Simulating Liquids with Graph Networks

Jonathan Klimesch, Philipp Holl, Nils Thuerey

March 16, 2022

Abstract

Simulating complex dynamics like fluids with traditional simulators is computationally challenging. Deep learning models have been proposed as an efficient alternative, extending or replacing parts of traditional simulators. We investigate graph neural networks (GNNs) for learning fluid dynamics and find that their generalization capability is more limited than previous works would suggest. We also challenge the current practice of adding random noise to the network inputs in order to improve its generalization capability and simulation stability. We find that inserting the real data distribution, e.g. by unrolling multiple simulation steps, improves accuracy and that hiding all domain-specific features from the learning model improves generalization. Our results indicate that learning models, such as GNNs, fail to learn the exact underlying dynamics unless the training set is devoid of any other problem-specific correlations that could be used as shortcuts.

1 Introduction

Simulating complex dynamics is invaluable to many fields in computer graphics and the natural sciences. Numerical simulations of such dynamics are computationally challenging and are often inaccurate in approximating the underlying physical systems \cite{17}. Recent work suggests, that deep learning models can be an efficient and accurate alternative to traditional, hand-crafted simulation approaches \cite{18, 13, 21}. Graph networks in particular have become popular in collider physics \cite{20}, astrophysics \cite{6} or chemistry \cite{12} and have been used to build various data-driven simulators \cite{4, 12, 3, 19, 16}.

Recently, Sanchez-Gonzalez \textit{et al.} proposed the \textit{Graph Network-based Simulators} (GNS) framework, where they approximate fluid dynamics by learned message-passing on particle graphs \cite{18}. We replicate their results on data from our custom fluid-Implicit-Particle (FLIP) \cite{5, 25} simulator, and extend the GNS framework with new training variants, investigating the impact of the random noise and the generalization of the trained model to different domains.

We use the differentiable physics framework $\Phi_{\text{Flow}}$ \cite{14} to implement our FLIP simulator in a differentiable manner, allowing us to roll out multiple simulation steps during training, similar to \cite{22}. This enables us to formulate an alternative training scheme where we feed the predicted next simulation state back to the network as an input. The network can thus learn under conditions matching inference time, which makes the noise term superfluous.

2 Network architecture

We adopt the Graph Network-based Simulators architecture from Sanchez-Gonzalez \textit{et al.} \cite{18} and train it on the dynamics of our FLIP simulator. They compose several GN blocks into an encode-process-decode configuration which they name \textit{Graph Network-based Simulators} (GNS). They demonstrate that the GNS is capable of learning complex dynamics of different materials, such as fluids or sand. Figure 1 displays the GNS architecture schematically. On a high level, the GNS is a parameterization of dynamics which map the current state of the world to a future state. In case of the FLIP simulation, the dynamics are described by fluid dynamics. In the Lagrangian view, the state of a fluid can be described in terms of single particles which interact with each other. The GNS takes the positions of these particles from the FLIP simulation as input and uses its encoder to transform these positions into a graph, where each node corresponds to a particle, and edges indicate particles interacting with each other. For each particle, edges are added to all neighboring particles within a certain connectivity radius, which is a hyperparameter of the GNS architecture. The node features are calculated by a multilayer perceptron.
Figure 1: Schematic overview of the Graph Network Simulator (GNS) architecture.

(MLP) which takes the five previous particle velocities, the particle type and its distance to the domain boundaries as input, and outputs a latent vector of size 128. The edge features are calculated by another MLP, which takes the relative distance between the connected nodes as input and produces another latent vector of size 128.

The particle interactions are then computed by performing message passing on the resulting graph. This is done by a sequence of 10 different interaction networks, all having residual connections between the node and edge attributes at input and output. The update functions of the interaction networks are implemented as MLPs and the aggregation function as an element-wise sum. The parameterization of the fluid dynamics thus takes the form of message-passing on a graph, performed by MLPs.

The resulting graph is then used to map the input state, the particle positions, to the next state. After the message passing, the GNS first uses its decoder, another MLP, to transform the node features into particle-wise accelerations. These accelerations are then used to calculate future velocities and positions for each particle using Euler integration (Figure 1).

