Physically-Motivated Machine Learning Models for Lagrangian Fluid Mechanics

Rene Winchenbach
Technical University of Munich
Physics-based Simulations
Munich, Germany
rene.winchenbach@tum.de

Nils Thuerey
Technical University of Munich
Physics-based Simulations
Munich, Germany

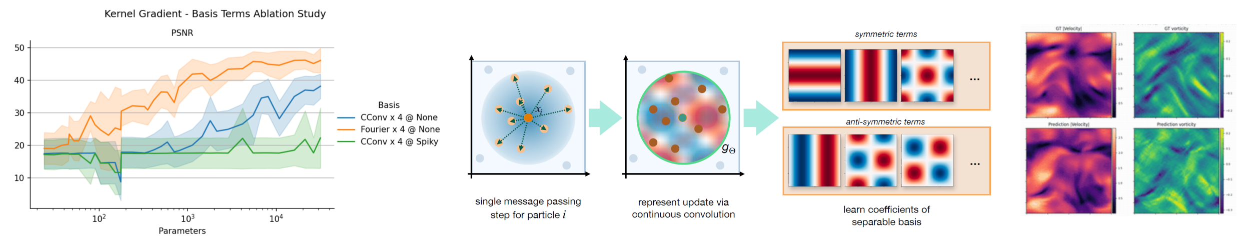

Fig. 1: Our Machine Learning Model built on symmetries (middle) allows for much better performance across a broad range of problems (left). In contrast, our Fourier-based approach allows for significantly greater performance for equal-sized networks compared to prior work, adding non-physically plausible biases does not improve performance, such as non-gaussian window functions (CConv @ Spiky) and yields stable predictions in unseen SPH simulations in a one-shot training using only a single validation case for training (right, velocity color-coded, top ground-truth bottom prediction).

***Abstract*— Machine learning models are often treated as black boxes with vast amounts of data, and they produce the desired predictions, given enough resources. While initial research in learning Lagrangian fluid simulations relied heavily on this approach[1], later results demonstrated that simply providing data is generally insufficient to learn physically correct behavior, e.g., conservation of momentum[2]. Consequently, there has been a growing trend of machine learning approaches that directly enforce certain constraints, e.g., symmetries, allowing them to significantly improve performance compared to prior methods by incorporating hard constraints in the model itself instead of soft data constraints. A key finding of these approaches is that including *inductive* biases based on domain knowledge, e.g., SPH theory, significantly improves these models.**

The present paper aims to demonstrate a physically motivated Machine Learning Model inherently built around symmetries and incorporates several insights from classical numerical approaches to improve existing methods. These insights include higher-order numerical interpolation, e.g., Chebyshev polynomials, SPH-informed kernel choices, e.g., Wendland kernel functions and coordinate system transformations. By building the Machine Learning Model directly upon higher-order basis functions with inherent symmetries, i.e., Fourier-series terms, the learned behavior of these models is also smoother and more physically plausible [3]. Several key aspects of our SPH simulation for data generation were also used to train our neural networks.

We furthermore demonstrate that although these Machine Learning Models are trained solely on data-driven metrics, i.e., comparisons of particle position trajectories, they also exhibit much better performance concerning emergent properties, such as divergence-freedom, despite not being trained on any explicit physical constraints or loss terms. Finally, we demonstrate how these networks are also improving upon the performance of prior Machine Learning Models in the two-dimensional Taylor-Green Vortex validation case and how using this as a one-shot training setup yields neural networks that perform well in general simulation cases not seen in training.

I. Introduction

Numerical solutions of Partial Differential Equations (PDEs) have been a long studied and essential part of many scientific fields, e.g., Computational Fluid Dynamics (CFD) [4], with much of this research focused on improving the accuracy and performance of classic numerical solvers [5]. Building such an accurate and performant solver relies on a vast amount of prior knowledge, e.g., which solver and time-stepping scheme to use and which conservation laws to enforce explicitly and implicitly [6], which is not easily acquired. On the other hand, neural solvers exist on a spectrum ranging from methods that learn purely from data [1], [7] to methods that include many inductive biases to enforce difficult-to-learn aspects, e.g., rotational invariance [8]–[10], momentum conservation [2] and symmetry [11].

Regarding Neural Solvers within CFD, networks run on either Eulerian or Lagrangian method-based data or directly solve problems in coordinate space using Physics Informed Neural Networks (PINNs) [12]. However, these are fundamentally different from data-based approaches and beyond our scope. For structured Eulerian simulations, e.g., using cartesian grids, Convolutional Neural Networks (CNNs) have found broad adoption [13]–[15]; however, applying CNNs

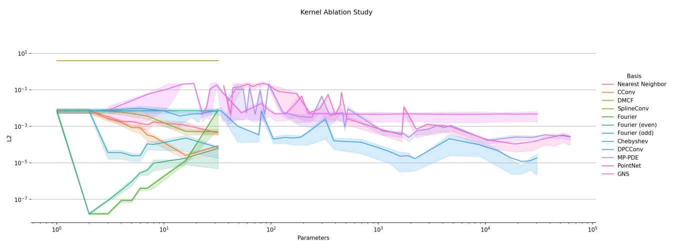

Fig. 3: This figure shows how different neural network setups for learning a simple 1D SPH kernel function behave based on the number of parameters. Note that MP-PDE, PointNet, and GNS refer to methods built on MLPs, whereas all other methods are variants of continuous convolutions. The L_2 error shown here is the mean error across all particles in a 1D SPH Simulation.

