Leveraging Stochastic Predictions of Bayesian Neural Networks for Fluid Simulations

Maximilian Mueller [1 2]   **Robin Greif** [2 3]   **Frank Jenko** [2 3]   **Nils Thuerey** [3]

Abstract

We investigate uncertainty estimation and multi-modality via the non-deterministic predictions of Bayesian neural networks (BNNs) in fluid simulations. To this end, we deploy BNNs in three challenging experimental test-cases of increasing complexity: We show that BNNs, when used as surrogate models for steady-state fluid flow predictions, provide accurate physical predictions together with sensible estimates of uncertainty. Further, we experiment with perturbed temporal sequences from Navier-Stokes simulations and evaluate the capabilities of BNNs to capture multimodal evolutions. While our findings indicate that this is problematic for large perturbations, our results show that the networks learn to correctly predict high uncertainties in such situations. Finally, we study BNNs in the context of solver interactions with turbulent plasma flows. We find that BNN-based corrector networks can stabilize coarse-grained simulations and successfully create multimodal trajectories.

1. Introduction

Even though Bayesian neural networks (BNNs) have been studied for a long time in the Machine Learning community (Hinton & van Camp, 1993; MacKay, 1992b), they have only recently received increased attention in the wild. While conventional, non-Bayesian deep learning techniques, that have mostly been used in fluid simulation setups, provide point estimates, BNNs allow to obtain stochastic predictions, since they learn a distribution over the network's weight parameters. Exploring to which extent those stochastic predictions can be exploited in the context of fluid simulations is the central goal of this work. In particular, we identify and investigate two use-cases of a combination of BNNs with fluid simulations: uncertainty estimation and multi-modal synthesis.

A central motivation for the use of BNNs is the estimation of uncertainty (MacKay, 1992a). In the context of fluid simulations, it is an open question whether neural networks can successfully provide sensible uncertainty estimates. This includes uncertainty location on the one hand: If a neural network is for instance deployed as surrogate model to a physical solver, the uncertainty locations in the predictions should correspond to regions that are harder to predict, e.g., more turbulent locations. On the other hand, a key challenge is to relate the predictive performance to the uncertainty that comes along with BNN predictions quantitatively. In practice, this can be done by comparing a suitable measure of predictive performance, such as the mean absolute error (MAE), to a measure of uncertainty, e.g., the standard deviation over repeated predictions. We then expect the standard deviation to correlate with the mean absolute error. Ideally, this way the model can communicate when it is uncertain about a prediction, or when the learning process failed to converge.

Further, fluid models typically provide a deterministic description of an inherently chaotic and stochastic process (Pope, 2000). Many initial conditions lead to bifurcation points where epsilon changes of the flow state can lead to fundamentally different solutions over time (Ko et al., 2008), e.g., a vortex turning left or right. Since conventional neural networks typically act as deterministic predictors that provide point estimates, they are limited in describing such setups. It is therefore interesting to investigate if the stochasticity of the BNN predictions can be exploited in order to resemble multimodal solutions. One particular case of interest is plasma physics transport modelling (Balescu, 2005), one of the key challenges on the path to a practical fusion device (Freidberg, 2008). There, transport is driven by micro-instabilities and the resulting systems are highly turbulent. Consequently, minor changes in the initial conditions can lead to very different solution states after a short period of time, creating a particularly relevant setup to study multimodality.

[1]Department of Computer Science, University of Tübingen, Tübingen, Germany [2]Max Planck Insitute for Plasma Physics, Munich, Germany [3]Department of Computer Science, TU Munich, Germany. Correspondence to: Maximilian Mueller <maximilian.mueller@wsii.uni-tuebingen.de>.

In the following, we assess the performance of BNNs on these tasks in three test-cases. We show that a trained BNN produces meaningful uncertainty estimates for complex fluid simulation scenarios, like Reynolds-averaged Navier-Stokes flow around airfoils, and perturbed buoyancy-driven Navier-Stokes flow. In addition, our experiments demonstrate that BNNs successfully generate varied predictions when working in conjunction with a numerical simulator in the plasma turbulence setup.

2. Related Work

Leveraging data-driven methods in the context of PDE models has been a long-standing goal (Brunton et al., 2016; Bindal et al., 2006; Crutchfield & McNamara, 1987). In the past years, the focus has shifted towards deep-learning approaches. Those were successfully applied to a wide variety of tasks, such as deep learning for reduced models with Koopman theory (Li et al., 2020; Morton et al., 2018), identifying model equations (Raissi et al., 2018; Long et al., 2018), and learned discretizations (Bar-Sinai et al., 2019). Additionally, Tompson et al. (2017) investigated unsupervised learning of corrections while Sirignano & Spiliopoulos (2018) used physics-informed methods by deploying PDE-based loss functions. Other research focused on efficient simulations by learning conservation laws (Cranmer et al., 2020; Greydanus et al., 2019), or aimed at correcting iterative solvers (Hsieh et al., 2019). Turbulence modelling received particular attention, e.g., from Beck et al. (2019), Tracey et al. (2015) and Novati et al. (2021).

Convolutional neural networks, which we will likewise use in our experiments, were used in the context of flow problems as the basis for generative models (Chu & Thuerey, 2017; Kim et al., 2019), or for corrector models (Um et al., 2018; Thuerey et al., 2020). Further, the recent development of geometric deep learning approaches (Bronstein et al., 2021) also impacted fluid flow problems: Mesh-based methods (Pfaff et al., 2020), graph neural networks (Sanchez-Gonzalez et al., 2020) and continuous convolutions (Ummenhofer et al., 2020) were deployed successfully.

Differentiable components and differentiable programming have been leveraged by a variety of recent works (Amos & Kolter, 2017; Innes et al., 2019; Hu et al., 2020; Chen et al., 2019). These approaches enable an end-to-end training which was shown to have advantages in rigid body control (de Avila Belbute-Peres et al., 2018) and advection-diffusion systems (Yin et al., 2021). Within the scope of our work, we will use the differentiable simulator *PhiFlow* (Holl et al., 2020), in particular in a tight integration with neural networks, as suggested by Um et al. (2021).

Early contributions to Bayesian networks can be attributed to MacKay (1992a;b), Hinton & van Camp (1993) and Neal
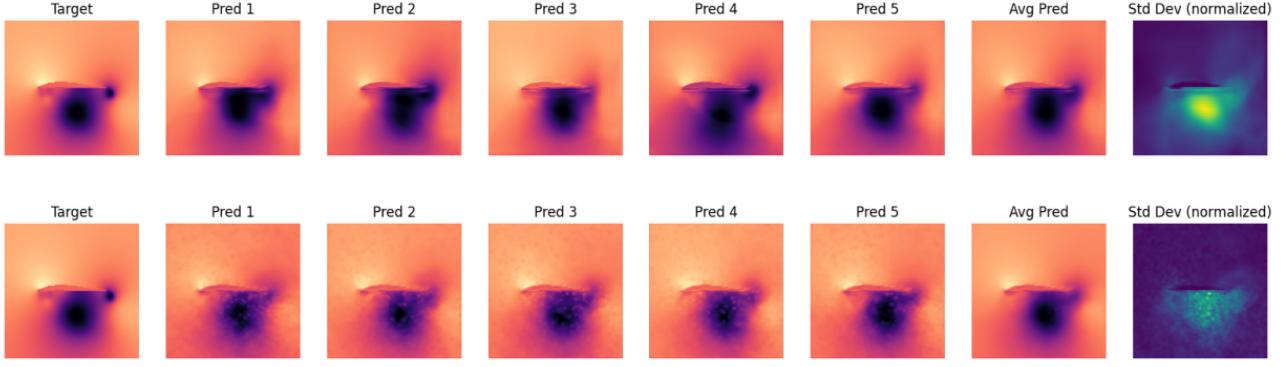
(1996). In this paper, we will use Bayesian networks based on variational inference (Kingma et al., 2015), taking advantage of reparametrization techniques (Kingma & Welling, 2014; Blundell et al., 2015) and Monte-Carlo-based methods (Gal & Ghahramani, 2016; Graves, 2011; Welling & Teh, 2011). In the wild, BNNs have mostly been deployed for segmentation tasks (Badrinarayanan et al., 2016): LaBonte et al. (2020) obtained 3D geometric uncertainties for CT scans, Kwon et al. (2018) performed image segmentation on biomedical data, Deodato et al. (2019) on cellular images and McClure et al. (2019) on brain segmentation tasks. More recently, (Cranmer et al., 2021) leveraged BNNs to predict the dissolution of compact planetary systems. Even though uncertainty quantification has been a long-standing topic for computational fluid dynamics (Roache, 1997), the use of BNNs in this area remains largely unexplored - with a few exceptions like Sun & Wang (2020), who deployed a physics-constrained Bayesian network to reconstruct fluid flow from sparse and noisy data. In contrast to their work, we do not constrain our networks and investigate a broader class of problems and use-cases of BNNs.