All MLPs consist of two hidden layers with ReLU activations and size 128. The MLPs of the encoder use LayerNorm after their output layer. A detailed analysis of the hyperparameters of the GNS architecture (e.g. number of velocities to calculate node features, number of interaction networks) can be found in the work of Sanchez-Gonzales et al. They implemented the GNS architecture in TensorFlow using Sonnet and the Graph Nets library. The implementation was published on GitHub and is also used to apply different variants of the GNS to our FLIP simulation, as described in section 4.

3 FLIP data sets

To apply the GNS architecture to the FLIP simulation, we first use our simulator to generate training, validation and test sets. Table 1 summarizes the parameters which we use to generate the training and validation sets. The training set consists of 2000 simulations with 400 frames each, using a time step of 0.05 s. The scenes are randomly generated and can contain liquid blocks, elongated obstacles and a liquid pool at the bottom. Figure 2 shows initial simulation states as examples of the training set. The validation set consists of 20 simulations with length 400 and is generated with the same random distribution as the training set, using a different random seed. We generate our test set manually to evaluate the behavior of the models in 10 challenging scenarios. Examples from the test set are shown in figure 3.

Sanchez-Gonzales et al. normalize all input and target vectors element-wise to zero mean and unit variance. Thus, we calculate the data set statistics online during data generation and use them later.
| Parameter description                  | Value       |
|---------------------------------------|-------------|
| Trajectory duration (steps)           | 400         |
| Time step (s)                         | 0.05        |
| Maximum number of particles           | 1300        |
| Domain size                           | $32 \times 32$ |
| Probability for pool                  | 0.3         |
| Pool height                           | [3, 8]      |
| Block size                            | [2, 20]     |
| Probability for multiple blocks       | 0.3         |
| Number of blocks if there are multiple| [2, 3]      |
| Probability for obstacles             | 0.8         |
| Number of obstacles                   | [1, 5]      |
| Obstacle length                       | [2, 20]     |
| Obstacle rotations                    | [0, 90]     |
| Probability for initial velocity      | 0.3         |
| Initial velocity range                | [-5, 5]     |

Table 1: Parameter specifications for data set generation. Tuples indicate ranges from which values were drawn randomly. The use of some of the parameters is dependent on the corresponding probabilities (e.g. number of obstacles is only considered if probability for obstacles is met).

![Simulation examples from our training set.](image)

for normalization during training and inference. Due to memory constraints, we limit the number of particles in each FLIP simulation to less than 1300. All trajectories are generated on domains of size $32 \times 32$ and scaled to size $0.8 \times 0.8$ with positions laying between 0.1 and 0.9.

The training set used by Sanchez-Gonzales et al. [18] is mostly generated by simulators based on the Material Point Method (MPM). They show that the connectivity radius is one of the most important hyperparameters of the GNS architecture. Their experiments indicate that greater connectivity radii yield lower errors. As explained in their work, larger particle neighborhoods enable long-range communication among the particles, supporting the message passing which applies the dynamics. However, this communication benefit stands in trade-off with the amount of computation and memory, which increases with larger radii due to the size of the graphs.

Sanchez-Gonzales et al. use a connectivity radius of 0.015 on a domain of size $0.8 \times 0.8$. Using this connectivity radius to train the GNS on FLIP trajectories yielded unstable results where the fluid blocks were torn apart within the first few time steps. Instead, a higher connectivity radius of 0.03 was necessary in order to produce physical trajectories. Figure 4 shows the time evolution of the distribution of neighboring particles (averaged over 50 simulations), using a connectivity radius of 0.03 for the FLIP and MPM WaterRamps data set from [18]. The left side of Figure 4 shows that the neighbor number in our FLIP simulations drops at around frame number 50 indicating that the liquid blocks splash at the bottom. After that initial drop, the mean number of neighbor particles (thick red dots) stays nearly constant. This is expected due to the incompressibility constraints of the simulation. For the MPM trajectories (Figure 4 right), the mean number of neighbor particles slightly increases, indicating that the fluid gets compressed over time. This enables long-range communications with a smaller radius. The maximum number of neighbor particles differs strongly between FLIP and MPM simulations, reaching higher values in the MPM case. This further indicates that even a low connectivity radius enables long-
range communication between particles in the MPM case while a higher radius is required in case of our FLIP data set.

Figure 3: Simulation examples from our custom-designed test set.

Figure 4: Distribution of neighbor numbers for the connectivity radius 0.03 at different time steps, averaged over 50 data set samples. The left side shows results for the FLIP data set, the right side shows results for the MPM WaterRamps data set used in [18]. The thick red dots mark the mean value of neighbor numbers. The distribution was interpolated between values at the positions of the thin red dots. The color indicates the number of particles with the respective number of neighbors.