to unstructured grids is not directly possible. Graph Neural Networks (GNNs) [16], [17] approaches have been popular for unstructured grids, e.g., *GNS* [18], with cell centers treated as vertices and cell faces as graph edges. These GNNs have found broad application for various PDEs, e.g., Burger's equation, for unstructured grids and particle-based systems [19], [20]. *Continuous Convolutions* (CConvs) [21] is a subset of GNNs, where interactions are based solely on the relative position of cells or particles. These continuous approaches are especially suited for Lagrangian simulations, as particle positions change continuously over time. CConvs often use convolutional filters built using interpolation approaches instead of using Multilayer-Perceptrons (MLPs), as these filters have inherently valuable properties, such as smoothness [22] and antisymmetry [2]. The reliance of PDEs on terms such as spatial gradients means that these inductive biases can free up computational resources. Consequently, CConv approaches learn more accurate solutions than general GNNs for an equivalent number of parameters in physical problems [2].

II. Background and Related Work

Before discussing our recent results, we will provide an overview of prior work and relevant background information. For this, we will first discuss a general definition of Neural Network Time Integrators (NNTIs) in Section II-A, followed by an overview of relevant baseline SPH methodologies in Section II-B and prior work regarding machine learning for simulations in Section II-C.

A. *Neural Network Time Integration*

At a given time t with particles at position $\mathbf{x}^t \in \mathbb{R}^d$, with d spatial dimensions, and corresponding physical quantities $\mathbf{q}^t \in \mathbb{R}^d$, the goal of a physics simulation is to compute a new set of positions and quantities at a new time-point $t^\star$, i.e., $\mathbf{x}^{t^\star}$ and $\mathbf{q}^{t^\star}$, subject to a PDE $\delta^t \mathbf{u} = F(t, \mathbf{x}, \mathbf{u}, \delta^x \mathbf{u}, \delta^{xx}\mathbf{u}, \dots)$, where $\mathbf{u}$ is solved for. Based on the PDE, a solver now computes an update for the particle positions, $\Delta\mathbf{x}^t$, and particle quantities, $\Delta\mathbf{q}^t$, for a given discrete timestep Δt, which yields:

$$\mathbf{x}^{t+\Delta t} = \mathbf{x}^t + \Delta\mathbf{x}^t; \quad \mathbf{q}^{t+\Delta t} = \mathbf{q}^t + \Delta\mathbf{q}^t. \tag{1}$$

Within the SPH field, we now distinguish two kinds of solvers as those either using a summation-based approach, i.e., where particle density is recomputed every timestep [23], and solvers where density is integrated over time [24]. For the former, $\mathbf{q} = \mathbf{v}$, i.e., the only important quantity to integrate is velocity. In contrast, for the latter $\mathbf{q} = [\mathbf{v}, \rho]$, i.e., velocity and density need to be integrated. Furthermore, for simulation schemes involving particle shifting, $1/\Delta t \Delta\mathbf{x}^t \neq \Delta\mathbf{v}^t$. Existing neural solvers [1], [2] generally ignore particle shifting and simplify this problem by inferring the velocity update from the position update, i.e., $\Delta\mathbf{v}^t = \Delta\mathbf{x}^t/\Delta t$. Including an initial velocity $\mathbf{v}^t$ and external constant forces $\mathbf{a}^t$, such as gravity, yields our general learning task as

$$\mathbf{x}^{t+\Delta t} = \mathbf{x}^{t,\prime} + G(\mathbf{x}^t, \mathbf{f}^t, \Theta); \quad \mathbf{x}^{t,\prime} = \mathbf{x}^t + \Delta t\mathbf{v}^t + \Delta t^2\mathbf{a}^t, \tag{2}$$

where G is a Neural Network with parameters Θ and a feature vector $\mathbf{f}$. Note that the initial velocity $\mathbf{v}^t$, due to the inference of velocity from position updates, is not the actual instantaneous velocity of the particles but is defined via $\mathbf{v}^t = \frac{\mathbf{x}^t - \mathbf{x}^{t-\Delta t}}{\Delta t}$. The minimization task is then given as

$$\min_{\Theta} L(\mathbf{x}^{t,\prime} + G(\mathbf{x}^t, \mathbf{f}^t, \Theta), \mathbf{y}^{t+\Delta t}), \tag{3}$$

where $\mathbf{y}^{t+\Delta t}$ are known, ground-truth positions from a given dataset and L being a loss function, commonly chosen as L^2. In the following, we refer to solvers of this form as Neural Network Time Integrators (NNTIs). Note that the timestep Δt and associated time-stepping scheme do not need to be the same for ground-truth simulation and NNTI. For example, it is possible to predict multiple ground truth-simulation steps simultaneously using multi-stepping or temporal-bundling [1], [19], with many different datasets for SPH-based simulations being available [1]–[3], [25].