3. Background: Bayesian Neural Networks

Bayesian Neural Networks provide stochastic predictions by incorporating the Bayesian paradigm into deep learning. The network weights $\boldsymbol{w}$ are thought to follow a prior distribution $p(\boldsymbol{w})$, which is updated to the posterior distribution $p(\boldsymbol{w}|\boldsymbol{X}, \boldsymbol{Y})$ after observing the data consisting of inputs $\boldsymbol{X}$ and targets $\boldsymbol{Y}$. In this work, we used two variants of BNNs. Both stem from variational inference (Kingma et al., 2015) and aim at minimizing the KL-divergence between true and approximate posterior, or equivalently maximize the evidence lower bound (ELBO):

$$\mathbb{E}_{\boldsymbol{w} \sim q_{\boldsymbol{\theta}}}[\log p(\boldsymbol{Y}|\boldsymbol{w}, \boldsymbol{X})] - \frac{1}{\lambda} D_{\mathrm{KL}}\left(q_{\boldsymbol{\theta}}(\boldsymbol{w}) \| p(\boldsymbol{w})\right) \quad (1)$$

where $q_{\boldsymbol{\theta}}(\boldsymbol{w})$ is the approximate posterior distribution over the weights, which is parametrized by θ. D_{KL} denotes the KL-divergence and $\lambda \geq 1$ is a scaling factor that was empirically shown to improve the performance of BNNs (see e.g. Wenzel et al. (2020)). We choose our noise model such that the log-probability can be written as the mean absolute error, which has shown to lead to good performance in 2d fluid settings (Thuerey et al., 2018), or the mean squared error. Intuitively, the two terms in equation 1 have opposite goals: Maximizing the expected log-probability encourages the variational distribution to fit the data well. Minimizing the KL-divergence, in contrast, forces the approximate posterior distribution to stay close to the prior, which penalizes complex distributions and can be seen as a form of regularization. The scaling factor λ intuitively assigns different weight to those goals: For $\lambda \rightarrow 1$ we have a Bayesian network where the log-likelihood and the KL-

Figure 1. Repeated samples from BNN with spatial dropout (top) and conventional dropout (bottom) for a specific (hard) test case. The individual predictions of the BNN with spatial dropout are smoother and show larger variations compared to the conventional dropout case, where the difference between predictions is on a smaller scale.

term receive equal weight. For $\lambda \to \infty$ the second term disappears, and the optimization objective turns into the negative log-likelihood. In order to estimate equation 1, and in particular derivatives thereof, sub-sampling and Monte-Carlo techniques are typically leveraged together with a reparametrization trick (Kingma & Welling, 2014), that allows to backpropagate gradients through distributions. One particular stochastic estimator we use in this work is the *flipout* estimator (Wen et al., 2018), which computes decorrelated stochastic gradient estimates of equation 1 with few perturbation samples. Additionally, we leverage the work of Gal & Ghahramani (2016), who showed that under mild conditions, conventional neural networks that were trained with dropout regularization, can be seen as a form of Bayesian neural networks. A Dropout layer randomly sets input units to 0 with the specified rate. Spatial Dropout, which we will also use in this work, sets entire feature-maps to 0. For both variants, obtaining stochastic predictions according to the posterior distribution is as simple as extending dropout to the prediction phase. In all considered BNNs, the marginal prediction can be obtained by computing the mean of repeated forward passes of a given input. Likewise, the standard deviation over the repeated samples can be seen as a measure of uncertainty.

4. Experiments

We investigate three scenarios of increasing complexity: (1) A static setup, where the learning goal is to infer steady-state Reynolds-averaged Navier-Stokes (RANS) solutions around airfoils. (2) A perturbed buoyancy-driven Navier-Stokes (NS) flow for which the time evolution is taken into account, and (3) turbulent Hasegawa-Wakatani simulations with a tightly integrated BNN.

In the following, we will explain the experimental setup and discuss the corresponding results for each of the three cases.

4.1. Reynolds-averaged Navier-Stokes flow

Previous work has successfully used conventional neural networks to infer RANS solutions around airfoils (Thuerey et al., 2018). We follow this experimental setup, but instead deploy a dropout Bayesian neural network as surrogate model. Thus, we investigate if BNNs are capable of obtaining similar results and can provide sensible uncertainty information.

Setup. We use the open-source code *OpenFOAM*, which solves a one equation turbulence model (Spalart-Allmaras), to generate ground truth data for training. We consider a range of Reynolds numbers $Re = [0.5, 5] \times 10^6$, incompressible flow and angles of attack in a range of $[-22.5°, +22.5°]$. With this, we simulate velocity and pressure distributions of flows around 1505 different airfoil shapes from the UIUC database (Selig, 1996). Following Thuerey et al. (2018), where a more detailed explanation of the data-generating process is available, we encode freestream conditions and airfoil shape in a $128 \times 128 \times 3$ grid, denoting 3 fields, each at 128×128 resolution: The first field is a mask of the airfoil shape, the other two x- and y- velocity components, respectively. The output data sets have the same size, but now the first channel describes the pressure p, whereas the other two channels still contain x- and y- velocity components of the desired RANS solution. We normalize with respect to the freestream velocity by: (1) dividing the target velocity $\mathbf{v}_o$ by the magnitude of the input velocity $\mathbf{v}_i$, $\hat{\mathbf{v}}_o = \mathbf{v}_o/|\mathbf{v}_i|$, (2) the target pressure p_o by the square of the input velocity, $\tilde{p}_o = p_o/|\mathbf{v}_i|^2$ and (3) removing the mean pressure from each solution. This learning problem corresponds to a regular supervised setup and can be formalized as minimizing

$$\mathbb{E}_{\boldsymbol{w} \sim q_{\boldsymbol{\theta}}} \left[\sum_{i=1}^{N} \|f(\mathbf{x}_i|\boldsymbol{w}) - \mathbf{y}_i\|_1 \right] - \frac{1}{\lambda} D_{\mathrm{KL}} \qquad (2)$$

Table 1. RANS-Flow Performance

	Dropout				Spatial dropout			
	0.01	0.05	0.1	0.25	0.01	0.05	0.1	0.25
Non-Bayesian MAE $\times 100$	0.70	0.70	0.75	0.98	0.60	0.70	**0.73**	**0.96**
BNN MAE-avg $\times 100$	**0.69**	**0.68**	**0.64**	**0.72**	**0.59**	0.70	0.77	0.97
BNN MAE-std $\times 100$	0.26	0.45	0.58	0.78	0.33	0.60	0.78	1.10

with respect to θ. f is a Bayesian neural network whose weights **w** are sampled from the approximate posterior distribution. $\mathbf{x}_i$ and $\mathbf{y}_i$ are the ith input and target, respectively, each consisting of three 128×128 grids. For readability, we omitted the arguments of D_{KL}, which are the prior and approximate posterior distributions like in equation 1. We train a U-Net with Monte-Carlo dropout (both spatial dropout and conventional dropout) for 100 epochs with a learning rate of 0.006, learning rate decay and the *Adam* optimizer (Kingma & Ba, 2017). Details of the neural network architecture can be found in the appendix.

Results. We find that the considered BNNs show similar performance to the non-Bayesian networks of Thuerey et al. (2018) in terms of mean absolute error. Table 1 illustrates that in our experiments, most BNNs are even slightly superior to their non-Bayesian counterparts. Since the non-Bayesian networks are likewise trained with dropout, we hypothesize that this is not caused by stronger regularization in BNNs. Instead, we think that the BNN posterior can successfully capture many compelling but different solutions, which are combined in an ensemble-like manner during marginalization (Wilson & Izmailov, 2020).

The values for the BNN in table 1 are computed as follows: For every input sample, 20 forward passes are performed, and the average per-cell prediction is computed. This is illustrated in figure 1, where the target is shown in the first column, followed by 5 of the 20 predictions. To the right, the averaged prediction is shown, and the last column corresponds to the uncertainty field, which is computed as the standard deviation per cell over the 20 predictions. The first row in figure 1 illustrates the forward pass for a BNN with spatial dropout, and the second row for a BNN with conventional dropout. Figure 1 also serves as an example of what has been observed generally when contrasting spatial dropout with conventional dropout: While the spatial dropout implementation leads to smooth large-scale variations across the repeated predictions, conventional dropout provides blurry, per-cell noise. The resulting average predictions, however, are both smooth and similar to each other. The standard deviation, in contrast, again differs in scale and smoothness of the variation, even though the general location is similar and sensible for both implementations. This is shown in figure 2, where 5 unseen airfoil shapes (first column) are shown together with the corresponding average

prediction (second column), uncertainty distribution (third column) and target (fourth column), both for spatial and conventional dropout.

