizes additional *world edges* between close-by mesh nodes, but does not incorporate point cloud observations. We adopt this explicit representation of edge types for the MGN baseline and experiment with it for GGNS in Appendix D. Additionally, we evaluate a variant of MGN that has access to the underlying Poisson's ratio υ as a node feature, called MGN (M). This additional information leads to a deterministic ground truth simulation w.r.t. the initial system state, and upper bounds the performance of MGN. We also compare to GGNS+LSTM, which uses an LSTM (Hochreiter & Schmidhuber, 1997) layer on the node output features of the GNN to explicitly include recurrency into the model.



(a) Deformable Plate Task (b) IoU without initial Mesh (c) Grounding Frequencies

Figure 4: a Rollout MSE of GGNS, GGNS+LSTM and MGN baselines evaluated on the Deformable Plate test set using point clouds every k-th time step. In (b) the initial mesh is created from the initial the point cloud. The achieved Intersection over Unions (IoUs) are lower compared to using the ground truth mesh, but GGNS ($k \leq 2$) still outperforms all baselines. (c) Normalized benefit of using a point cloud every k-th time step for all three tasks. MGN performance corresponds to 0.0.

Datasets. For the *Deformable Plate* dataset, we consider a family of 2-dimensional trapezoids that are deformed by a circular collider with varying size and starting position and constant velocity. We use a total of $675/135/135$ trajectories for our training, validation and test sets. Each trajectory consist of $T = 50$ time steps. For the *Tissue Manipulation* task, we simulate a robot-assisted surgery scenario where a piece of tissue is deformed by a solid gripper. We vary the direction of the gripper's motion and its gripping position on the tissue. In the *Cavity Grasping* dataset, a simulated robot gripper grasps and deforms cone-shaped cavities with random radii from different positions. For both $3d$ tasks, $600/120/120$ trajectories are used, each of which is rolled out for $T = 100$ time steps.

4 Results

For all tasks, we find that GGNS can use the point cloud information to produce high quality rollouts that closely match the true system states. Figure 3 shows a qualitative example. Additional visualizations for all tasks and methods can be found in Appendix C. The performance of GGNS for different model hyperparameters, as well as for noisy and partially observable point clouds is shown in Appendix D. Similar to MGN, GGNS is robust to most parameter choices. For the $2d$ data of the Deformable Plate task, we also compare our imputation model to the GGNS+LSTM approach, which can use the recurrence of LSTMs to pass information over time. Figure 4a shows that GGNS outperforms this alternative approach for each k. Additionally, GGNS trains significantly faster, likely due to the additional complexity of training the recurrent model. Using the IoU, we can compare objects across different mesh representations. Figure 4b shows that GGNS still produces accurate rollouts for similar-sized meshes that are directly generated from the initial point cloud. In this case, the method avoids the dependence on *any* simulation data, which marks an important step towards using GNSs on real world data. Figure 4c shows the normalized performance of GGNS for grounding frequencies $k \in \{1..10\}$ across all three tasks. Here, 1.0 corresponds to the performance for $k = 1$, and 0.0 to the performance of MGN. For all tasks there is a clear advantage in utilizing the point cloud information, and performance increases with the point cloud frequency.

5 Conclusion

We propose Grounding Graph Network Simulator (GGNS), an extension of the Graph Network Simulator framework that can utilize auxiliary observations to accurately simulate complex dynamics from incomplete initial system states. Utilizing a neighborhood graph from point cloud information and an imputation-based training scheme, our model is able to *ground* its prediction in observations of the true system state. We show experimentally that this leads to high quality simulations in complex $2d$ and $3d$ deformation tasks, outperforming existing approaches. In future work, we will extend GGNSs to explicitly model uncertainty and maintain a belief over the latent variables of the system, e.g., by employing a Kalman filter in a learned latent space (Becker et al., 2019). Finally, we will employ our model for Model Predictive Control and Model-based Reinforcement Learning.

REFERENCES

Kelsey R Allen, Tatiana Lopez Guevara, Yulia Rubanova, Kimberly Stachenfeld, Alvaro Sanchez-Gonzalez, Peter Battaglia, and Tobias Pfaff. Graph network simulators can learn discontinuous, rigid contact dynamics. *Conference on Robot Learning (CoRL).*, 2022.

Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, and koray kavukcuoglu. Interaction networks for learning about objects, relations and physics. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper/2016/file/3147da8ab4a0437c15ef51a5cc7f2dc4-Paper.pdf.

Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinícius Flores Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Çaglar Gülçehre, H. Francis Song, Andrew J. Ballard, Justin Gilmer, George E. Dahl, Ashish Vaswani, Kelsey R. Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matthew Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks. *CoRR*, abs/1806.01261, 2018. URL http://arxiv.org/abs/1806.01261.

Philipp Becker, Harit Pandya, Gregor Gebhardt, Cheng Zhao, C James Taylor, and Gerhard Neumann. Recurrent kalman networks: Factorized inference in high-dimensional deep feature spaces. In *International Conference on Machine Learning*, pp. 544–552, 2019.

Saakaar Bhatnagar, Yaser Afshar, Shaowu Pan, Karthik Duraisamy, and Shailendra Kaushik. Prediction of aerodynamic flow fields using convolutional neural networks. *Computational Mechanics*, 64(2):525–545, jun 2019. doi: 10.1007/s00466-019-01740-0. URL https://doi.org/10.1007%2Fs00466-019-01740-0.

Eduardo F Camacho and Carlos Bordons Alba. *Model predictive control*. Springer science & business media, 2013.

Mengyu Chu and Nils Thuerey. Data-driven synthesis of smoke flows with cnn-based feature descriptors. *ACM Trans. Graph.*, 36(4), jul 2017. ISSN 0730-0301. doi: 10.1145/3072959.3073643. URL https://doi.org/10.1145/3072959.3073643.

François Faure, Christian Duriez, Hervé Delingette, Jérémie Allard, Benjamin Gilles, Stéphanie Marchesseau, Hugo Talbot, Hadrien Courtecuisse, Guillaume Bousquet, Igor Peterlik, and Stéphane Cotin. SOFA: A Multi-Model Framework for Interactive Physical Simulation. In Yohan Payan (ed.), *Soft Tissue Biomechanical Modeling for Computer Assisted Surgery*, volume 11 of *Studies in Mechanobiology, Tissue Engineering and Biomaterials*, pp. 283–321. Springer, June 2012. doi: 10.1007/8415_2012_125. URL https://hal.inria.fr/hal-00681539.