B. *Classical Solvers*

We utilize the Smoothed Particle Hydrodynamics (SPH) method as the basis for our test cases and to inform some of our inductive biases. SPH is a Lagrangian simulation technique initially introduced in an astrophysics context [26] but has found broad application in various fields, e.g., in CFD [27] and Computer Graphics [28], [29]. At the heart of SPH are interpolation operations $\langle A\rangle$ of quantities A carried by particles, with positions $\mathbf{x}$, mass m and density ρ, using a Gaussian-like kernel function W as

$$\langle A_i\rangle = \sum_{j\in\mathcal{N}_i} A_j \frac{m_j}{\rho_j} W(\mathbf{x}_j - \mathbf{x}_i, h), \tag{4}$$

where h is the support radius and $\mathcal{N}_i$ being the neighbors of i, including itself, which are all particles closer than h, see [30] for a broader overview. This SPH operation can be seen as a message-passing step using the edge lengths and features of the adjacent vertices with a summation message-gathering operation. Within SPH, many solvers exist for various problems, and choosing the correct one for a problem can be challenging as they are vastly different. As our test cases involve compressible, weakly-compressible, and incompressible SPH solvers to generate the data, understanding their background and requirements is valuable.

For compressible simulations, SPH utilizes either an explicit pressure formulation using a compressible Equation of State, as do we, or using Riemannian solvers [31], which would be an important direction for future research due to their complexity. For weakly compressible simulations, the most commonly utilized technique is the δ-SPH method [32], using explicit pressure forces combined with diffusion models for velocity and density terms using very small timesteps the δ^+-SPH [24] variant that we use for our data generation further expanding this approach. Finally, for incompressible SPH [23], [28], the simulation revolves around solving a Pressure Poisson Equation using an implicit solver using many iterations per timestep.

C. Neural Networks

Many methods have been proposed to solve PDEs using machine learning techniques [7], [33], [34], where Physics-Informed Neural Networks (PINNs) are among the most prominent approaches to solving such PDEs [12]. PINNS are coordinate networks that learn the solution of a PDE without first requiring a discretization using continuous residuals. While powerful, these methods have many drawbacks, including difficulties in training and the fact that they cannot be applied to discrete simulation data. Another prominent machine learning technique is point-cloud classification and segmentation, e.g., *PointNet* [35]. However, these approaches are unsuited to Lagrangian simulations as particles are much more interconnected than point clouds. Neural Operators have recently also found research interest in solving PDEs [36]–[38]. However, applying these approaches to irregular grids or particle-based simulations is impossible. On the other hand, Graph Neural Networks (GNNs) [18] naturally map to simulations as most can be seen as message passing.

Graph Neural Networks can be directly applied to SPH simulations, where each particle is a vertex, and the particle neighborhoods describe the graph connectivity. GNNs use the graph to generate messages using just the vertex information [35], edge information [21], edge and vertex [18], or edge, vertex, and additional feed-through features collected on vertices using pooling operations [39]. These collected messages are then either used directly, as new features on the vertices, or combined with existing vertex features. Further operations, such as input encoding and output decoding, have also been proposed [18]. The message processing is performed using MLPs with relatively shallow but broad hidden architectures, e.g., 2 hidden layers with 128 neurons. Finally, there also has been significant research regarding symmetries and equivariance in GNNs [9], [39], [40].

Continuous Convolutions are a subset of GNNs and utilize only coordinate distances as inputs to the filter functions, similar to SPH kernel functions [1]. These filter functions are then combined with the features of adjacent vertices to form the messages. While this imposes an inductive bias, it can

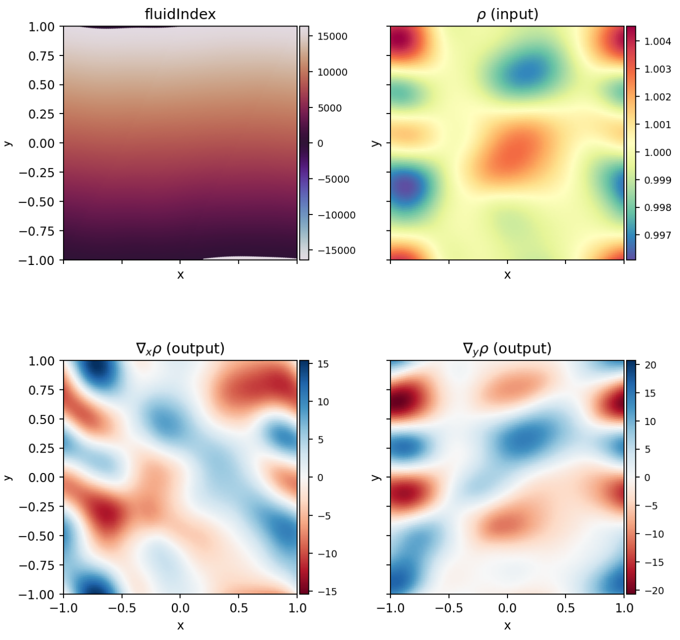

Fig. 4: A common task in an SPH simulation is the computation of gradients. This figure shows an early timepoint in a simulation where a density field (top right) is used to compute a spatial density gradient (bottom row). This test case is challenging for neural networks to learn accurately due to the magnitude of quantities involved.

make learning the problem more manageable. The coordinate distances can then be processed using an MLP [21] or other function interpolation techniques, e.g., linear interpolation [1] or spline-based interpolation [22]. Several extensions of these approaches have been proposed for physical simulations, e.g., using antisymmetry [2] to conserve particle momentum, or both symmetry and antisymmetry [3].

Fourier and Chebyshev Methodes: Fourier Neural Operators (FNOs) [36] learn physical simulations by transforming a given spatial discretization into a spectral representation using an FFT. By applying the learning task in the spectral domain, these operators can learn spatially invariant behavior but are limited to regularly sampled data due to their reliance on FFTs. Fourier encodings have also been applied in image classification and segmentation to make networks less spatially dependent [38]. Our recent work [3] combined Fourier terms with CConvs to learn Lagrangian Fluid Mechanics.