Across all shapes, the average prediction matches the target distribution closely, and the bulk of uncertainty is located at more turbulent regions close to the airfoil. The full test set with all unseen shapes is plotted in the appendix (figures 7 and 8)), together with the MAE-uncertainty relation per test sample (figure 11).
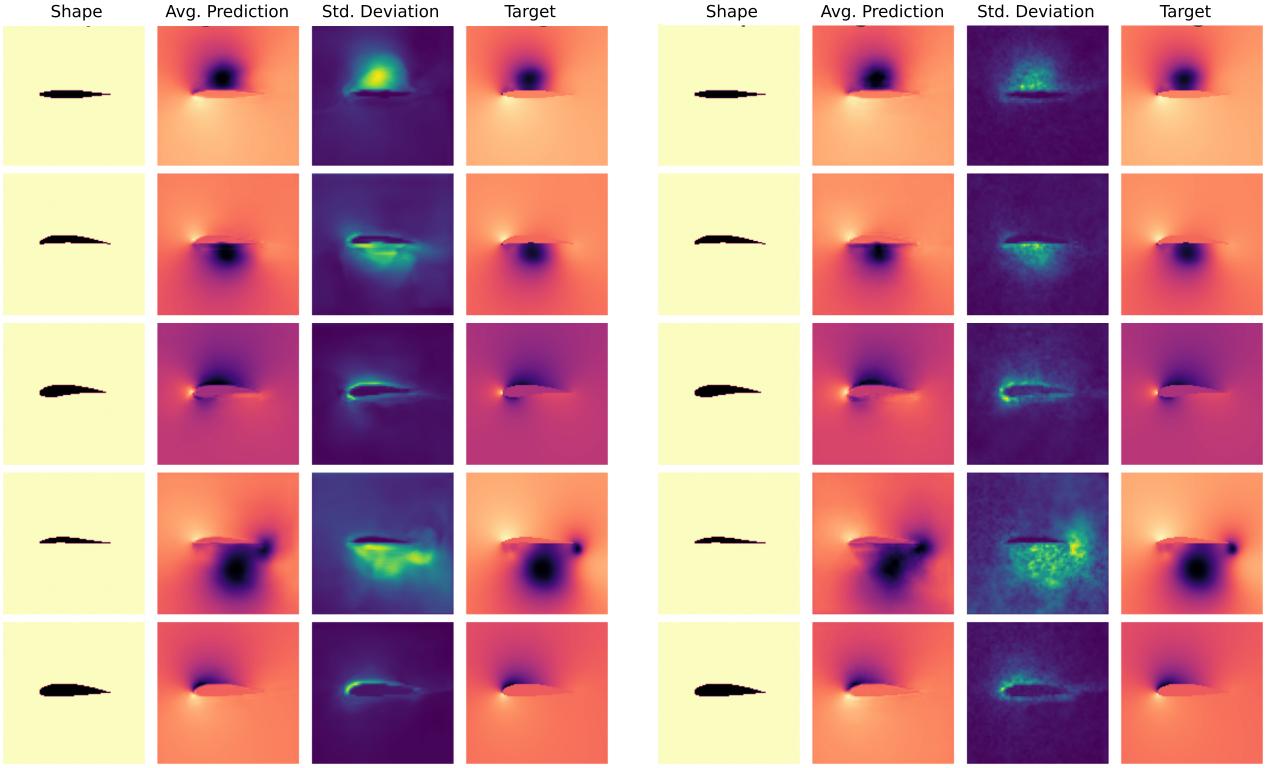
4.2. Perturbed buoyancy-driven Navier-Stokes flow

By perturbing Navier-Stokes simulations in a controlled manner, we create a supervised multimodal learning setup. We deploy a flipout-BNN and investigate if the multimodal nature inherent in the data-generating process can be captured by the stochastic predictions of the network. Our results show that more chaotic trajectories cause very significant difficulties for the trained BNNs. To investigate to what extent the BNN uncertainty allows us to distinguish unsuccessful cases from well performing ones, we relate the network's uncertainty to its predictive performance.

Setup. We leverage the simulation framework *PhiFlow* (Holl et al., 2020) to simulate time sequences of incompressible Navier-Stokes flows with a Boussinesq model for buoyancy forces. We simulate 2d trajectories on a 64×64 computational grid. For each trajectory, we add an inflow location with fixed $y-$ and random $x-$position and a buoyancy factor of 0.2. We simulate 64 time steps and add noise to the velocity profile in every simulation step. Hence, the solver receives an already perturbed input and performs the solver step on noisy data. For large perturbations, this can lead to very chaotic trajectories, as shown via an advected marker field in figure 12. Formally, we can obtain a state $\mathbf{s}_i^t$ of trajectory t at simulation time i by repeatedly applying a Navier-Stokes simulation operator $\mathcal{P}$ and a perturbation operator $\mathcal{Q}$:

$$\mathbf{s}_i^t = \mathcal{P}\mathcal{Q}\mathbf{s}_{i-1}^t = (\mathcal{P}\mathcal{Q})^i \mathbf{s}_0^t$$

where the state $\mathbf{s}_i^t = (\mathbf{d}_i^t, \mathbf{v}_i^t)$ with $\mathbf{d}_i^t$ denoting the marker field and $\mathbf{v}_i^t$ the velocity fields. The fluid model defining $\mathcal{P}$ is provided in the appendix A.4 together with a formal description and a visualization (figure 13) of the perturbation operator $\mathcal{Q}$. The learning goal in this setup is to predict the

Figure 2. BNN with spatial dropout (left) and conventional dropout (right) acting on unseen shapes. The first column shows the input airfoil shape, the second column the average BNN prediction for the pressure field, the third column the corresponding uncertainty distribution, and the last column the true target pressure field. The predictions and the uncertainty distribution are sensible: closer to the airfoil and for low pressure pockets, the network is more uncertain.

velocity profiles advanced by n simulation steps from the current marker profile, i.e. minimizing

$$\mathbb{E}_{\boldsymbol{w} \sim q_{\boldsymbol{\theta}}} \left[\sum_{i=1}^{N-n} \sum_{t=1}^{T} \left\| f(\mathbf{d}_i^t | \boldsymbol{w}) - \mathbf{v}_{i+n}^t \right\|_1 \right] - \frac{1}{\lambda} D_{\mathrm{KL}} \quad (3)$$

with respect to θ. In the following experiments, we use $T = 100$ trajectories with $N = 64$ frames each and an offset of $n = 10$. We monitor uncertainty estimation and predictive performance over the range of noise loads $[0., 0.9]$. We deploy a U-Net with flipout layers in the decoder part and train it with the *RMSprop* optimizer and a learning rate of 0.0014.

Results. In figure 3, a fine-grained analysis of the relation between mean absolute error and standard deviation is shown for the perturbed buoyancy driven Navier-Stokes data. Each dot represents the performance of an individual BNN, trained with a certain λ value (indicated by the color of the dot) on data with a certain noise level (indicated by the size of the dot). The noise level ranges from unperturbed simulations (noise load 0.) to strongly perturbed simulations (noise load 0.9). The latter corresponds to very chaotic trajectories (an example is shown in figure 12 in

the appendix). In such cases, the network is typically not capable of reasonably approximating the target distribution. In figure 3 the transition from small λ values (i.e. networks for which the KL term has larger weight) towards conventional, non-Bayesian networks with increasing λ values is clearly visible: Across all noise levels, the non-Bayesian network (red) is performing best in terms of mean absolute error. For a given noise level, larger values of λ furthermore always imply MAE values closer to the conventional network's performance. Like in the RANS flow case, BNNs are hence capable of obtaining predictive performance similar to their non-Bayesian counterparts (for large KL-prefactors), even though in this example they cannot outperform them. Also, the MAE-uncertainty relation is sensible: For each λ, the uncertainty is an increasing function of the MAE. The exact relation, however, depends on λ itself. It is close-to-linear for small KL-prefactors (for e.g. $\lambda = 100$, a linear regression yields a slope of 0.47) and becomes sub-linear for larger prefactors. A mapping from uncertainty to MAE is hence in principle possible.