Meire Fortunato, Tobias Pfaff, Peter Wirnsberger, Alexander Pritzel, and Peter Battaglia. Multiscale meshgraphnets. In *ICML 2022 2nd AI for Science Workshop*, 2022. URL https://openreview.net/forum?id=G3TRIsmMhhf.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1263–1272. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/gilmer17a.html.

Pedro Gomes, Silvia Rossi, and Laura Toni. Spatio-temporal graph-rnn for point cloud prediction. In *2021 IEEE International Conference on Image Processing (ICIP)*, pp. 3428–3432, 2021. doi: 10.1109/ICIP42928.2021.9506084.

Xiaoxiao Guo, Wei Li, and Francesco Iorio. Convolutional neural networks for steady flow approximation. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pp. 481–490, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450342322. doi: 10.1145/2939672.2939738. URL https://doi.org/10.1145/2939672.2939738.

Xu Han, Han Gao, Tobias Pffaf, Jian-Xun Wang, and Li-Ping Liu. Predicting physics in mesh-reduced space with temporal attention. *CoRR*, abs/2201.09113, 2022. URL https://arxiv.org/abs/2201.09113.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997. doi: 10.1162/neco.1997.9.8.1735.

Jian-Ming Jin. *The finite element method in electromagnetics*. John Wiley & Sons, 2015.

Byungsoo Kim, Vinicius C. Azevedo, Nils Thuerey, Theodore Kim, Markus Gross, and Barbara Solenthaler. Deep Fluids: A Generative Network for Parameterized Fluid Simulations. *Computer Graphics Forum (Proc. Eurographics)*, 38(2), 2019.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

Yunzhu Li, Jiajun Wu, Russ Tedrake, Joshua B. Tenenbaum, and Antonio Torralba. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rJgbSn09Ym.

Teik-Cheng Lim. *Auxetic Materials and Structures*. Springer Singapore, 01 2015. ISBN 978-981-287-274-6. doi: 10.1007/978-981-287-275-3. URL https://doi.org/10.1007/978-981-287-275-3.

Ting Long, Can Huang, Dean Hu, and Moubin Liu. Coupling edge-based smoothed finite element method with smoothed particle hydrodynamics for fluid structure interaction problems. *Ocean Engineering*, 225:108772, 2021.

A V Mazaev, O Ajeneza, and M V Shitikova. Auxetics materials: classification, mechanical properties and applications. *IOP Conference Series: Materials Science and Engineering*, 747 (1):012008, jan 2020. doi: 10.1088/1757-899x/747/1/012008. URL https://doi.org/10.1088/1757-899x/747/1/012008.

Thomas M Moerland, Joost Broekens, and Catholijn M Jonker. Model-based reinforcement learning: A survey. *arXiv preprint arXiv:2006.16712*, 2020.

Damian Mrowca, Chengxu Zhuang, Elias Wang, Nick Haber, Li F Fei-Fei, Josh Tenenbaum, and Daniel L Yamins. Flexible neural representation for physics prediction. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/fd9dd764a6f1d73f4340d570804eacc4-Paper.pdf.

Jinhyung Park, Dohae Lee, and In-Kwon Lee. Flexible networks for learning physical dynamics of deformable objects. *CoRR*, abs/2112.03728, 2021. URL https://arxiv.org/abs/2112.03728.

Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W. Battaglia. Learning mesh-based simulation with graph networks. In *International Conference on Learning Representations*, 2021. URL https://arxiv.org/abs/2010.03409.

Athanasios S Polydoros and Lazaros Nalpantidis. Survey of model-based reinforcement learning: Applications on robotics. *Journal of Intelligent & Robotic Systems*, 86(2):153–173, 2017.

Singiresu S Rao. *The finite element method in engineering*. Butterworth-heinemann, 2017.

Lovely Sabat and Chinmay Kumar Kundu. History of finite element method: a review. *Recent Developments in Sustainable Infrastructure*, pp. 395–404, 2021.

Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. Graph networks as learnable physics engines for inference and control. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 4470–4479. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/sanchez-gonzalez18a.html.

Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks. In *Proceedings of the 37th International Conference on Machine Learning*, pp. 8459–8468. PMLR, 2020.

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, 2009. doi: 10.1109/TNN.2008.2005605.

Max Schwenzer, Muzaffer Ay, Thomas Bergs, and Dirk Abel. Review on model predictive control: An engineering perspective. *The International Journal of Advanced Manufacturing Technology*, 117(5):1327–1349, 2021.

Eva Stanova, Gabriel Fedorko, Stanislav Kmet, Vieroslav Molnar, and Michal Fabian. Finite element analysis of spiral strands with different shapes subjected to axial loads. *Advances in engineering software*, 83:45–58, 2015.

Priya Sundaresan, Rika Antonova, and Jeannette Bohg. Diffcloud: Real-to-sim from point clouds with differentiable simulation and rendering of deformable objects. *CoRR*, abs/2204.03139, 2022. doi: 10.48550/arXiv.2204.03139. URL https://doi.org/10.48550/arXiv.2204.03139.

Jonathan Tompson, Kristofer Schlachter, Pablo Sprechmann, and Ken Perlin. Accelerating eulerian fluid simulation with convolutional networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pp. 3424–3433. JMLR.org, 2017.

Benjamin Ummenhofer, Lukas Prantl, Nils Thuerey, and Vladlen Koltun. Lagrangian fluid simulation with continuous convolutions. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=B1lDoJSYDH.

Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. sanchezgonzalez2018graphdynamic graph cnn for learning on point clouds. *ACM Trans. Graph.*, 38(5), oct 2019. ISSN 0730-0301. doi: 10.1145/3326362. URL https://doi.org/10.1145/3326362.

Nicholas Watters, Daniel Zoran, Theophane Weber, Peter Battaglia, Razvan Pascanu, and Andrea Tacchetti. Visual interaction networks: Learning a physics simulator from video. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/8cbd005a556ccd4211ce43f309bc0eac-Paper.pdf.

Zehang Weng, Fabian Paus, Anastasiia Varava, Hang Yin, Tamim Asfour, and Danica Kragic. Graph-based task-specific prediction models for interactions between deformable and rigid objects. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5741–5748, 2021. doi: 10.1109/IROS51168.2021.9636660. URL https://arxiv.org/abs/2103.02932.

Longying Xiao, Gianluca Fiandaca, Bo Zhang, Esben Auken, and Anders Vest Christiansen. Fast 2.5 d and 3d inversion of transient electromagnetic surveys using the octree-based finite-element method. *Geophysics*, 87(4):E267–E277, 2022.