III. Experiments

After discussing all relevant related work and prior methods, we will now discuss some of the various insights we have gained in the past. These primarily focus on how inductive biases, i.e., applying domain-specific knowledge to a machine learning model, can influence and, potentially, improve the performance of a neural network. However, we will also discuss some insights highlighting that machine learning models *can* benefit from inductive biases but that applying counter-intuitive biases *can* lead to similar or improved performance.

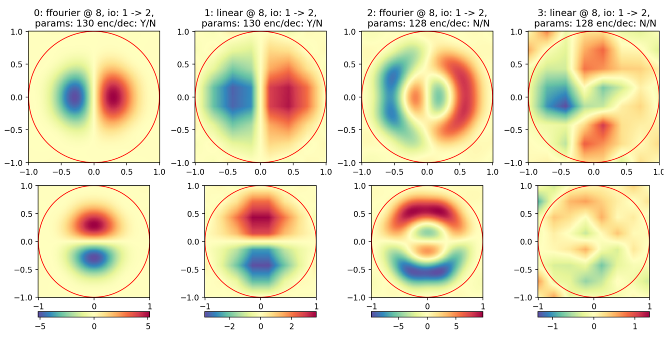

Fig. 5: This figure shows how different network formulations learn an SPH gradient operator for the problem shown in Fig. 4. Color indicates the value of the learned convolution kernel, with the top row showing the x-coordinate filter and the bottom row showing the y-component of the gradient.

Finally, we will also discuss how standard CFD methodologies, such as convergence studies, can help understand the behavior of machine learning models. In this section, we will begin by discussing how symmetries built into the network, e.g., ensuring that all operations are symmetric or antisymmetric, can be beneficial, see Sec. III-A. Next, we will discuss how including standard SPH kernel functions as so-called window functions, see [3], can be helpful to improve the behavior of some methods but can also lead to worse performance for other methods, see Sec. III-B. Following this, we will discuss how we can use convergence studies to understand how different parameters influence network performance, see Sec. III-D, and how physical quantities might need to be scaled to achieve good performance when used in machine learning models, see Sec. III-C. Finally, we will conclude by discussing how Particle Shifting Techniques (PSTs), can significantly improve the behavior of machine learning models with little overhead, see Sec. III-E.

A. *Builtin Symmetries*

An essential aspect of many physical systems is symmetries, e.g., Newton's third law, which results in many vital equivariances and conservation laws in the respective systems, e.g., conservation of momentum. While it is possible to rely solely on the learning task to ensure that conservation of momentum is adhered to, based purely on the observed data, using such a simulation in an engineering context might be difficult or impossible, as there is no guarantee of conservation and neural networks, generally, do not provide certainty measures for metrics unseen during training.

Prior work often [1], [18] relied solely on data-driven adherence to these symmetries instead of enforcing them directly as an inductive bias. While the results of these methods were noteworthy, their lack of conservation led to potentially severe instabilities. The work by Prantl et al. [2] included an inductive bias for antisymmetric convolutions that results in a guarantee of conservation of momentum within a learned simulation. While this was an important addition, many operations within an SPH simulation are also reliant on radially symmetric interactions, i.e., a kernel density estimate is radially symmetric instead of antisymmetric, consequently restricting the neural network from learning only antisymmetric terms, prevents it from learning any non-antisymmetric term, as shown recently in our prior work [3].

In this recent work, we performed a simple experiment using a one-dimensional pseudo-compressible SPH simulation with an initially random density profile (for more details, see [3]). As one of the experiments we evaluated how well a network is able to learn an SPH kernel function, see Fig. 3, where, in addition to the previous results [3], we also considered MLP based GNNs, such as *DPCConv* [21], *PointNet* [35], *GNS* [18] and *MP-PDE* [19]. For this experiment, we found a clear benefit of using CConvs over MLP-based approaches, i.e., *GNS* and *MP-PDE* performed significantly worse for equal parameter counts relative to the MLP-based CConv approach of Wang et al. [21]. This approach, in turn, required orders of magnitudes more parameters for equal performance than filter function-based CConvs. Furthermore, the results clearly show that the approach of Prantl et al [2] could not learn this behavior at all. In contrast, methods that incorporate symmetries and antisymmetries, i.e., Fourier terms [3], were able to learn this behavior accurately.

Overall, these results highlight the strength of including inductive biases, i.e., using a Fourier-based CConv approach results in 2 orders of magnitude lower error scores than using an MLP-based CConv approach while using four orders of magnitude fewer parameters. Furthermore, these results indicate that while using such biases can be very beneficial, they can also make it impossible for a network to learn a specific task, i.e., the DMCF approach [2] is well suited for learning physical force interactions but is incapable of learning symmetric interactions. Consequently, the usage of inductive biases needs to be carefully evaluated for a specific given task, and using fewer, less restrictive biases may be more beneficial early on when it is not clear yet which biases perform best.

B. *Kernel Functions*

Our recent work [3] evaluated a two-dimensional bounded flow problem with an initially random velocity field. One of the main observations we made regarding the stability of network predictions was related to an imposed noise on the predicted velocity field. This noise was significantly more apparent in some methods compared to others. While these methods might be stable under such noise for some time, the velocity field was affected considerably. This noise was most apparent in the divergence of the velocity field, which should be close to zero due to this being a divergence-free initial flow with a weakly compressible simulation scheme. However, some methods resulted in significant divergence.