Multimodality. The extent to which the stochastic BNN predictions can capture the multimodal nature of the data can
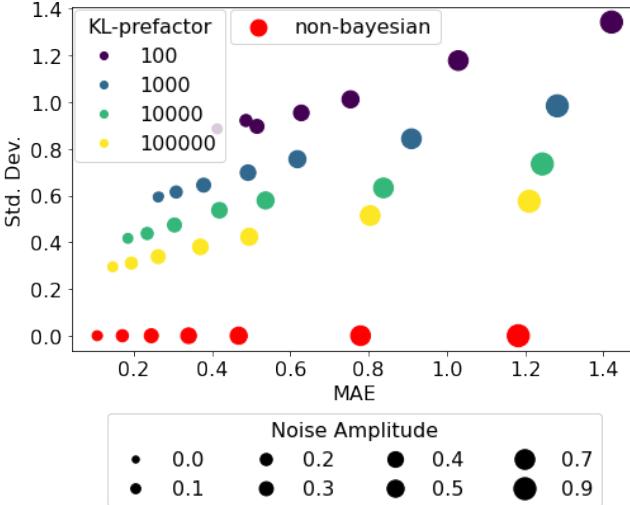
Figure 3. Uncertainty vs. mean absolute error for perturbed buoyancy-driven Navier-Stokes data. The performance of the conventional, non-Bayesian network is shown in red.

be seen qualitatively in figure 4. Each row shows repeated predictions of a network (trained with the noise level and λ value indicated in the first column) for a certain input. For a given noise level, larger values of λ lead to more precise predictions and fewer variations across repeated samples. In row 3 and 4 of figure 4, for instance, the model with larger λ value approximates the target distribution better, but shows very little variation. The model with smaller λ value shows more variation, but cannot approximate the detailed structures of the target distribution. Large noise levels, with examples shown in row 5 and 6, lead to very chaotic trajectories and cannot be approximated well at all. It is reassuring to see that the uncertainty for these very difficult learning tasks is significantly larger than for low-noise cases that were predicted reliably.

Across all noise levels, smaller λ values lead to more variations. However, they do not capture large, physically sensible multimodal solutions (e.g. the plume being twisted to the left instead of the right). Instead, for a given noise load the target is just approximated worse, with mostly small-scale variations in the predictions. Hence, obtaining sensible multimodal solutions with the naive approach of a BNN as surrogate model to a fluid solver was not successful with the proposed setup. However, we will revisit this goal in the next experiment and show that multimodal solutions can be achieved when BNNs are deployed via a solver-in-the-loop training.

4.3. Plasma Turbulence Simulations

In our third experiment, we consider a variation of the previous Navier-Stokes simulations: the simulation of drift wave turbulence in plasma transport. We use the two-dimensional Hasegawa-Wakatani system, which is a simplified, yet powerful coupled set of equations relating the number density field n with the electrostatic potential ϕ and its vorticity Ω. We leverage a differentiable implementation of the Hasegawa-Wakatani model (Greif et al., 2022) and deploy a dropout BNN to work as a corrector function, as suggested for non-Bayesian networks and other fluid contexts by Um et al. (2021). Our experiment shows that the BNN can not only significantly stabilize the simulations (like its non-Bayesian counterpart) but also successfully generate multimodal trajectories.

Setup. The *solver-in-the-loop* setup from Um et al. (2021), was demonstrated to have advantages in the setting of learned corrector functions for PDE solvers with regular, deterministic neural networks. Building upon this work, the HW-solver in our experiment is likewise realized such that it allows for gradient flow that supports backpropagation. During training, we simulate several predictor-corrector steps, compute losses with respect to a fine-grained pre-computed solution, and then backpropagate through the solver and corrector steps in order to update the corrector network parameters. Intuitively, this allows the network to explore the true, underlying physics and thus learn the highly non-linear nature of the errors. Furthermore, the corrector network ideally learns to correct its own behavior to reach a steady state in the learning process. Formally, the learning problem can be written as minimizing

$$\mathbb{E}_{\boldsymbol{w} \sim q_{\boldsymbol{\theta}}} \left[\sum_{i=0}^{n-1} \|\mathcal{C}(\mathcal{P}_S(\tilde{\mathbf{s}}_{t+i})|\boldsymbol{w}) - \mathcal{T}\mathbf{r}_{t+i+1}\|_2^2 \right] - \frac{1}{\lambda} D_{\mathrm{KL}}$$

with respect to θ. $\mathcal{C}$ is the corrector network with weights $\boldsymbol{w}$, $\mathcal{P}_S$ the solver, $\tilde{\mathbf{s}}_{t+i}$ a simulation state at time $t + i$ (that has potentially already been corrected in previous steps), and $\mathcal{T}\mathbf{r}_{t+i+1}$ is the ground truth state the simulation is compared to. Details for this setup are given in the appendix. Deploying a BNN as corrector network allows us to obtain a varied solution space for a given set of initial conditions, since a correction is then non-deterministic: Starting from the same initial frame, unrolling different trajectories is made possible by repeatedly applying the solver and the stochastic BNN corrector. In this setup, we deploy a ResNet with Bayesian dropout and enforce periodic boundary conditions by suitable padding. Details of this setup can likewise be found in the appendix.

Results. The simulations in this experiment are best inspected in the video available at https://youtu.be/725ulH9JA-8. We furthermore illustrate them in figure 5, where each row shows the frames of the temporal evolution of the ϕ-field during a simulation. The reference in the first row was simulated in a grid of 128×128 cells, downsampled to a
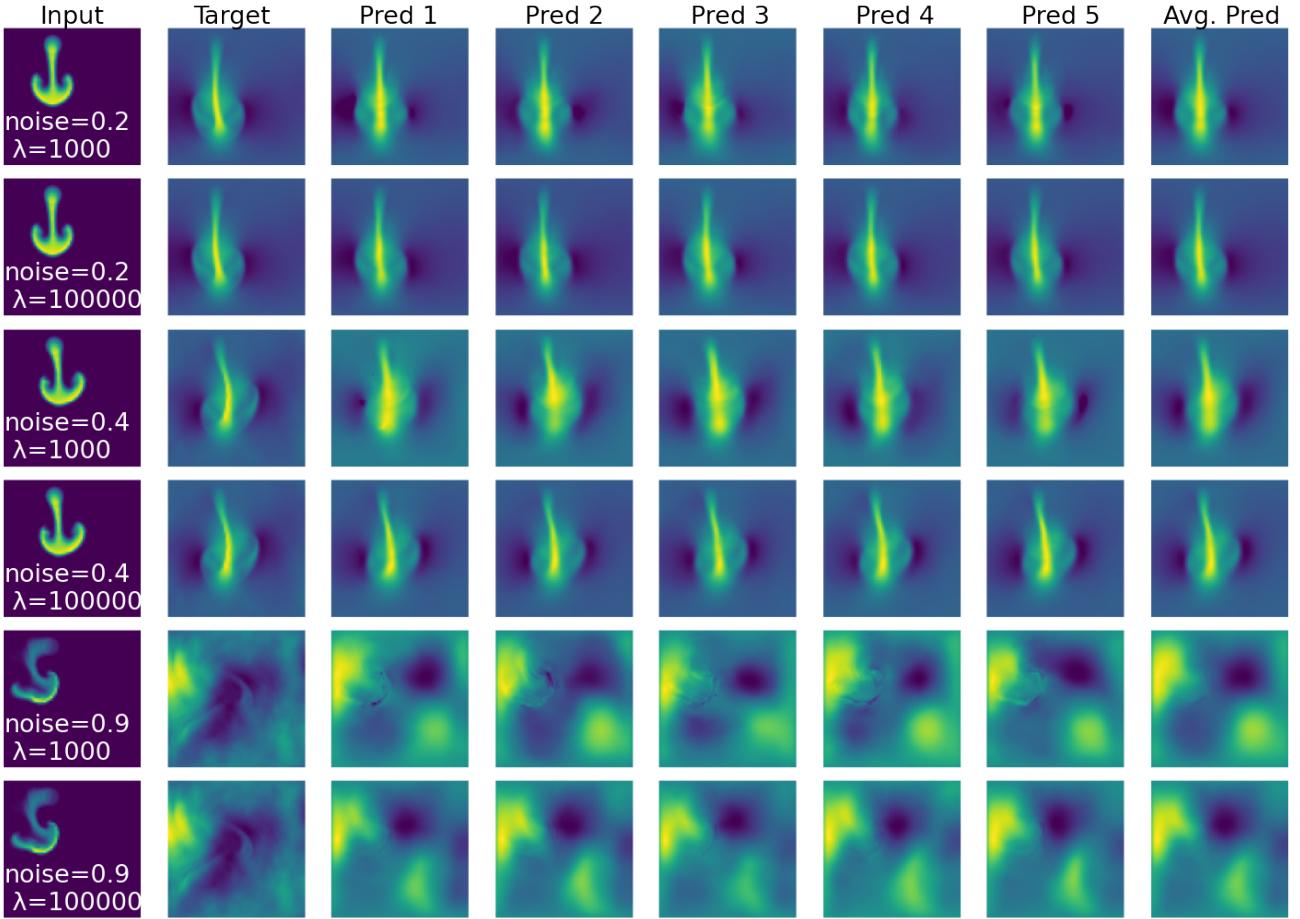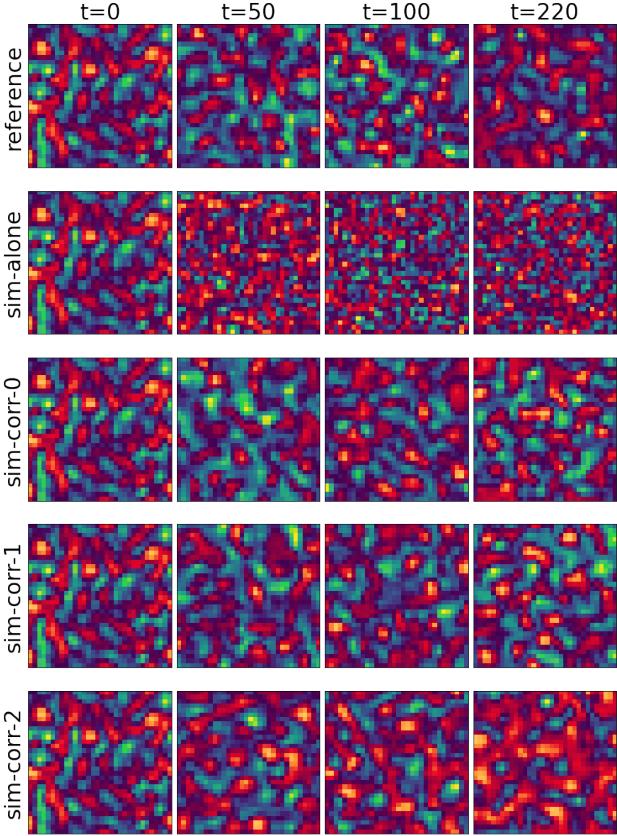
Input	Target	Pred 1	Pred 2	Pred 3	Pred 4	Pred 5	Avg. Pred