4 Training schemes and loss functions

Sanchez-Gonzales et al. [18] optimize the GNS parameters Θ using the $L_2$ loss

$$\mathcal{L} = \|\vec{p}_{GNS}^t - \vec{p}_{GT}^t\|^2$$

which compares the predicted accelerations (the outputs of the GNS) with the corresponding ground truth (GT) accelerations at time step $t$. A trained GNS can then be used to generate longer simulation rollouts by using the GNS outputs as its new inputs for the next time step. However, due to the complex mechanics and imperfect optimization, the GNS would then accumulate its own error over time. With the loss defined in equation 1 the GNS model is only trained on one-step data and is therefore not trained for handling this accumulated error. One countermeasure used by Sanchez-Gonzales et al. [18] is to corrupt the model input during training with artificial noise, which ideally matches the noise produced by the model during rollout generation.

First, we verify the capabilities of the GNS framework by training it on our FLIP data set. Except for the connectivity radius described in the last section, we train the first GNS (named 1-step-noise, 1sn)
with the same procedure as used by Sanchez-Gonzales et al. [18]. Additionally, we train another GNS (named 1-step, 1s) without the additional artificial noise.

As a next step, we implement an alternative to the artificial noise. This alternative uses a loss function which spans over more than one step and is described as

\[
L = \frac{1}{n} \left( \| \hat{p}_{GNS}^{t}(p_{GT}^{t-1}) - \hat{p}_{GT}^{t} \|^2 + \sum_{i=1}^{n} \| \hat{p}_{GNS}^{t+i}(p_{GNS}^{t+i-1}) - \hat{p}_{GT}^{t+i} \|^2 \right)
\]

(2)

where \(n\) is the number of additional steps using the GNS positions as input for the next prediction. The GNS first takes the five previous ground truth velocities as input (\(p_{GT}^{t-1}\)), outputs the particle-wise accelerations and calculates the new velocities and particle positions. In the next step, it takes the four previous ground truth velocities and the newly predicted velocities as input (\(p_{GNS}^{t+i-1}\)) and calculates the next velocities which replace another ground truth velocity in the next step. This way, the model is constantly confronted with its own error which should improve its error mitigation capabilities during rollout generation. Due to memory constraints, we test this alternative only with \(n = 1\). We optimize one GNS using this loss from scratch (named 2-step-scratch, 2ss) and another one by initializing it using pretrained weights from the 1-step model described above (named 2-step-initialized, 2si).

As described above, the node features in the graph encoding are calculated by an MLP using the five previous velocities, the particle types and the distances to the domain boundaries as input. However, giving the GNS information about the domain boundaries restricts its generalization capabilities to the domain it has seen during training. An alternative which we explore in this report is to model the domain boundaries as obstacles and to remove the additional domain boundary distances from the node feature calculation. This alternative has two advantages. First, the model is not restricted to any specific domain, but operates solely on the interaction between fluid and obstacle particles. Second, the model sees more interactions between fluid and obstacle particles during training, which might improve its capabilities even further. We train a fifth model on a new version of our FLIP training set where we represent the domain boundaries as obstacle particles. We train this model with artificial noise, using the loss from equation 1 and name it 1-step-noise-bounded (1snb).

We optimize the parameters of our GNS models with the Adam optimizer [15] and a batch size of 2. The learning rate is decreased exponentially from \(10^{-4}\) to \(10^{-6}\). We adopted this optimization procedure from Sanchez-Gonzales et al. and use it for all our experiments.

5 Quantitative comparison

To compare the model variants from section 4 quantitatively, we use 4 different metrics. The first metric is the particle-wise mean-squared error of one-step acceleration predictions (MSE-acc 1) as defined in equation 1. The second metric (MSE 20) is the MSE averaged across time, particles and spatial dimensions of 20 frames, taken at each 20 steps of the full 400-step rollouts. This metric indicates the model performance at different stages of the FLIP trajectories (e.g. the falling liquid block in the beginning or the sloshing liquid at the bottom of the domain at later time steps). The third metric (MSE 400), is the MSE averaged across time, particle and spatial dimensions, but applied to the full rollouts of length 400.