An important observation that we made in these experiments was that the noise was strongly dependent on the formulation of the convolution filter, i.e., a smooth convolution filter resulted in lower noise than a discontinuous filter, and how the

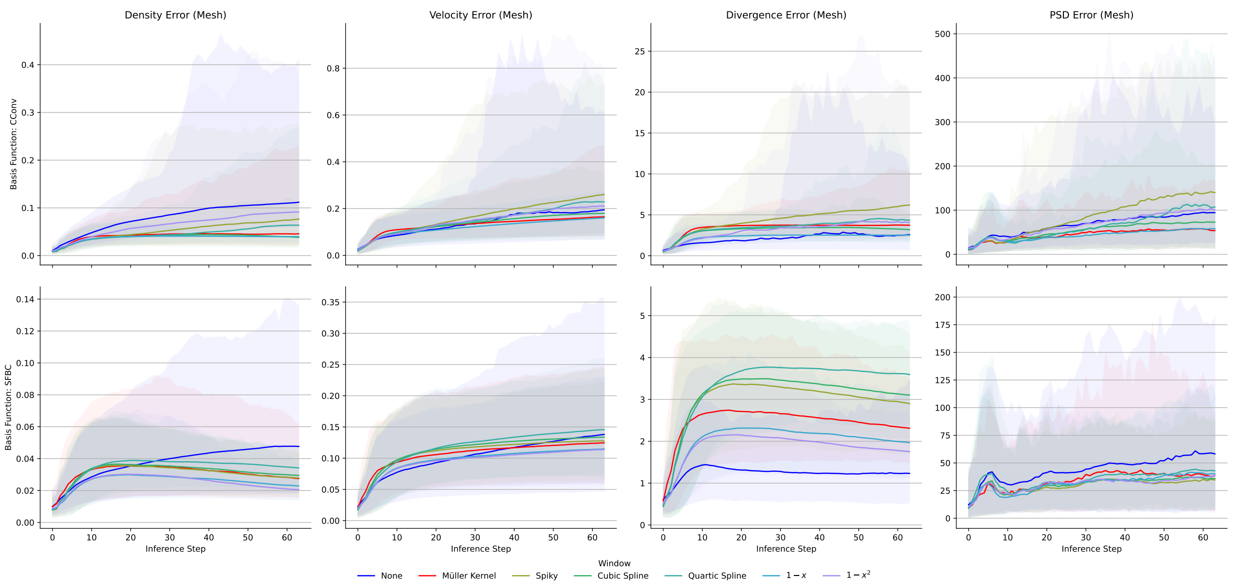

Fig. 6: This figure shows how different physical errors, e.g., divergence, behave for different Continuous Convolution approaches when using different window functions as an additional inductive bias. Note that the network was not trained on any of these error metrics and that the underlying 2D simulation is weakly compressible, i.e., there should be little to no divergence.

spatial sampling frequency of the filter related to the noise, i.e., a method with a higher sampling frequency resulted in higher imposed noise. In general, SPH simulations are based on kernel functions that are smooth Gaussian-like functions with compact support, which can also be applied to CConv architectures. These so-called window functions can ensure that the learned convolution filters are significantly smoother than otherwise, which results in a reduced noise for otherwise noisy methods, see Fig. 6.

However, an important observation, in line with the results from the prior subsection, here was that applying such an inductive bias to an already smooth basis, e.g., a Fourier series, often resulted in a worse overall performance as the imposition of this bias limited what the network could learn. Again, these results reinforce that inductive biases can substantially benefit a machine learning model. In many cases, they can equally make methods perform significantly worse. Ideally, a machine learning model should not be restricted in what it can learn, as this will limit the ultimate performance of the model; however, for smaller networks and current state-of-the-art approaches, imposing inductive biases can still yield significantly improved performance for equal parameter counts.

C. Scaling

For our following evaluations, we set up a new dataset built on random, periodic flows in 2D; see Fig. 7. For these simulations, we utilized an initial rectilinear grid of 128×128 particles with both an incompressible DFSPH simulation scheme [23] and a δ^+-based weakly compressible SPH simulation, both with particle shifting [24], fixed time-stepping of $\Delta t = 0.1ms$, Runge-Kutta 4 integration, and a Reynolds number of $\mathrm{Re} \approx 2000$. For the DFSPH-based simulations, we re-evaluated the particle density in every timestep using an SPH summation. In contrast, for the weakly-compressible scheme, we integrated density over time a speed of sound such that the expected compression was $\approx 0.1\%$, with a rest density of $\rho_0 = 1$. We utilized an octave-noise-based approach for the initial velocity fields using simplex noise [41]. For our dataset, we generated 16 simulations of various octave noise configurations. However, we base our evaluations here primarily on the lowest frequency sample, as the results are reflected well in other cases.

Analogous to the experiment in Sec. III-A and [3], we first considered a simple toy problem of learning an SPH-based density gradient, using a formulation that is accurate for constant fields [42], given the density field, see Fig. 4. However, simply feeding the data into a neural network did not yield good results, see Fig. 5 (right), as the magnitudes of quantities were challenging for a network to handle. In general, neural networks perform well for inputs that are *normalized* in some sense, which commonly refers to input quantities being normalized such that they have mean 0 with a standard deviation of 1, however, the density field provided to the network in this case had mean 1 with a standard deviation of 0.002. In practice, this meant that the network could not differentiate different inputs and converged towards producing a fixed output that fit the mean of the output distribution.