You Xie, Erik Franz, Mengyu Chu, and Nils Thuerey. tempoGAN: A Temporally Coherent, Volumetric GAN for Super-resolution Fluid Flow. *ACM Transactions on Graphics (TOG)*, 37(4):95, 2018.

Mohd Hafiz Zawawi, A Saleha, A Salwa, NH Hassan, Nazirul Mubin Zahari, Mohd Zakwan Ramli, and Zakaria Che Muda. A review: Fundamentals of computational fluid dynamics (cfd). In *AIP conference proceedings*, pp. 020252. AIP Publishing LLC, 2018.

Yao Zhang, Woong Je Sung, and Dimitri N. Mavris. Application of convolutional neural network to predict airfoil lift coefficient. In *2018 AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, 2018. doi: 10.2514/6.2018-1903. URL https://arc.aiaa.org/doi/abs/10.2514/6.2018-1903.

Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3d: A modern library for 3d data processing. *CoRR*, abs/1801.09847, 2018. URL http://arxiv.org/abs/1801.09847.

Olek C Zienkiewicz and Robert Leroy Taylor. *The finite element method for solid and structural mechanics*. Elsevier, 2005.

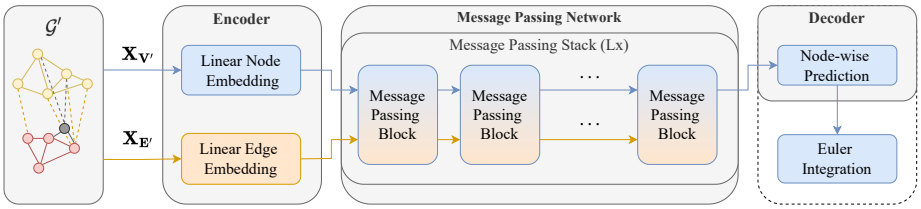


Figure 5: A detailed view of the GNN part of GGNS . Given a graph $\mathcal{G}'$, the node and edge features $\mathbf{X}_{\mathbf{V}'}$ and $\mathbf{X}_{\mathbf{E}'}$ are linearly embedded into a latent space and then updated with L Message Passing Blocks. The resulting predictions are interpreted as dynamic quantities that are used to update the system.

A Model Details

The Message Passing Network employed by GGNS is displayed in 5. As node-wise predictions we use velocities, which are Euler-integrated once to update the positions of the mesh of the deformable object.

B Environment Details

Here, we describe all key aspects, which are valid for all three environments. All datasets are simulated using SOFA and include different material properties. Therefore, we choose discrete Poisson's ratios from $\nu \in \{-0.9, 0.0, 0.49\}$ for one-third of all simulated trajectories each. Other material parameters are kept constant, e.g., for the mass we choose large values for the solid object and smaller values for the deformable to ensure sufficient deformation. The chosen parameters do not represent the full reality, as there are other material parameters that could be varied. However, as we want to showcase the capabilities of our method, we selected these parameters as they displayed the biggest impact on the deformation behavior.

B.1 Point Cloud Generation

The required point clouds are not directly available in SOFA, but instead rendered from the scene of the meshes using *Raycasting* from Open3D (Zhou et al., 2018). We therefore place virtual cameras around and on top of the scene to generate partial point clouds from different directions. For the Deformable Plate dataset one camera is sufficient, while the other two tasks rely on four cameras around and one camera on top of the scene. This results in a good, but not complete coverage of the entire surface with points of the point cloud. Even though there are five cameras around the scene, there are areas that are not covered: For the tissue, the parts that are occluded by the red liver, and for the cavity, parts of the inner surface depending on how the upper and lower radii deviates from one another. Also, as there can be no camera from below, there are naturally no points on the lower surface for both datasets. In Appendix D we additionally provide results for less cameras on the cavity dataset, leading to only partially observable point clouds. If more than one point cloud camera is used, the resulting point clouds are fused and subsampled accordingly to achieve a processable number of points. We voxel subsample in world space, so the points do not belong to any specific part of the mesh, but can rather be seen as some "interpolation" between mesh vertexes. The main challenge is that there are no point correspondences and that the model needs to figure out which point of the point cloud belongs to which vertex in the mesh to do the correction of the mesh nodes for grounding the simulation. Still, voxel subsampling leads to the most structured results compared to other subsampling techniques, which helps the model to account for correspondences between points over time.

B.2 INPUT FEATURES

In addition to encoding the node or edge type as one-hot features, we add an encoding to static nodes and encode the velocity of the collider in its node features. We encode the positions in space as relative features in the edges instead of absolute encodings in the node features following previous work (Sanchez-Gonzalez et al., 2020). All edges thus receive their relative world coordinates, while mesh edges additionally contain relative coordinates in mesh space.

B.3 COLLISION HANDLING

SOFA as the ground truth simulator handles collision between objects using triangular surface meshes of all objects involved to detect collisions. The detection is implemented using the *LocalMinDistance* method and detected collisions are included in the constraints of the system. Using Lagrangian multipliers, the constraints are then processed together with the other forces from the deformation to solve the complete FEM system (Faure et al., 2012). In contrast to that, GGNS uses one-hot encoded edges between the rigid and the soft body that are used by the model to compute the dynamics. There is no explicit handling of collisions, the network learns to avoid them and adapts the mesh accordingly.

B.4 DEFORMABLE PLATE

For this environment, we simulate a family of 2-dimensional trapezoids deformed by a circular collider with constant velocity. We vary the size of the collider by sampling from a triangular distribution between 15 and $60\ \%$ of the edge length of the deformable object. For the collider start position we sample from a uniform distribution between the left and right corner of deformable object. We record 50 time steps per trajectory and 945 trajectories in total, which are split in $675/135/135$ trajectories per train, evaluation and test set. A single data sample contains approx. 700 nodes: 57 nodes for the collider, 81 nodes for the mesh oft the deformable object and around 600 points in the subsampled point cloud. The mesh itself consists of 416 edges, the total number of edges is about 3 K depending on the deformation in the according time step. In contrast to the Poisson's ratio, the other adjustable material parameter in SOFA, the Young's modulus is kept constant for all samples at $E = 5\,000\text{Pa}$. It describes the compressive stiffness when a force is applied lengthwise. The different material properties together with the different trapezoidal shapes introduce uncertainty in the form of multi-modality into the data. The reason for this is that different deformations result in states that cannot be clearly assigned to a single trapez-material combination. We construct this dataset because it comes with lower computational cost due to the restriction to 2d, but already allows for more general statements due to the non-trivial deformations and the multi-modality. Therefore, it is especially suitable as a proof-of-concept and for ablations.