Figure 4. Repeated samples for different noise-per-frame training setups with $\lambda \in \{10^3, 10^5\}$ and noise amplitude $\in \{0.2, 0.4, 0.9\}$.

resolution of 32×32. This provides the ground truth solution. The second row (*sim-alone*) shows the evolution of the trajectory of a simulation that was performed entirely in the source domain of size 32×32, without the BNN corrector. Hence, the first frame of all rows are identical, but the evolutions diverge over time. The *sim-alone* simulation fails to maintain the turbulent state with large-scale structures that are clearly visible in row 1, producing mostly random noise towards the end of the simulation. This is due to the low resolution discretization not resolving interactions with relevant, but unresolved, wavelengths which we aim to correct. When simulating with resolutions that are too coarse, energy accumulates at the grid scale, dominating the behavior and leading to the loss of all physical meaning.

Rows 3, 4 and 5 (*sim-corr-0*, *sim-corr-1* and *sim-corr-2*) show 3 trajectories simulated in low resolution that were unrolled with the BNN acting as corrector. In all three cases the scale of the structures is preserved. Hence, the BNN was capable of learning suitable corrections such that the turbulent state could be simulated in a stable manner. Im-

portantly, the corrections in the last 3 rows were performed with the same trained network. Further, all three simulations start with the same initial conditions (see first column). The different evolutions of the trajectories are caused by the stochastic nature of the corrections. It is interesting to see that the trajectories indeed differ from another significantly: From visible inspection, there is no clear correlation between the frames of the last 3 rows. Figure 6 shows the pairwise L2-distance between the 3 corrected trajectories as a function of simulation time and underlines this observation: Since all trajectories start from the same frame, the distance is 0 initially, but increases over time. After about 30 internal time units, the distance stabilizes at the average L2 distance. This is about a factor of 10 larger than the autocorrelation time and hence shows, that after a short initial phase, the BNN indeed creates trajectories that are different to an extent where no correlation is measurable. In the appendix, the same behavior as in figure 5 is illustrated for the n and the Ω fields.
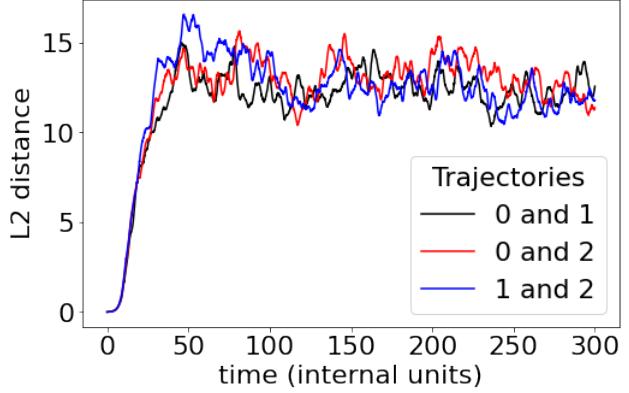
Figure 5. A BNN trained in the loop is able to stabilize turbulent Hasegawa-Wakatani simulations, as shown here with normalized ϕ fields. The stochastic nature of the corrections allows unrolling different trajectories from the same frame. The network was trained with spatial dropout and a rate of 0.01.

5. Conclusions and Discussion

We have presented a first study of BNNs deployed in a varied context of fluid simulations. We identified two use-cases of stochastic predictions, uncertainty assessment and multimodality, and investigated these in three complex setups with different variants of BNNs.

We were able to show that BNNs can reach slightly superior performance when used as pure surrogate for the RANS-flow case. We argue that the increased performance is not caused by stronger regularization in BNNs. The BNN posterior can successfully capture many compelling but different solutions, which are combined in an ensemble-like manner during marginalization (Wilson & Izmailov, 2020). We find qualitative differences in the uncertainty estimates when comparing spatial dropout to conventional dropout implementations, with spatial dropout providing smoother and larger variations. This is especially desirable, when realistic, individually sampled solutions are required, rather than the marginal prediction. We found the latter to be very similar

Figure 6. $L2$-distance between 3 BNN-corrected density trajectories: The initial correlation fades out after only about 30 internal time units (which is about 10 times the autocorrelation time).

for both spatial and conventional dropout.

In the more challenging setup of perturbed Navier-Stokes flow, we could show that BNNs trained on more chaotic data consistently provided a larger uncertainty. This underlines the sensibility of the obtained uncertainty estimates. However, the BNNs were not able to capture large-scale multi-modalities of the flow data set for this case.

Finally, we employed a Bayesian *solver-in-the-loop* framework to highly turbulent Hasegawa-Wakatani simulations. We could show that BNNs can successfully stabilize the simulations. Additionally, the stochasticity of the trained network can be leveraged to generate qualitatively different yet sensible trajectories from the same initial conditions. Especially for setups like the considered turbulent plasma physics simulations, this is very desirable. For such a case, the global statistical quantities, like average heat flux and particle transport, are typically much more important than specific microscopic states. This poses a very interesting avenue for future work, namely whether the relevant, large-scale statistics can successfully be recovered by the BNNs.

References

Amos, B. and Kolter, J. Z. OptNet: Differentiable Optimization as a Layer in Neural Networks. In *International Conference on Machine Learning*, pp. 136–145. PMLR, July 2017. URL http://proceedings.mlr.press/v70/amos17a.html. ISSN: 2640-3498.

Badrinarayanan, V., Kendall, A., and Cipolla, R. SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. *arXiv:1511.00561 [cs]*, October 2016. URL http://arxiv.org/abs/1511.00561. arXiv: 1511.00561.

Balescu, R. *Aspects of Anomalous Transport in Plasmas*. CRC Press, April 2005. ISBN 978-1-4200-3468-4. Google-Books-ID: Yaupom_qdKIC.