A common approach in machine learning to avoid this behavior is to normalize the data, which can either be done a priori on the dataset or as part of the network with a learn normalization layer. We now added an encoder layer, built using a single group norm layer, to learn a set of scaling coefficients to ensure the inputs have a more beneficial distribution. This encoder resulted in the network being readily capable of learning an appropriate SPH kernel gradient, see Fig. 5 (left), where we can again observe how a network with builtin symmetries and smoothness, i.e., *SFBC* [3] produces a significantly more accurate model than one without. This case highlights how networks can often struggle to learn anything meaningful simply because the data provided does not have *high enough* information content, which can usually be resolved through normalization of the input data using, for example, normalization layers.

D. Convergence

Within CFD research, a widespread way to analyze a solver is to perform a convergence study, i.e., to evaluate how an increase in resolution results in an increase in accuracy. Performing such a study directly for most machine learning models would be very challenging as most state-of-the-art models are trained only for one specific resolution and generally cannot generalize well to other resolutions.

We performed this evaluation on our DFSPH based dataset for different CConv architectures, see Fig. 8. In this study, we changed the number of message-passing operations features per layer, added input and output decoders, and added a vertex and edge MLP transformation. For this specific setup, the results indicate that the method with 4 basis terms per filter (blue, Fig. 8) has already *converged*, even for small parameter count,s and no change of other parameter makes any difference. It is important to remember that converging in this case does not mean that the network accurately reflects the ground truth simulation, but only that the network does not improve anymore with more parameters. Contrary to the results for 4 basis terms, for 8 basis terms, we see an increase in performance with an increase in parameters, which is most evident for the case with both an input encoder and output decoder (Fig. 8, right).

This result, so far, is not that surprising as, generally, adding more parameters will improve the results. However, the most interesting result relates to adding an MLP layer to transform the edge features of the graph, i.e., instead of performing a continuous convolution on the distance between particle pairs, the continuous convolution operator is applied within some transformed and learned latent space instead. This transformation, generally, does not make much physical sense as the motivation of the CConv operation heavily relies on convolution filters based on particle distances and for a network without an input and output decoder (Fig. 8, left), such networks generally do not yield any beneficial behavior, as expected. However, for some cases, especially those using both vertex and edge MLP layers, the networks do show some learning behavior, albeit at a performance worse than other networks of similar size. These results, once again, indicate that while some inductive biases can be very beneficial, i.e., using particle distances instead of some latent values, can result in a useful behavior, not using such a bias can still result in a network that can learn a given task, even though it may not be intuitive.

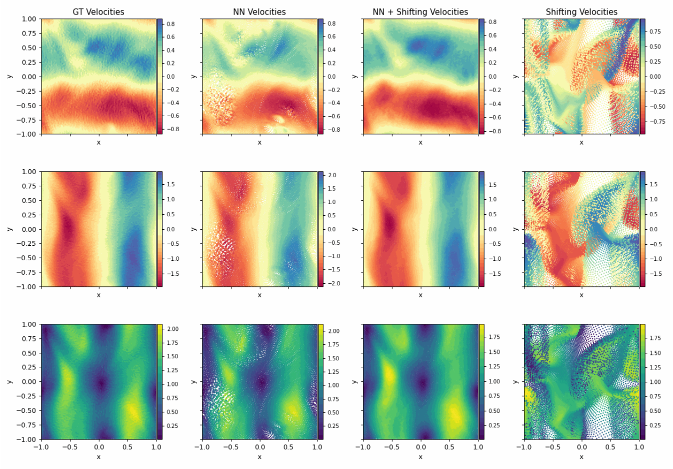

Fig. 7: This figure shows how a ground truth trajectory (left column) behaves after 20 timesteps from an initial simulation time point compared to a purely machine learning-based model (middle-left column) and a machine learning model augmented with a particle shifting technique (PST) (middle-right column) as well as only using a PST. The top row shows x-velocity, the middle row shows y-velocity, and the bottom row shows velocity magnitude.

E. Shifting

As a final experiment, we considered the influence of particle disorder on neural network performance. Within classic SPH solvers, particle shifting has long been understood as an essential aspect of the stability of a simulation [24], [43], with many SPH simulations applying a variety of Particle Shifting Techniques (PSTs) to stabilize a simulation. This issue occurs similarly in neural network-based simulations but can readily become more challenging. Any high-quality baseline simulation used for training a neural network will have a low particle disorder, either by using a PST or other approaches, such as background pressure terms. Consequently, the neural network during training has never seen strong particle disorder and cannot adequately handle these unseen distributions, leading to strong instabilities, see Fig. 7. To resolve this problem, we include a PST during the application of a network, i.e., at inference time, to restore an isotropic particle distribution and stabilize the simulation. While this improves the neural network's stability, it is not enough to keep the simulation stable in all cases, mainly as we only utilized a simple explicit PST [24]. Further work on including the PST during training is still required but beyond our scope.