B.5 TISSUE MANIPULATION

Here, a piece of tissue is deformed by a rigid gripper which could be part of a robot-assisted surgery scenario. To generate diversity, we generate random motions in a $2d$ plane and sample a random gripping point from the 19 top mesh points. We record 100 time steps per trajectory and 840 trajectories in total, which we split in $600/120/120$ trajectories per train, evaluation and test set. A single data sample consists of approx. $1\,200$ nodes: 361 for the mesh, one for the gripper and about 850 for the point cloud. The mesh consists of $2\,154$ edges, which leads to a total number of about $3\,800$ edges depending on the time step. To ensure physically plausible deformation, each Poisson's ratio is assigned its specific Young's modulus from $E \in \{10\,000, 80\,000, 30\,000\}\text{Pa}$. If instead it were kept the same for each Poisson's ratio, the gripper could penetrate the deformable object or pull it without touching it. The uncertainty in this dataset is mainly in the initial state, which can result in different deformations depending on the material from the same initial state.

B.6 CAVITY GRASPING

We randomly generate cone-shaped cavities with radii between 87.5% and 50% of the maximum possible gripping width. The cone shape helps to increase uncertainty in the form of multi-modality in the data, because the states resulting from deformation cannot be clearly assigned to a single

cone-material combination. The deformable cavities are deformed by a simulated Panda [1] robot gripper located at random positions in space. The positions are sampled form a hexahedron around the geometrical center of the cavity ensuring collision free starting positions. For the grasping, the gripper moves as quickly as it is allowed to the gripping position and then closes its fingers with constant velocity. We record 100 time steps per trajectory and 840 trajectories in total, which are split in $600/120/120$ trajectories per train, evaluation and test set. A single data sample consists of approx. 2.4 K nodes: 750 for the mesh, 636 for the gripper and about 1 K for the point cloud. The mesh consists of $4\,500$ edges, the overall number of edges in the graph is about 8.5 K depending on the exact time step. The motivation for the creation of this environment is that a successful use of our method in this setting is an important step on the way to a real-world application.

[1] FRANKA EMIKA GmbH, Munich, Germany

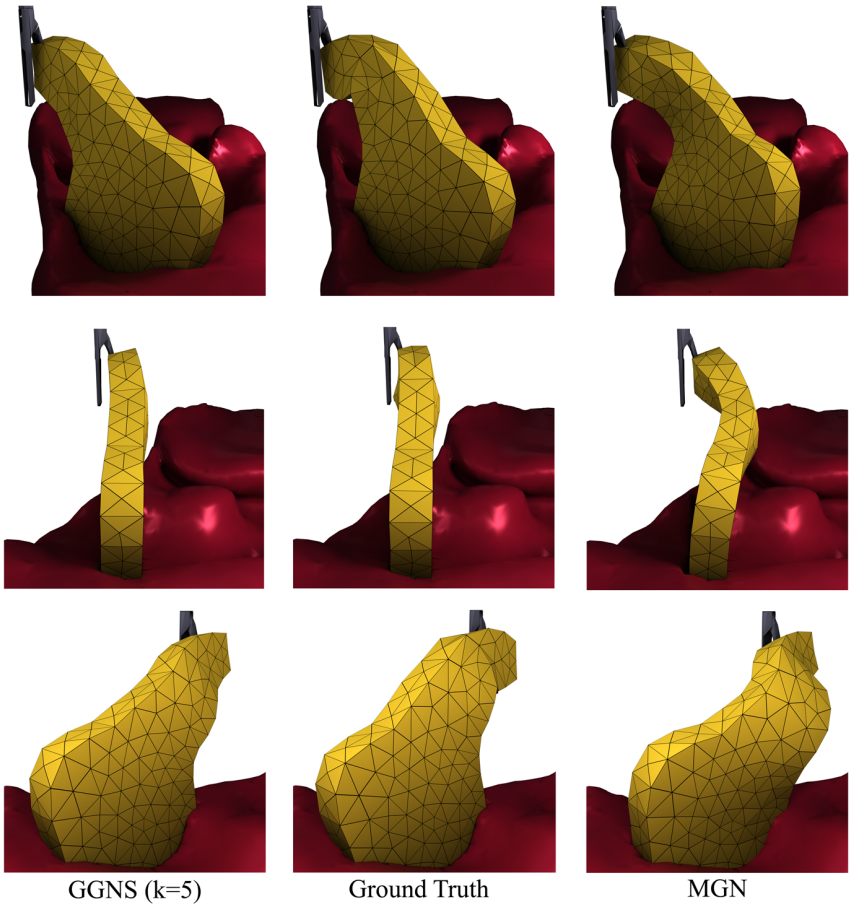


Figure 6: Visualization of a test trajectory of the Tissue Manipulation dataset from three different viewing angles (rows) at time step $t = 70$ for GGNS (left), the ground truth simulation (middle) and MGN (right).

C Qualitative Results

In addition to the qualitative illustrations in the main paper, we also provide further views and examples here: Figure 6 shows the same trajectory as Figure 3 but from three additional viewing angles. Figure 8 and Figure 9 show an overlay of the point cloud on the deformable object during the time step where the simulation is grounded by the point cloud. This representation is comparable to Figure 1 for the Cavity Grasping dataset. Furthermore, we provide example visualizations for a test rollout over time for the Deformable Plate task in Figure 10, for the Tissue Manipulation task in Figure 11, and for the Cavity Grasping in Figure 12. Throughout all tasks, GGNS closely matches the ground truth simulation for the complete rollout, achieving close to optimal results when provided with frequent point cloud information ($k = 2$). Opposed to this, MGN sometimes fails to predict the correct material, leading to poor predictions over time and large mismatches in the final system states.

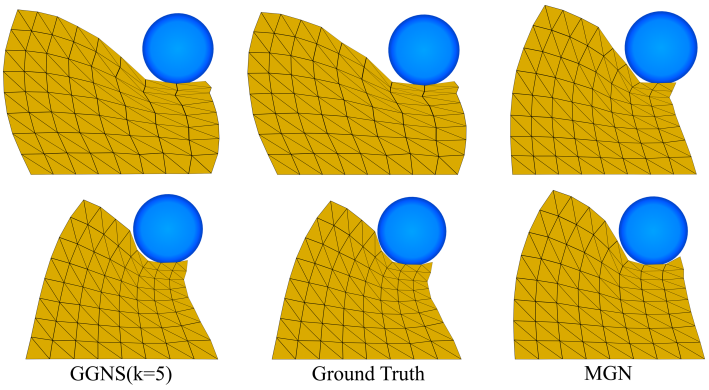


Figure 7: Final simulated meshes ($t = 50$) for GGNS ($k = 5$) (left), the ground truth simulation (middle) and MGN (right) for two test rollouts with different material properties for the Deformable Plate task. Our model closely matches the ground truth simulations for both materials, while MGN predicts the same material every time.