Bar-Sinai, Y., Hoyer, S., Hickey, J., and Brenner, M. P. Learning data-driven discretizations for partial differential equations. *Proceedings of the National Academy of Sciences*, 116(31):15344–15349, July 2019. ISSN 0027-8424, 1091-6490. doi: 10.1073/pnas.1814058116. URL http://www.pnas.org/lookup/doi/10.1073/pnas.1814058116.

Beck, A. D., Flad, D. G., and Munz, C.-D. Deep Neural Networks for Data-Driven Turbulence Models. *Journal of Computational Physics*, 398:108910, December 2019. ISSN 00219991. doi: 10.1016/j.jcp.2019.108910. URL http://arxiv.org/abs/1806.04482. arXiv: 1806.04482.

Bindal, A., Ierapetritou, M. G., Balakrishnan, S., Armaou, A., Makeev, A. G., and Kevrekidis, I. G. Equation-free, coarse-grained computational optimization using timesteppers. *Chemical Engineering Science*, 61(2):779–793, January 2006. ISSN 00092509. doi: 10.1016/j.ces.2005.06.034. URL https://linkinghub.elsevier.com/retrieve/pii/S0009250905005737.

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. Weight Uncertainty in Neural Networks. *arXiv:1505.05424 [cs, stat]*, May 2015. URL http://arxiv.org/abs/1505.05424. arXiv: 1505.05424.

Bronstein, M. M., Bruna, J., Cohen, T., and Veličković, P. Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges. *arXiv:2104.13478 [cs, stat]*, May 2021. URL http://arxiv.org/abs/2104.13478. arXiv: 2104.13478.

Brunton, S. L., Proctor, J. L., and Kutz, J. N. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937, April 2016. ISSN 0027-8424, 1091-6490. doi: 10.1073/pnas.1517384113. URL https://www.pnas.org/content/113/15/3932. Publisher: National Academy of Sciences Section: Physical Sciences.

Camargo, S. J., Biskamp, D., and Scott, B. D. Resistive drift-wave turbulence. *Physics of Plasmas*, 2(1):48–62, January 1995. ISSN 1070-664X. doi: 10.1063/1.871116. URL https://aip.scitation.org/doi/10.1063/1.871116. Publisher: American Institute of Physics.

Chen, R. T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. Neural Ordinary Differential Equations. *arXiv:1806.07366 [cs, stat]*, December 2019. URL http://arxiv.org/abs/1806.07366. arXiv: 1806.07366.

Chu, M. and Thuerey, N. Data-driven synthesis of smoke flows with CNN-based feature descriptors. *ACM Transactions on Graphics*, 36(4):1–14, July 2017. ISSN 0730-0301, 1557-7368. doi: 10.1145/3072959.3073643. URL https://dl.acm.org/doi/10.1145/3072959.3073643.

Cranmer, M., Greydanus, S., Hoyer, S., Battaglia, P., Spergel, D., and Ho, S. Lagrangian Neural Networks. *ICLR 2020*, 2020.

Cranmer, M., Tamayo, D., Rein, H., Battaglia, P., Hadden, S., Armitage, P. J., Ho, S., and Spergel, D. N. A bayesian neural network predicts the dissolution of compact planetary systems. *Proceedings of the National Academy of Sciences*, 118(40):e2026053118, Oct 2021. ISSN 1091-6490. doi: 10.1073/pnas.2026053118. URL http://dx.doi.org/10.1073/pnas.2026053118.

Crutchfield, J. and McNamara, B. S. Equations of Motion from a Data Series. *Complex Syst.*, 1987.

de Avila Belbute-Peres, F., Smith, K., Allen, K., Tenenbaum, J., and Kolter, J. Z. End-to-end differentiable physics for learning and control. *Advances in neural information processing systems*, 31:7178–7189, 2018.

Deodato, G., Ball, C., and Zhang, X. Bayesian Neural Networks for Cellular Image Classification and Uncertainty Analysis. *bioRxiv*, pp. 824862, October 2019. doi: 10.1101/824862. URL https://www.biorxiv.org/content/10.1101/824862v1. Publisher: Cold Spring Harbor Laboratory Section: New Results.

Freidberg, J. P. *Plasma Physics and Fusion Energy*. Cambridge University Press, July 2008. ISBN 978-1-139-46215-0. Google-Books-ID: Vyoe88GEVz4C.

Gal, Y. and Ghahramani, Z. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pp. 1050–1059. JMLR.org, 2016.

Graves, A. Practical variational inference for neural networks. In Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F., and Weinberger, K. Q. (eds.), *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011. URL https://proceedings.neurips.cc/paper/2011/file/7eb3c8be3d411e8ebfab08eba5f49632-Paper.pdf.

Greif, R., Thuerey, N., and Jenko, F. Deep learning for plasma physics simulations, 2022.

Greydanus, S., Dzamba, M., and Yosinski, J. Hamiltonian Neural Networks. *arXiv:1906.01563 [cs]*, September 2019. URL http://arxiv.org/abs/1906.01563. arXiv: 1906.01563.

Hasegawa, A. and Wakatani, M. Plasma Edge Turbulence. *Physical Review Letters*, 50(9):682–686, February 1983. doi: 10.1103/PhysRevLett.50.682. URL https://link.aps.org/doi/10.1103/PhysRevLett.50.682. Publisher: American Physical Society.

Hinton, G. E. and van Camp, D. Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory*, COLT '93, pp. 5–13, New York, NY, USA, August 1993. Association for Computing Machinery. ISBN 978-0-89791-611-0. doi: 10.1145/168304.168306. URL https://doi.org/10.1145/168304.168306.

Holl, P., Koltun, V., and Thuerey, N. Learning to Control PDEs with Differentiable Physics. *arXiv:2001.07457 [physics, stat]*, January 2020. URL http://arxiv.org/abs/2001.07457. arXiv: 2001.07457.

Hsieh, J.-T., Zhao, S., Eismann, S., Mirabella, L., and Ermon, S. Learning Neural PDE Solvers with Convergence Guarantees. *ICLR*, 2019.

Hu, Y., Anderson, L., Li, T.-M., Sun, Q., Carr, N., Ragan-Kelley, J., and Durand, F. DiffTaichi: Differentiable Programming for Physical Simulation. *ICLR*, 2020.

Innes, M., Edelman, A., Fischer, K., Rackauckas, C., Saba, E., Shah, V. B., and Tebbutt, W. A Differentiable Programming System to Bridge Machine Learning and Scientific Computing. *ArXiv*, 2019.

Kim, B., Azevedo, V. C., Thuerey, N., Kim, T., Gross, M., and Solenthaler, B. Deep Fluids: A Generative Network for Parameterized Fluid Simulations. *Computer Graphics Forum*, 38(2):59–70, May 2019. ISSN 0167-7055, 1467-8659. doi: 10.1111/cgf.13619. URL http://arxiv.org/abs/1806.02071. arXiv: 1806.02071.

Kingma, D. and Ba, J. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, January 2017. URL http://arxiv.org/abs/1412.6980. arXiv: 1412.6980.

Kingma, D. P. and Welling, M. Auto-Encoding Variational Bayes. *arXiv:1312.6114 [cs, stat]*, May 2014. URL http://arxiv.org/abs/1312.6114. arXiv: 1312.6114.

Kingma, D. P., Salimans, T., and Welling, M. Variational Dropout and the Local Reparameterization Trick. *arXiv:1506.02557 [cs, stat]*, December 2015. URL http://arxiv.org/abs/1506.02557. arXiv: 1506.02557.

Ko, J., Lucor, D., and Sagaut, P. Sensitivity of two-dimensional spatially developing mixing layers with respect to uncertain inflow conditions. *Physics of Fluids*, 20(7):077102, 2008.

Kwon, Y., Won, J.-H., Kim, B. J., and Paik, M. C. Uncertainty quantification using Bayesian neural networks in classification: Application to ischemic stroke lesion segmentation. pp. 13, 2018.

LaBonte, T., Martinez, C., and Roberts, S. A. We Know Where We Don't Know: 3D Bayesian CNNs for Credible Geometric Uncertainty. *arXiv:1910.10793 [cs, eess]*, April 2020. URL http://arxiv.org/abs/1910.10793. arXiv: 1910.10793.

Li, Y., He, H., Wu, J., Katabi, D., and Torralba, A. Learning Compositional Koopman Operators for Model-Based Control. *arXiv:1910.08264 [cs, math, stat]*, April 2020. URL http://arxiv.org/abs/1910.08264. arXiv: 1910.08264.

Long, Z., Lu, Y., Ma, X., and Dong, B. Pde-net: Learning pdes from data. January 2018. 6th International Conference on Learning Representations, ICLR 2018 ; Conference date: 30-04-2018 Through 03-05-2018.