Evaluating rollouts solely with particle-wise MSE can be misleading. Considering two particles A and B, the MSE could be high if the model predicts particle A at the position of particle B and vice versa, even so the particle distributions of prediction and ground truth match. Due to the chaotic nature of fluid motion, the MSE can therefore be misleading and one might favour a distributional metric, which is invariant under particle permutations. Therefore, we evaluate the model variants from section 4 with a fourth metric (EMD) using optimal transport (OT) and the earth mover’s distance or Wasserstein metric [23, 8].

The optimal transport problem describes an optimization problem of the form

\[
d_{M}(r, c) = \min_{P \in U(r, c)} \sum_{i,j} P_{ij} M_{ij},
\]

(3)
Table 2: Testset performance of all model variants from section 4, evaluated with multiple metrics. The MSE is reported for one-step (MSE-acc 1) acceleration predictions or averaged over either 20-step rollouts (MSE 20) or full rollouts (MSE 400). Bold values indicate the best score of the respective metric.

| Model variant              | EMD (10^{-2}) | MSE-acc 1 (10^{-1}) | MSE 20 (10^{-2}) | MSE 400 (10^{-2}) |
|----------------------------|---------------|---------------------|-----------------|-------------------|
| 1-step                     | 2.598         | **3.775**           | 5.653           | 2.853             |
| 1-step-noise               | 1.617         | 4.980               | 5.574           | 2.469             |
| 1-step-noise-bounded       | **1.336**     | 4.985               | 5.648           | 2.959             |
| 2-step-scratch             | 1.428         | 4.535               | 5.181           | 2.682             |
| 2-step-initialized          | 1.367         | 3.933               | **4.711**       | **2.430**         |

Here, \( r \) and \( c \) describe the probabilistic weights (sum to 1) of all particles in the source and target distribution and \( U(r, c) \) is the set of positive \( n \times m \) matrices defined by

\[
U(r, c) = \{ P \in \mathbb{R}_{>0}^{n \times m} \mid P1_m = r, P^\top 1_n = c \}.
\]

\( M \) is the cost matrix, which in case of a fluid with \( n \) particles has the size \( n \times n \) and contains the particle-wise distances. The optimization problem, described by equation 3, is to find the Wasserstein distance \( d_M(r, c) \) which is calculated by using the element of \( U(r, c) \) which yields the lowest distance between the particle distribution of ground truth and prediction, measured by multiplying it with the cost matrix \( M \). This distance is also called the earth mover distance (EMD). This metric therefore relates each particle of the predicted particle distribution to its nearest neighbors in the target distribution and is invariant under particle permutations. For our evaluations, we calculate the Wasserstein distance using the Python Optimal Transport (POT) library [11].

We evaluate the performance of each model on the validation set during training. Figure 5 shows exemplary screening results using all four metrics for the 1-step-noise model variant described in section 4. As the MSE-acc 1 is calculated in the same way as the loss with which this model was trained (equation 1), this metric shows a stable downwards trend with increasing number of training steps. The MSE 20 shows a similar behavior as it only includes 20-step rollouts and is still quite similar to the training loss. The MSE 400 and EMD metrics produce relatively noisy screenings which still show a strong downwards trend. The increase in noise is explained by the fact that the model is not directly optimized for the tasks measured by these metrics. Due to the chaotic weight updates within the network during training, small updates for the optimization with the one-step loss can have large impacts on the performance in the full trajectory rollout task.

To favour models which have a high performance on generating full-scale rollouts, we use the MSE 400 to choose model checkpoints for further analysis and apply the best checkpoint of each model variant to the test set. Figure 6 summarizes the performances of all models evaluated by the four metrics described before. The exact scores underlying this figure can be found in table 2 in the appendix. Figure 7 shows the trajectories of the MSE 400 and the EMD over time for all model variants, averaged over the entire test set. Figure 8 shows exemplary rollout predictions from all model variants for one of the test set scenarios.

![Figure 5: Screening results of the 1-step-noise model on the validation set during training.](image)

The 1-step model variant shows the best performance with respect to the MSE-acc 1 metric. This model is solely trained for this task, without any artificial noise or multi-step error, leading to its superior
performance on the specific task of one-step predictions. While its MSE 20 and MSE 400 evaluations are rather similar to other model variants, its EMD evaluation shows a far worse performance. Therefore, the model’s rollout prediction follows the ground truth rollout closely, but rather tries to hold the particles at similar positions, than to really apply dynamics which produce a similar visual behavior compared to the FLIP trajectory. The rollout example in figure 8 shows that the model sometimes produces strange, nonphysical motions (last column).