IV. Conclusion

In conclusion, we performed a set of experiments showing how inductive biases from classical CFD research can strongly influence and improve the performance of neural networks. In this regard, we found strong benefits for including symmetries, smoothness, and particle shifting. However, we also found that

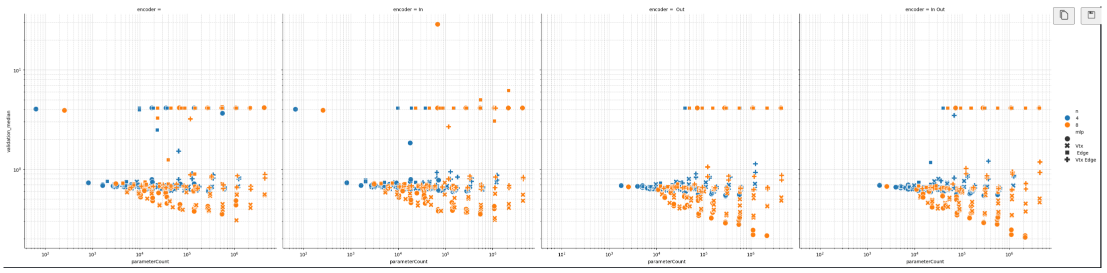

Fig. 8: This figure shows how different networks perform for a Taylor Green Vortex-like simulation setup. Colors indicate the number of basis terms in a continuous convolution. This figure shows how there are many parameters, some physical, such as resolution, and some non-physical, such as edge MLP transformations, and how changing these Hyperparameters leads to a difference in convergence.

some inductive biases, such as window functions, may deteriorate the performance of the learned simulations and that some counterintuitive approaches, such as relative positions being transformed into latent spaces, can still improve, or at least not degrade, the performance of a neural network. Overall, this points into an exciting direction for future research as the combination of inductive biases can allow for much better network performance with limited resources, however, it is unclear how these biases would perform with foundational scale models, i.e., models with hundreds of millions of parameters.

References

[1] B. Ummenhofer, L. Prantl, N. Thuerey, and V. Koltun, “Lagrangian fluid simulation with continuous convolutions,” in *International Conference on Learning Representations*, 2019.

[2] L. Prantl, B. Ummenhofer, V. Koltun, and N. Thuerey, “Guaranteed conservation of momentum for learning particle-based fluid dynamics,” *Adv. Neural Inf. Process. Syst.*, vol. 35, pp. 6901–6913, 2022.

[3] R. Winchenbach and N. Thuerey, “Symmetric basis convolutions for learning lagrangian fluid mechanics,” in *12th International Conference on Learning Representations, ICLR*, 2024.

[4] C. Zhang, M. Rezavand, and X. Hu, “A multi-resolution SPH method for fluid-structure interactions,” *J. Comput. Phys.*, vol. 429, p. 110 028, 2021.

[5] R. Vacondio, C. Altomare, M. De Leffe, *et al.*, “Grand challenges for Smoothed Particle Hydrodynamics numerical schemes,” *Comput. Particle Mech.*, vol. 8, no. 3, pp. 575–588, May 2021.

[6] S. B. Pope, *Turbulent Flows*. Cambridge University Press, 2000. DOI: 10.1017/CBO9780511840531.

[7] J. Morton, A. Jameson, M. J. Kochenderfer, and F. Witherden, “Deep dynamical modeling and control of unsteady fluid flows,” in *Advances in Neural Information Processing Systems*, 2018.

[8] V. G. Satorras, E. Hoogeboom, and M. Welling, “E(n) equivariant graph neural networks,” in *Proceedings of the 38th International Conference on Machine Learning*, M. Meila and T. Zhang, Eds., 18–24 Jul 2021, pp. 9323–9332.

[9] M. Lino, S. Fotiadis, A. A. Bharath, and C. D. Cantwell, “Multi-scale rotation-equivariant graph neural networks for unsteady eulerian fluid dynamics,” *Phys. Fluids*, vol. 34, no. 8, 2022.

[10] N. Thomas, T. E. Smidt, S. Kearnes, *et al.*, “Tensor field networks: Rotation- and translation-equivariant neural networks for 3d point clouds,” *CoRR*, vol. abs/1802.08219, 2018.

[11] R. Wang, R. Walters, and R. Yu, “Incorporating symmetry into deep dynamics models for improved generalization,” in *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, OpenReview.net, 2021.

[12] S. Cai, Z. Mao, Z. Wang, M. Yin, and G. E. Karniadakis, *Physics-informed neural networks (pinns) for fluid mechanics: A review*, 2021.

[13] J. Tompson, K. Schlachter, P. Sprechmann, and K. Perlin, “Accelerating eulerian fluid simulation with convolutional networks,” in *Proceedings of Machine Learning Research*, 2017, pp. 3424–3433.

[14] Y. Bar-Sinai, S. Hoyer, J. Hickey, and M. P. Brenner, “Learning data-driven discretizations for partial differential equations,” *Proc. National Acad. Sci.*, vol. 116, no. 31, pp. 15 344–15 349, 2019.

[15] N. Thuerey, K. Weißenow, L. Prantl, and X. Hu, “Deep learning methods for reynolds-averaged navier–stokes simulations of airfoil flows,” *AIAA J.*, vol. 58, no. 1, pp. 25–36, 2020.

[16] T. Pfaff, M. Fortunato, A. Sanchez-Gonzalez, and P. W. Battaglia, “Learning mesh-based simulation with graph networks,” in *9th International Conference on Learning Representations, ICLR 2021*, 2021.

[17] J. Zhou, G. Cui, S. Hu, *et al.*, “Graph neural networks: A review of methods and applications,” *AI Open*, vol. 1, pp. 57–81, 2020. DOI: 10.1016/j.aiopen.2021.01.001.

[18] A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, and P. Battaglia, “Learning to simulate complex physics with graph networks,” in *Interna-*

tional conference on machine learning, PMLR, 2020, pp. 8459–8468.