MacKay, D. J. C. *Bayesian methods for adaptive models*. phd, California Institute of Technology, 1992a. URL https://resolver.caltech.edu/CaltechETD:etd-01042007-131447.

MacKay, D. J. C. A Practical Bayesian Framework for Backpropagation Networks. *Neural Computation*, 4(3):448–472, May 1992b. ISSN 0899-7667, 1530-888X. doi: 10.1162/neco.1992.4.3.448. URL https://direct.mit.edu/neco/article/4/3/448-472/5654.

McClure, P., Rho, N., Lee, J. A., Kaczmarzyk, J. R., Zheng, C. Y., Ghosh, S. S., Nielson, D. M., Thomas, A. G., Bandettini, P., and Pereira, F. Knowing What You Know in Brain Segmentation Using Bayesian Deep Neural Networks. *Frontiers in Neuroinformatics*, 0, 2019. ISSN 1662-5196. doi: 10.3389/fninf.2019.00067. URL https://www.frontiersin.org/articles/10.3389/fninf.2019.00067/full. Publisher: Frontiers.

Morton, J., Witherden, F. D., Jameson, A., and Kochenderfer, M. J. Deep Dynamical Modeling and Control of

Unsteady Fluid Flows. *arXiv:1805.07472 [cs]*, November 2018. URL http://arxiv.org/abs/1805.07472. arXiv: 1805.07472.

Neal, R. M. *Bayesian Learning for Neural Networks*, volume 118 of *Lecture Notes in Statistics*. Springer New York, New York, NY, 1996. ISBN 978-0-387-94724-2 978-1-4612-0745-0. doi: 10.1007/978-1-4612-0745-0. URL http://link.springer.com/10.1007/978-1-4612-0745-0.

Novati, G., de Laroussilhe, H. L., and Koumoutsakos, P. Automating turbulence modelling by multi-agent reinforcement learning. *Nature Machine Intelligence*, 3(1): 87–96, 2021.

Pfaff, T., Fortunato, M., Sanchez-Gonzalez, A., and Battaglia, P. Learning Mesh-Based Simulation with Graph Networks. September 2020. URL https://openreview.net/forum?id=roNqYL0_XP.

Pope, S. B. *Turbulent Flows*. Cambridge university press, 2000.

Raissi, M., Yazdani, A., and Karniadakis, G. Hidden Fluid Mechanics: A Navier-Stokes Informed Deep Learning Framework for Assimilating Flow Visualization Data. *ArXiv*, 2018.

Roache, P. J. Quantification of Uncertainty in Computational Fluid Dynamics. *Annual Review of Fluid Mechanics*, 29(1):123–160, 1997. doi: 10.1146/annurev.fluid.29.1.123. URL https://doi.org/10.1146/annurev.fluid.29.1.123. _eprint: https://doi.org/10.1146/annurev.fluid.29.1.123.

Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J., and Battaglia, P. W. Learning to Simulate Complex Physics with Graph Networks. *arXiv:2002.09405 [physics, stat]*, September 2020. URL http://arxiv.org/abs/2002.09405. arXiv: 2002.09405.

Selig, M. *UIUC airfoil data site*. Department of Aeronautical and Astronautical Engineering University of Illinois at Urbana-Champaign, 1996.

Sirignano, J. and Spiliopoulos, K. DGM: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1339–1364, December 2018. ISSN 00219991. doi: 10.1016/j.jcp.2018.08.029. URL http://arxiv.org/abs/1708.07469. arXiv: 1708.07469.

Sun, L. and Wang, J.-X. Physics-constrained bayesian neural network for fluid flow reconstruction with sparse and noisy data, 2020.

Thuerey, N., Weissenow, K., Mehrotra, H., Mainali, N., Prantl, L., and Hu, X. Well, how accurate is it? A study of deep learning methods for reynolds-averaged navier-stokes simulations. *CoRR*, abs/1810.08217, 2018. URL http://arxiv.org/abs/1810.08217.

Thuerey, N., Weißenow, K., Prantl, L., and Hu, X. Deep Learning Methods for Reynolds-Averaged Navier–Stokes Simulations of Airfoil Flows. *AIAA Journal*, 58(1):25–36, 2020. ISSN 0001-1452. doi: 10.2514/1.J058291. URL https://doi.org/10.2514/1.J058291. Publisher: American Institute of Aeronautics and Astronautics _eprint: https://doi.org/10.2514/1.J058291.

Tompson, J., Schlachter, K., Sprechmann, P., and Perlin, K. Accelerating Eulerian Fluid Simulation With Convolutional Networks. *arXiv:1607.03597 [cs]*, June 2017. URL http://arxiv.org/abs/1607.03597. arXiv: 1607.03597.

Tracey, B. D., Duraisamy, K., and Alonso, J. A Machine Learning Strategy to Assist Turbulence Model Development. 2015. doi: 10.2514/6.2015-1287.

Um, K., Hu, X., and Thuerey, N. iquid splash modeling with neural networks. *ComputerGraphics Forum*, pp. 23, December 2018.

Um, K., Brand, R., Yun, Fei, Holl, P., and Thuerey, N. Solver-in-the-Loop: Learning from Differentiable Physics to Interact with Iterative PDE-Solvers. *arXiv:2007.00016 [physics]*, January 2021. URL http://arxiv.org/abs/2007.00016. arXiv: 2007.00016.

Ummenhofer, B., Prantl, L., Thürey, N., and Koltun, V. Lagrangian fluid simulation with continuous convolutions. In *ICLR*, 2020.

Wakatani, M. and Hasegawa, A. A collisional drift wave description of plasma edge turbulence. *Physics of Fluids*, 27(3):611, 1984. ISSN 00319171. doi: 10.1063/1.864660. URL https://aip.scitation.org/doi/10.1063/1.864660.

Welling, M. and Teh, Y. W. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, pp. 681–688, Madison, WI, USA, 2011. Omnipress. ISBN 9781450306195.

Wen, Y., Vicol, P., Ba, J., Tran, D., and Grosse, R. Flipout: Efficient Pseudo-Independent Weight Perturbations on Mini-Batches. *arXiv:1803.04386 [cs, stat]*, April 2018. URL http://arxiv.org/abs/1803.04386. arXiv: 1803.04386.

Wenzel, F., Roth, K., Veeling, B. S., Swiatkowski, J., Tran, L., Mandt, S., Snoek, J., Salimans, T., Jenatton, R., and Nowozin, S. How Good is the Bayes Posterior in Deep Neural Networks Really? *arXiv:2002.02405 [cs, stat]*, July 2020. URL http://arxiv.org/abs/2002.02405. arXiv: 2002.02405.

Wilson, A. G. and Izmailov, P. Bayesian deep learning and a probabilistic perspective of generalization. *CoRR*, abs/2002.08791, 2020. URL https://arxiv.org/abs/2002.08791.

Yin, Y., Le Guen, V., Dona, J., de Bézenac, E., Ayed, I., Thome, N., and Gallinari, P. Augmenting physical models with deep networks for complex dynamics forecasting. *Journal of Statistical Mechanics: Theory and Experiment*, 2021(12):124012, 2021.

A. Appendix

A.1. U-Net Architecture

In the first and second experiment, we use a variant of the U-Net architecture as network model. Initially developed as a tool for image segmentation task, the U-Net architecture has shown to be a powerful tool across a wide variety of tasks and domains. It has a close-to-symmetric encoder-decoder-like architecture and uses skip-layer connections. In the contracting path (encoder), two 3×3 filters with strided convolutions followed by ReLU activation downsample each spatial dimension by 50%. At the same time, at every downsampling step, the number of feature channels is doubled. In the expansive part (decoder), an upsampling of the feature map increases the spatial resolution and is followed by a 2×2 convolution which halves the number of feature channels. Then, the resulting tensor is concatenated with the correspondingly cropped feature map from the encoder through a skip-layer connection. Finally, two 3×3 filters with ReLU activation are applied. In our implementation, we additionally apply batch normalization after the convolutional layers. In the first RANS-flow experiment, we use MC dropout, i.e. we apply (spatial) dropout after every layer. For the non-Bayesian network, we apply dropout only during training, whereas we extend it to the prediction phase for the Bayesian implementation. In the perturbed Navier-Stokes experiment, we deploy the U-Net as 'half-Bayesian' flipout network: We apply conventional layers in the encoder part, but TensorFlows Flipout layers in the decoder part of the network.