The 1-step-noise model shows a much better EMD performance, but performs worse than the 1-step model when evaluated with MSE-acc 1. This is likely due to the artificial noise which challenges this model with an additional difficulty during training. Figure 8 shows that this model variant leads to fluid motion which is more compressed than in the ground truth rollout. Similar compression can be seen in the rollout of the 1-step model.

The 1-step-noise-bounded model has the best EMD performance, resembles the 1-step-noise model’s performance for MSE-acc 1, but has lower performance in case of MSE 20 and MSE 400. With removal of the boundary distance features, the model has to extract this information solely from the interaction between fluid particles and obstacles. Figure 8 shows that this model variant produces rollouts with a similar density than the ground truth. However, it produces dynamics which cause the fluid to slide faster along the boundaries, possibly caused by the constant interactions with boundary particles which normally lets a fluid block splash at an obstacle.

For the 2-step model variants, the 2-step-initialized model has better performances in all metrics compared to the 2-step-scratch model. It surpasses all other variants on the MSE 20 and MSE 400 metrics and reaches similar performances to the best models for the MSE-acc 1 and EMD metrics. Thus, using a 1-step model as initialization for 2-step models seems to have a positive effect. However, figure 8 shows that it has inherited the compression tendency of the 1-step model, leading to a lower density of the
fluid during rollout. In contrast, the 2-step-scratch model preserves the density throughout the rollout (similar to the 1-step-noise-bounded model).

![Figure 8: Example predictions for all five model variants. Different time steps are indicated at the top. The top row shows the ground truth trajectory. The initial state of the trajectory is shown at the top of the second column of figure 3. Videos available at https://git.io/JOOQG.](image)

As described in section 4, the 1-step-noise-bounded model should have the capability to generalize to larger domains. Figure 9 shows two examples from such a generalization experiment. We extend the domain size of $32 \times 32$ to $32 \times 64$ and adapt the scaling to put the position values into the range of 0.1 to 0.9 in $x$ and 0.1 to 1.8 in $y$ direction. Then, we use both models, 1-step-noise and 1-step-noise-bounded (trained on $32 \times 32$ domains), to generate rollouts for these new scenarios. Figure 9 shows that the 1-step-noise model produces dynamics which let the fluid splash when crossing half of the new domain. This height equals the bottom of the domain seen during training where fluid particles normally splash and are accelerated in $x$-direction. Therefore, the model seems to have learned to correlate accelerations...
in x-direction with the distance to the boundaries.

The 1-step-noise-bounded model shows better behavior than the 1-step-noise model but still deforms the fluid block after crossing half of the domain. This indicates that the model normally not only correlates accelerations and boundary distances but also accelerations and velocities. In the training examples, most fluids had similar velocities right before splashing at the bottom. When the fluid crosses half of the new domain at height 32, the particles cross this velocity threshold, causing the model to decelerate the particles in y- and accelerate the particles in x-direction.

Figure 9: Two example predictions from the domain generalization experiment. Both left and right examples show ground truth (FLIP), predictions from 1-step-noise and predictions from 1-step-noise-bounded at the frames indicated by the numbers. The red dashed lines indicate the original domain size. Videos available at https://git.io/JOOQn.

6 Conclusions

We have shown, that GNS models are capable of learning fluid dynamics from a dataset produced by our differentiable FLIP simulator. Training the GNS models with our multi-step loss enables the models to mitigate accumulating errors in simulation rollouts and yields competitive results compared to models trained with the artificial noise proposed by Sanches-Gonzales et al. [18]. Replacing the domain-specific boundary distance features with obstacle boundaries increases the generalization to larger domains. However, we find that these models still have the tendency to deform the fluids as soon as they are
passing the original domain size seen during training. This indicates that the GNS does not learn the true underlying physics but rather problem-specific correlations between input velocities and output accelerations. This is supported by the fact that the architecture takes the previous five velocities as inputs and does not only rely on just the previous velocity as one would expect from physical dynamics. Furthermore, most models are unable to retain the original density of the fluid and compress the fluid particles much more than the FLIP simulator.