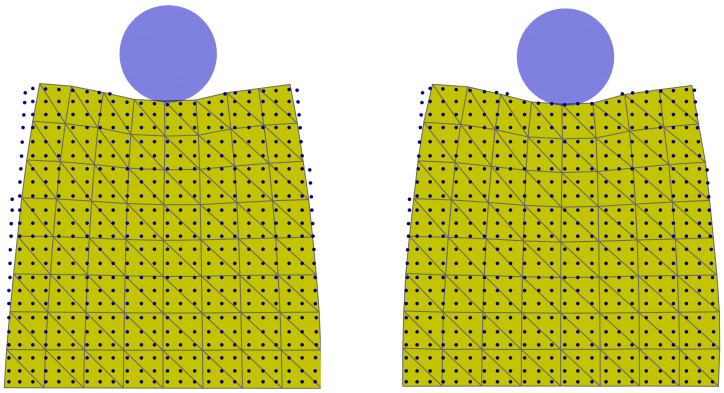


Figure 8: Overlay of the point cloud and the predicted mesh for two consecutive time steps $t = [10, 11]$ in the Deformable Plate dataset. We repeat the point cloud from the earlier simulation step in both images for clarity. The illustration shows the correction behavior of GGNS by including the point cloud to ground the mesh based simulation in this time step. This can be observed particularly well in the upper left and right corners of the plate.

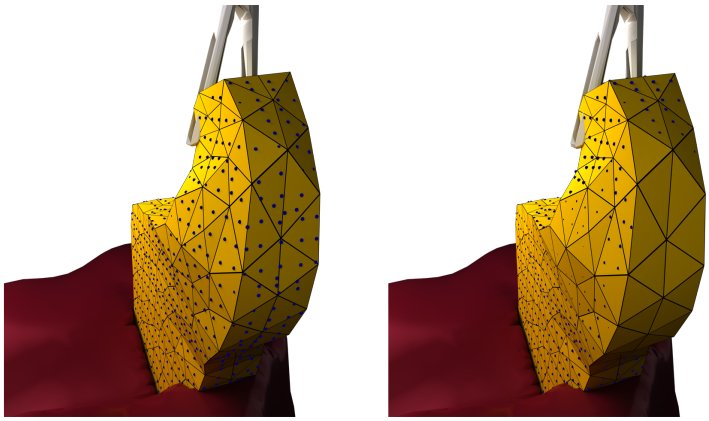


Figure 9: Overlay of the point cloud and the predicted mesh for two consecutive time steps $t = [70, 71]$ in the Tissue Manipulation dataset. We repeat the point cloud from the earlier simulation step in both images for clarity. The illustration shows the correction behavior of GGNS by including the point cloud to ground the mesh based simulation in the time step.

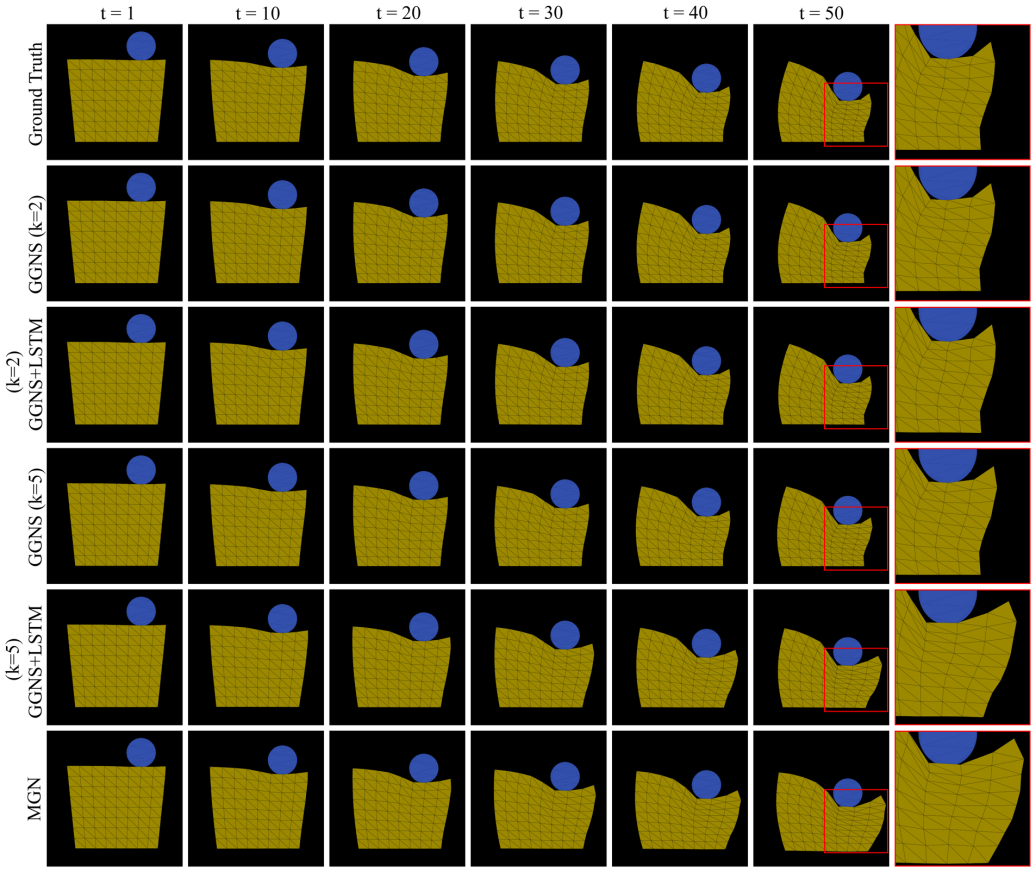


Figure 10: Test rollout visualization for the Deformable Plate task. The last column depicts a close-up of the final time step, which is shown in full in the previous column. Here, we additionally show qualitative results for the GGNS+LSTM model. We can see that for $k = 2$ it matches the ground truth quite well, while for $k = 5$ a large error occurs due to a prediction of the wrong material.