[19] J. Brandstetter, D. E. Worrall, and M. Welling, "Message passing neural PDE solvers," in *The Tenth International Conference on Learning Representations, ICLR*, OpenReview.net, 2022.

[20] K. R. Allen, T. Lopez-Guevara, K. L. Stachenfeld, *et al.*, "Inverse design for fluid-structure interactions using graph network simulators," in *Advances in Neural Information Processing Systems*, 2022.

[21] S. Wang, S. Suo, W.-C. Ma, A. Pokrovsky, and R. Urtasun, "Deep parametric continuous convolutional neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2589–2597.

[22] M. Fey, J. E. Lenssen, F. Weichert, and H. Müller, "Splinecnn: Fast geometric deep learning with continuous b-spline kernels," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 869–877.

[23] J. Bender and D. Koschier, "Divergence-free smoothed particle hydrodynamics," in *Proceedings of the 14th ACM SIGGRAPH/Eurographics symposium on computer animation*, ACM, 2015, pp. 147–155.

[24] P. Sun, A. Colagrossi, S. Marrone, M. Antuono, and A. Zhang, "Multi-resolution delta-plus-sph with tensile instability control: Towards high reynolds number flows," *Comput. Phys. Commun.*, vol. 224, pp. 63–80, 2018.

[25] A. Toshev, G. Galletti, F. Fritz, S. Adami, and N. Adams, "Lagrangebench: A lagrangian fluid mechanics benchmarking suite," *Adv. Neural Inf. Process. Syst.*, vol. 36, 2024.

[26] J. J. Monaghan, "Simulating free surface flows with sph," *J. computational physics*, vol. 110, no. 2, pp. 399–406, 1994.

[27] J. J. Monaghan, "Smoothed particle hydrodynamics," *Reports on progress physics*, vol. 68, no. 8, p. 1703, 2005.

[28] M. Ihmsen, J. Cornelis, B. Solenthaler, C. Horvath, and M. Teschner, "Implicit incompressible sph," *IEEE transactions on visualization computer graphics*, vol. 20, no. 3, pp. 426–435, 2013.

[29] M. Macklin and M. Müller, "Position based fluids," *ACM Trans. on Graph. (TOG)*, vol. 32, no. 4, p. 104, 2013.

[30] D. Koschier, J. Bender, B. Solenthaler, and M. Teschner, "Smoothed particle hydrodynamics techniques for the physics based simulation of fluids and solids," in *40th Annual Conference of the European Association for Computer Graphics, Eurographics 2019 - Tutorials*, W. Jakob and E. Puppo, Eds., Eurographics Association, 2019, pp. 1–41. DOI: 10.2312/egt.20191035.

[31] K. Puri and P. Ramachandran, "Approximate riemann solvers for the godunov sph (gsph)," *J. Comput. Phys.*, vol. 270, pp. 432–458, 2014.

[32] S. Marrone, M. Antuono, A. Colagrossi, G. Colicchio, D. Le Touzé, and G. Graziani, "δ-sph model for simulating violent impact flows," *Comput. Methods Appl. Mech. Eng.*, vol. 200, no. 13-16, pp. 1526–1542, 2011.

[33] P. Battaglia, R. Pascanu, M. Lai, D. Jimenez Rezende, *et al.*, "Interaction networks for learning about objects, relations and physics," *Adv. neural information processing systems*, vol. 29, 2016.

[34] K. Um, R. Brand, Y. (Fei, P. Holl, and N. Thuerey, "Solver-in-the-loop: Learning from differentiable physics to interact with iterative pde-solvers," in *Advances in Neural Information Processing Systems 33*, 2020.

[35] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 652–660.

[36] Z. Li, N. B. Kovachki, K. Azizzadenesheli, *et al.*, "Fourier neural operator for parametric partial differential equations," in *9th International Conference on Learning Representations*, 2021.

[37] J. Guibas, M. Mardani, Z. Li, A. Tao, A. Anandkumar, and B. Catanzaro, "Efficient token mixing for transformers via adaptive fourier neural operators," in *International Conference on Learning Representations*, 2021.

[38] Y. Li, S. Si, G. Li, C.-J. Hsieh, and S. Bengio, "Learnable fourier features for multi-dimensional spatial positional encoding," *Adv. Neural Inf. Process. Syst.*, vol. 34, pp. 15 816–15 829, 2021.

[39] J. Brandstetter, M. Welling, and D. E. Worrall, "Lie point symmetry data augmentation for neural pde solvers," in *International Conference on Machine Learning*, 2022.

[40] A. P. Toshev, G. Galletti, J. Brandstetter, S. Adami, and N. A. Adams, "Learning lagrangian fluid mechanics with e (3)-equivariant graph neural networks," in *International Conference on Geometric Science of Information*, Springer, 2023, pp. 332–341.

[41] R. Bridson, J. Houriham, and M. Nordenstam, "Curlnoise for procedural fluid flow," *ACM Trans. Graph.*, vol. 26, no. 3, 46–es, Jul. 2007, ISSN: 0730-0301.

[42] D. J. Price, "Smoothed particle hydrodynamics and magnetohydrodynamics," *J. Comput. Phys.*, vol. 231, no. 3, pp. 759–794, 2012.

[43] R. Vacondio, C. Altomare, M. De Leffe, *et al.*, "Grand challenges for smoothed particle hydrodynamics numerical schemes," *Comput. Particle Mech.*, vol. 8, no. 3, pp. 575–588, 2021.