Extending the GNS architecture with strong inductive biases towards physical laws and symmetries could improve its physical understanding and force it to learn actual dynamics instead of problem-specific correlations. Future work should also concentrate on examining the physical reasoning of learned simulators in more detail. Transforming parts of learned simulators into symbolic models [6, 7] and extending tools like the recently proposed GNNExplainer from Ying et al. [24] could provide further insights into the reasoning of Graph Networks and could yield new ideas on how to improve their physical understanding.

References

[1] M. Abadi et al. Tensorflow: A system for large-scale machine learning. USENIX Association, 2016.
[2] J. Ba, J. Kiros, and G. E. Hinton. Layer normalization. arXiv preprint arXiv:1607.06450, 2016.
[3] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, et al. Relational inductive biases, deep learning, and graph networks. arXiv preprint arXiv:1806.01261, 2018.
[4] P. W. Battaglia, R. Pascanu, M. Lai, D. J. Rezende, et al. Interaction networks for learning about objects, relations and physics. Advances in neural information processing systems, pages 4502–4510, 2016.
[5] J.U. Brackbill and H. M. Ruppel. Flip: A method for adaptively zoned, particle-in-cell calculations of fluid flows in two dimensions. Journal of Computational Physics, 65(2):314–343, 1986.
[6] M. Cranmer, A. Sanchez-Gonzalez, P. W. Battaglia, R. Xu, K. Cranmer, D. Spergel, and S. Ho. Discovering symbolic models from deep learning with inductive biases. arXiv preprint arXiv:2006.11287, 2020.
[7] M. Cranmer, R. Xu, P. Battaglia, and S. Ho. Learning symbolic physics with graph networks. arXiv preprint arXiv:1909.05862, 2019.
[8] M. Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In NIPS, 2013.
[9] DeepMind. Sonnet, 2017. https://github.com/deepmind/sonnet
[10] DeepMind. Graph nets library, 2018. https://github.com/deepmind/graph_nets
[11] R. Flamary and N. Courty. Pot python optimal transport library, 2017.
[12] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. Proceedings of the 34th International Conference on Machine Learning, 70:1263–1272, 2017.
[13] S. He, Y. Li, Y. Feng, S. Ho, S. Ravanbakhsh, W. Chen, and B. Póczos. Learning to predict the cosmological structure formation. Proceedings of the National Academy of Sciences of the United States of America, 116:13825–13832, 2019.
[14] P. Holl, V. Koltun, and N. Thuerey. Learning to control pdes with differentiable physics. arXiv preprint arXiv:2001.07457, 2020.
[15] D. P. Kingma and J. L. Ba. Adam: A method for stochastic optimization. 3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings, pages 1–15, 2015.
[16] Y. Li, J. Wu, R. Tedrake, J. B. Tenenbaum, and A. Torralba. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. In ICLR, 2019.
[17] S. Rasp, M. Pritchard, and P. Gentine. Deep learning to represent subgrid processes in climate models. *Proceedings of the National Academy of Sciences of the United States of America*, 115:9684–9689, 2018.

[18] A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, and Battaglia P. W. Learning to simulate complex physics with graph networks. *arXiv preprint arXiv:2002.09405*, 2020.

[19] A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, Mered J., M. Riedmiller, R. Hadsell, and P. W. Battaglia. Graph networks as learnable physics engines for inference and control. *arXiv preprint arXiv:1806.01242*, 2018.

[20] J. Shlomi, P. W. Battaglia, and J. Vlimant. Graph neural networks in particle physics. *arXiv preprint arXiv:2007.13681*, 2020.

[21] N. Thuerey, K. Weissennow, H. Mehrotra, and N. Mainali. Deep learning methods for reynolds-averaged navier–stokes simulations of airfoil flows. *AIAA Journal*, 58:15–26, 2018.

[22] K. Um, Y. Fei, R. Brand, P. Holl, and N. Thuerey. Solver-in-the-Loop: Learning from Differentiable Physics to Interact with Iterative PDE-Solvers. *arXiv preprint arXiv:2007.00016*, 2020.

[23] C. Villani. *Topics in optimal transportation*. American Mathematical Soc., 2003.

[24] R. Ying, D. Bourgeois, J. You, M. Zitnik, and J. Leskovec. Gnnexplainer: Generating explanations for graph neural networks. *Advances in neural information processing systems*, 32:9240–9251, 2019.

[25] Y. Zhu and R. Bridson. Animating sand as a fluid. *ACM Trans. Graph.*, 24(3):965–972, 2005.