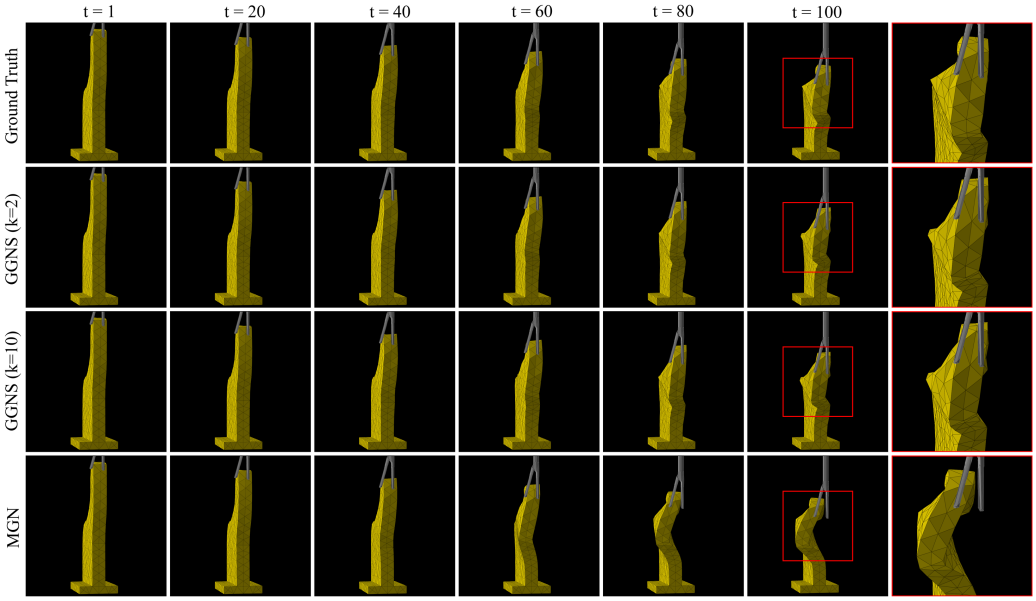


Figure 11: Test rollout visualization for the Tissue Manipulation task. The last column depicts a close-up of the final time step, which is shown in full in the previous column.

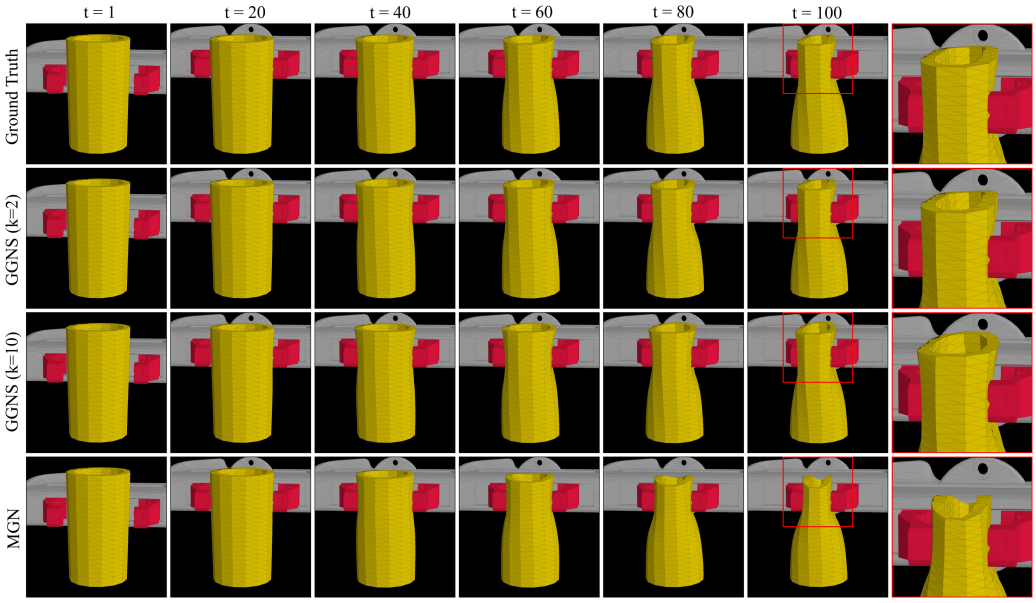


Figure 12: Test rollout visualization for the Cavity Grasping task. The last column depicts a close-up of the final time step, which is shown in full in the previous column.

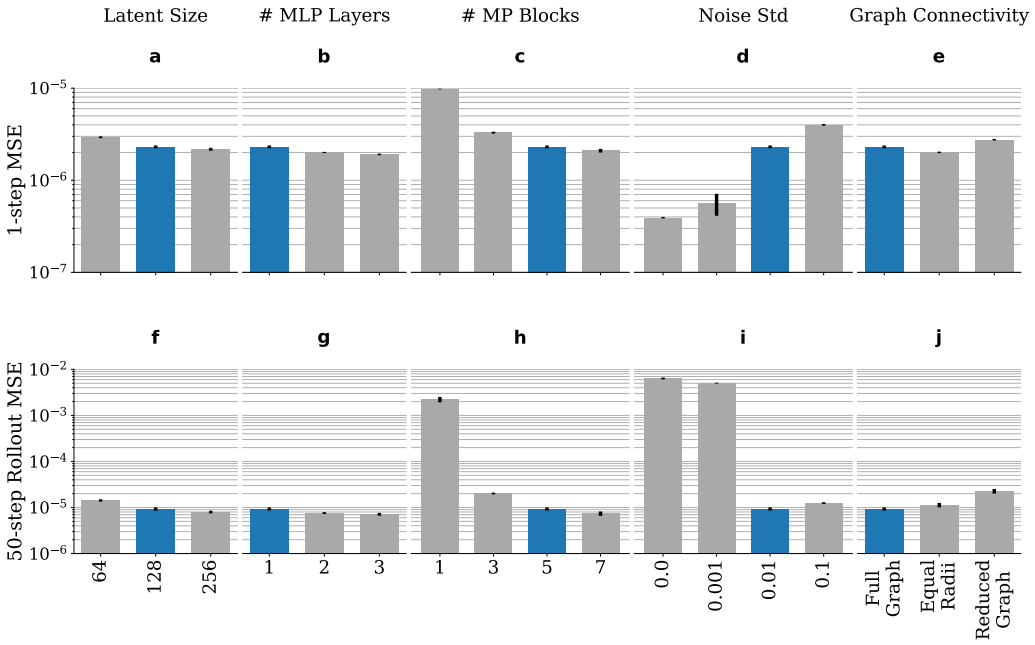


Figure 13: Performance for different changes in hyperparameter choices (grey) on the Deformable Plate dataset in comparison to our default model (blue) with $k = 1$. Error bars indicate one standard deviation. The top row shows the error for the next-step prediction, the bottom row that of full rollouts. We find that a suitable noise scale is crucial for stable rollouts, and that more information in the form of additional edges between the different types of graph nodes generally improves performance. Given enough Message Passing (MP) blocks, further increases in model capacity only lead to modest improvements.

D ABLATIONS

D.1 HYPERPARAMETER CHOICES

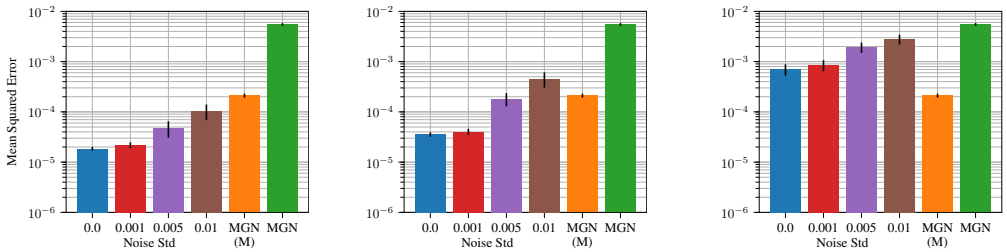
Figure 13 compares the performance of GGNS for different hyperparameter choices. We find that the most importance parameters are the number of Message Passing (MP) blocks and the scale of the noise used in training. Both are crucial to achieve a good performance over multi-step rollouts. In terms of training noise, there is a 1-step/multi-step loss trade-off. Other than that, our approach is robust to variations of the different hyperparameters. In terms of graph connectivity, it can be seen that all settings achieve similar performance. Additional information in the form of more local edges helps slightly, while larger connectivity radii do not do much. A detailed listing of the used edge radii is display in Table 1. In particular, the use of significantly more edges in the *Equal Radii* setting does not provide a significant advantage, which is why we use weaker connectivity *Full Graph* that saves computation time. The results for the *Reduced Graph* settings show that edges within the point cloud are not mandatory. For this reason, we omit these edges in the more complex $3d$ tasks in favor of shorter computation time.

D.2 NOISY POINT CLOUDS

Besides the ablations on our hyperparameter choices, we present further ablations on more realistic point cloud data. For this purpose, we use point clouds with additional noise and only partial observability to get closer to real world point clouds. Figure 14 shows the results for additional ablations on different scales of noise on the point cloud data of the Deformable Plate dataset. We add noise to the point cloud positions during training, evaluation and testing. This makes it more difficult to infer the correct behavior from the point cloud, but provides a more realistic scenario for, because real world point clouds often exhibit large noise. The results show the robustness of our

Table 1: Edge radii for the connectivities between point clouds $\mathcal{P}$ and meshes $\mathcal{M}$ on the 2D Deformable Plate Dataset.

Setting	$\mathcal{P}-\mathcal{P}$	$\mathcal{M}-\mathcal{P}$	World
Full Graph	0.1	0.08	-
Equal Radii	0.2	0.2	-
Reduced Graph	0.0	0.08	-
MGN	0.0	0.0	0.35



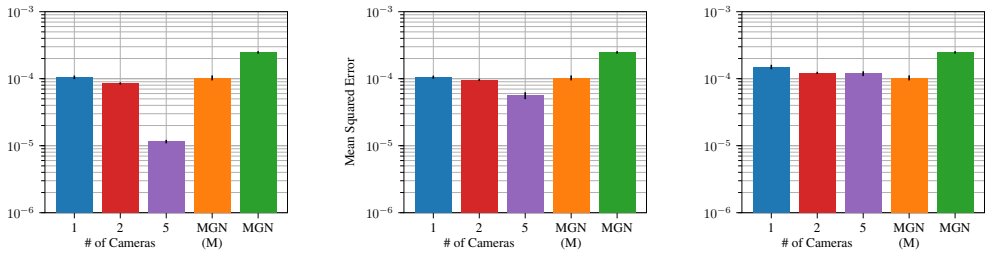
(a) $k = 1$ (b) $k = 2$ (c) $k = 5$

Figure 14: Additional ablations for more realistic point cloud data on two datasets. Here, four different noise levels on the point cloud are evaluated on the Deformable Plate datset. Different grounding frequencies of $k = 1$ in (a), $k = 2$ in (b) and $k = 5$ in (c). GGNS performs better than the baseline even when noise in the scale of the training noise of $\sigma = 0.01$ is applied to the point cloud.

method: Even when a noise level of $\sigma = 0.01$ is applied to the point cloud during testing, it clearly outperforms the baseline. This noise level corresponds to the amount of noise used on the mesh during training.

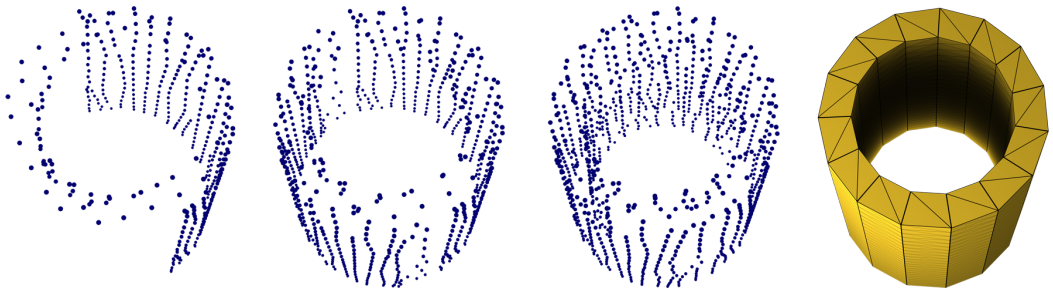
D.3 Partial Observable Point Clouds

For the ablations on the partial observability, we use the Cavity Grasping dataset. We generate the partial point clouds by using only one, two or five virtual point cloud cameras when using raycasting. The resulting point clouds are visualized for better clarity in Figure 16 for an example test trajectory at time step $t = 0$. One camera results in a coverage from only one half of the outer surface of the cavity and two cameras cover almost the complete outer hull but not the inner surface. With five cameras, the point cloud covers almost the entire mesh completely, except for the inside and bottom. The resulting point clouds have a very different number of points: About 400 for one camera, about 600 for two cameras, and about 1000 for five cameras compared to 750 mesh nodes for the cavity. The results in Figure 15 show that even with these much less complete point clouds, GGNS still outperforms the baseline. For $k \leq 5$ this is the case even if the baseline has access to the full initial state, which GGNS has not.



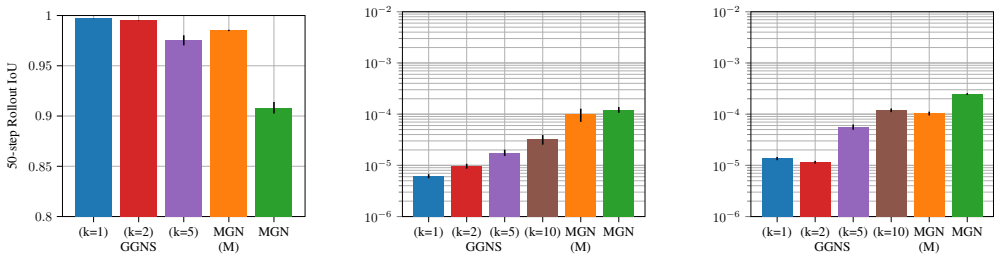
(a) $k = 2$ (b) $k = 5$ (c) $k = 10$

Figure 15: Additional ablations for more realistic point cloud data on the Cavity Grasping dataset. For this purpose, different numbers of cameras are used when generating the point cloud using raycasting. Comparison for three different grounding frequencies:$k = 2$ in (a), $k = 5$ in (b) and $k = 10$ in (c). GGNS outperforms the baseline for all camera settings and grounding frequencies k.



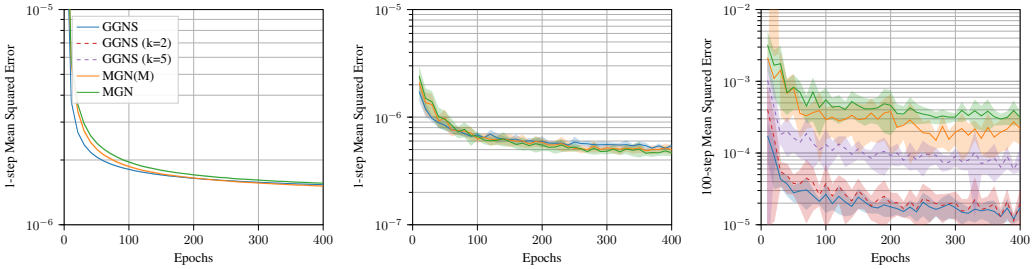
One Camera Two Cameras Five Cameras Mesh

Figure 16: Visualization of the point clouds using one, two or five cameras for the raycasting and the corresponding mesh for reference. It is clearly visible how better coverage of the object is achieved as the number of cameras increases.



(a) Deformable Plate IoU (b) Tissue Manipulation (c) Cavity Grasping

Figure 17: (a) Rollout IoU for the Deformable Plate dataset for GGNS and the MGN baseline. The main findings here are similiar to the MSE results. Rollout Mean Squared Error of GGNS and MGN baselines evaluated on the test set of the Tissue Manipulation dataset b and Cavity Grasping dataset c. We report the results for GGNS using point clouds in every k-th time step. MGN(M) indicates the baseline method of MGN that uses the ground truth material as input feature. GGNS outperforms the MGN baseline in all settings and in most cases even if it has access to the complete initial state.



(a) Train loss (b) 1-step Evaluation Loss (c) 100-step Evaluation Loss

Figure 18: Exemplary learning curves for the Cavity Grasping task. The light shaded area indicates one standard deviation. Both GGNS and the baselines learn the task pretty similarly in terms of 1-step predictions. Our model is only evaluated for the $k = 2$ and $k = 5$ variant during full rollout evaluation. Here, we can clearly see the advantage of using the point cloud information.

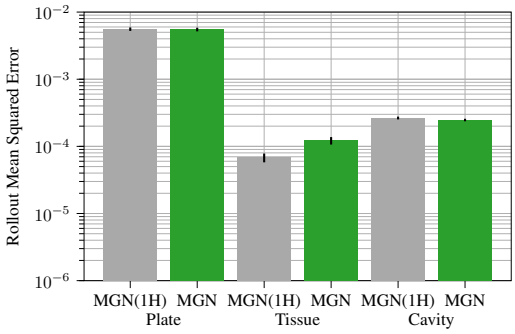


Figure 19: Comparison of the MGN baseline with a version using the one-hot encoded edge types instead of an explicit edge type partitioning indicated by *MGN (1H)*. Both are compared for all three tasks and no significant advantage of the explicit edges partitioning could be found. For this reason, GGNS uses the one-hot encoding, because it is both conceptually simpler and requires less computational power. The MGN baseline still uses explicit edge type partitioning throughout this work, following Pfaff et al. (2021).

Table 2: Configuration of the hyperparameters and key information of the training of our model for all experiments.

Parameter	Value
Batch Size	32
Optimizer	Adam
Learning Rate	5×10^{-4}
Activation Function	LeakyReLU
Aggregation Function	Mean
Encoder	Linear Layer
MP-Blocks	5
MLP Layers	1
Latent Dimension	128
Decoder	1-layer MLP
Residuals Connections	Around each MP block
Training Noise Std	0.01

Table 3: Task specific configuration and hyperparameters for our experiments. We vary the graph connectivity and the number of training epochs for different tasks to control the total training time of our method.

Parameter	Plate	Tissue	Cavity
Connectivity Setting	Full Graph	Reduced	Reduced
Number of Epochs	1000	800	400
Approx. Training Time	$21:00$ h	$40:00$ h	$38:00$ h

E HYPERPARAMETERS

We train all models on all tasks using the Adam optimizer (Kingma & Ba, 2015) with a learning rate of 5×10^{-4} and a batch size of 32, using early stopping on a held-out validation set to save the best model iteration for each setting. The models use a LeakyReLU activation function, 5 message passing blocks with 1-layer MLPs and a latent dimension of 128 for node and edge updates. We use a mean aggregation for the edge features and a training noise of 0.01. All tasks use a normalized task space of $[-1, 1]^d$. For all models and tasks, we normalize the task space to $[-1, 1]^d$ and add a training noise of 0.01. All experiments are repeated for 10 random seeds unless otherwise noted, and we report the mean and standard deviation of the results. Table 2 gives an overview of hyperparameters shared across tasks. Since GNS are generally robust to the choice of hyperparameters (c.f. D), we use the same hyperparameters for all task and for both, GGNS and MGN for simplicity. The only hyperparameters that vary over tasks are the graph connectivity and the number of training epochs, as shown in Table 3. We adapt these parameters to control for the total training time on a single GPU.