A.2. ResNet architecture

In the final solver-in-the-loop experiment, we modify the ResNet that has been used in the original paper by Um et al. (2021), consisting of 12 convolutional layers with kernel size 5 and 32 feature channels each. Again, we apply dropout after every layer and extend it to the prediction phase in order to obtain MC samples.

A.3. RANS-Flow

For completeness, we provide all 90 test samples together with their marginal prediction and the corresponding uncertainty. Figure 7 illustrates the spatial dropout implementation with rate 0.01, and figure 8 the conventional dropout implementation with rate 0.1. Further, we show repeated samples for varying dropout rates, again for spatial dropout in figure 9 and conventional dropout in figure 10. Figure 11 shows the uncertainty-MAE relation of all test samples for a BNN with spatial dropout rate 0.1.
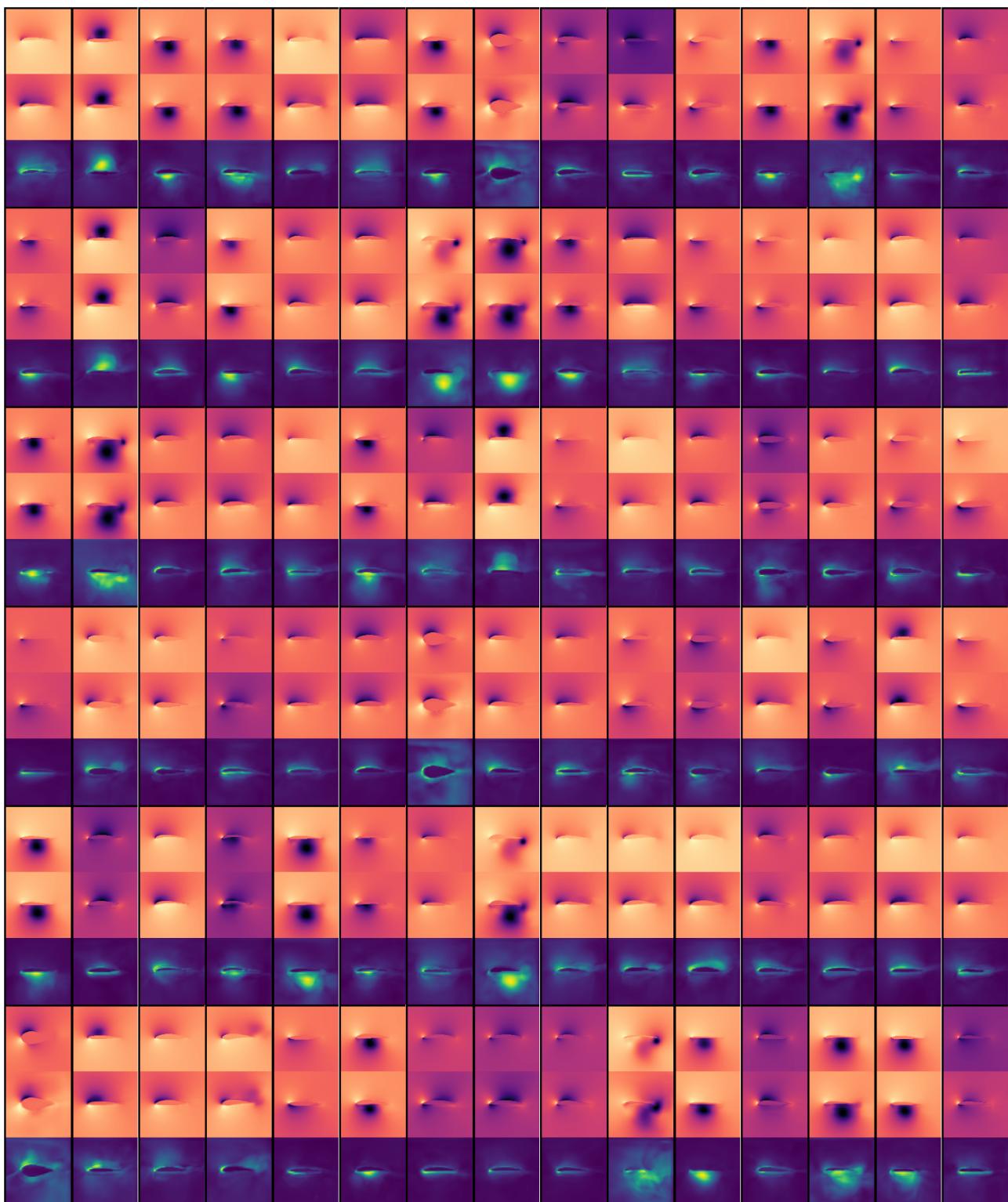
Figure 7. All test samples for a BNN with spatial dropout and dropout rate 0.01. For each sample, the target is shown on top, the average prediction in the middle, and the corresponding uncertainty (in different color map) on the bottom.
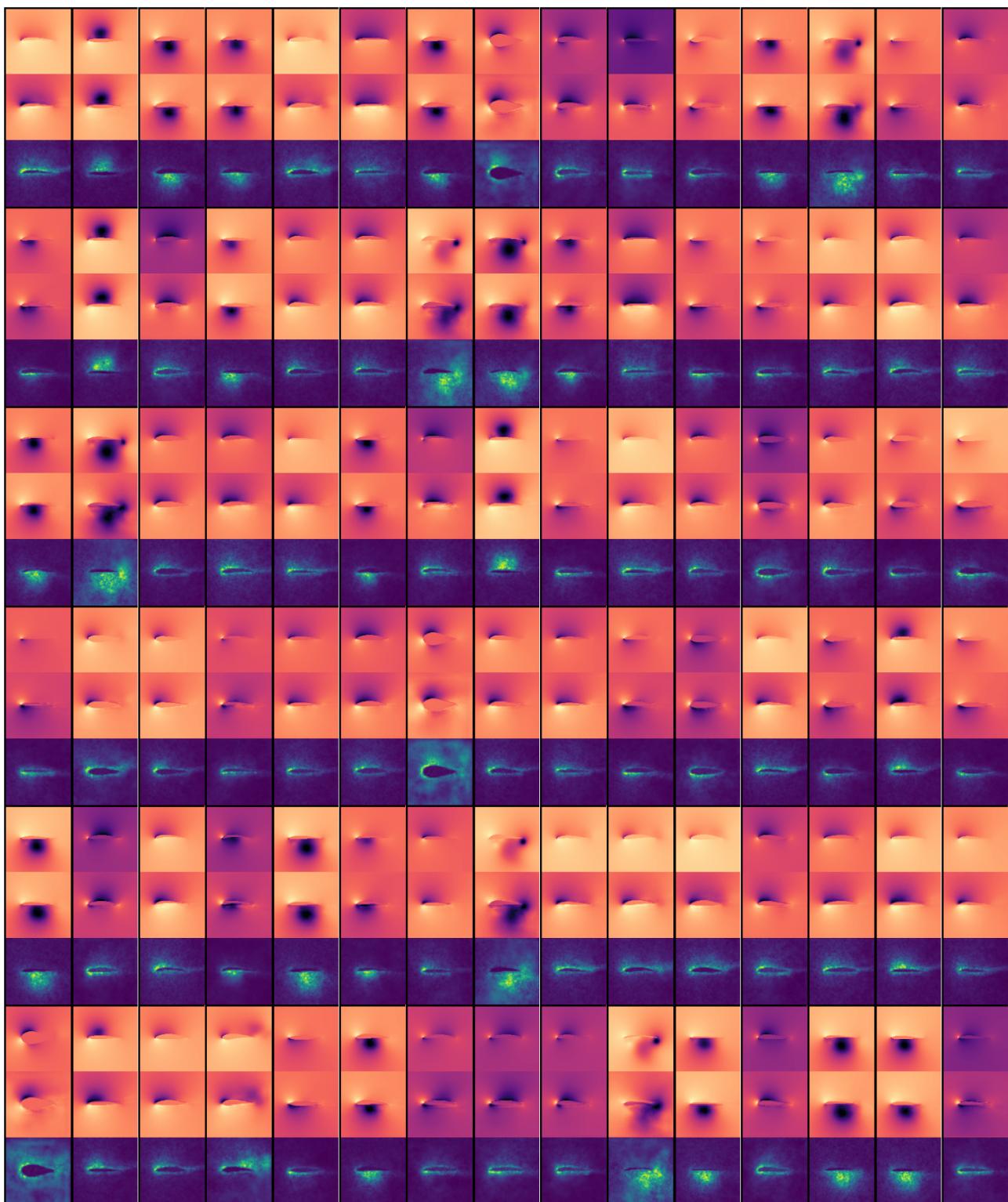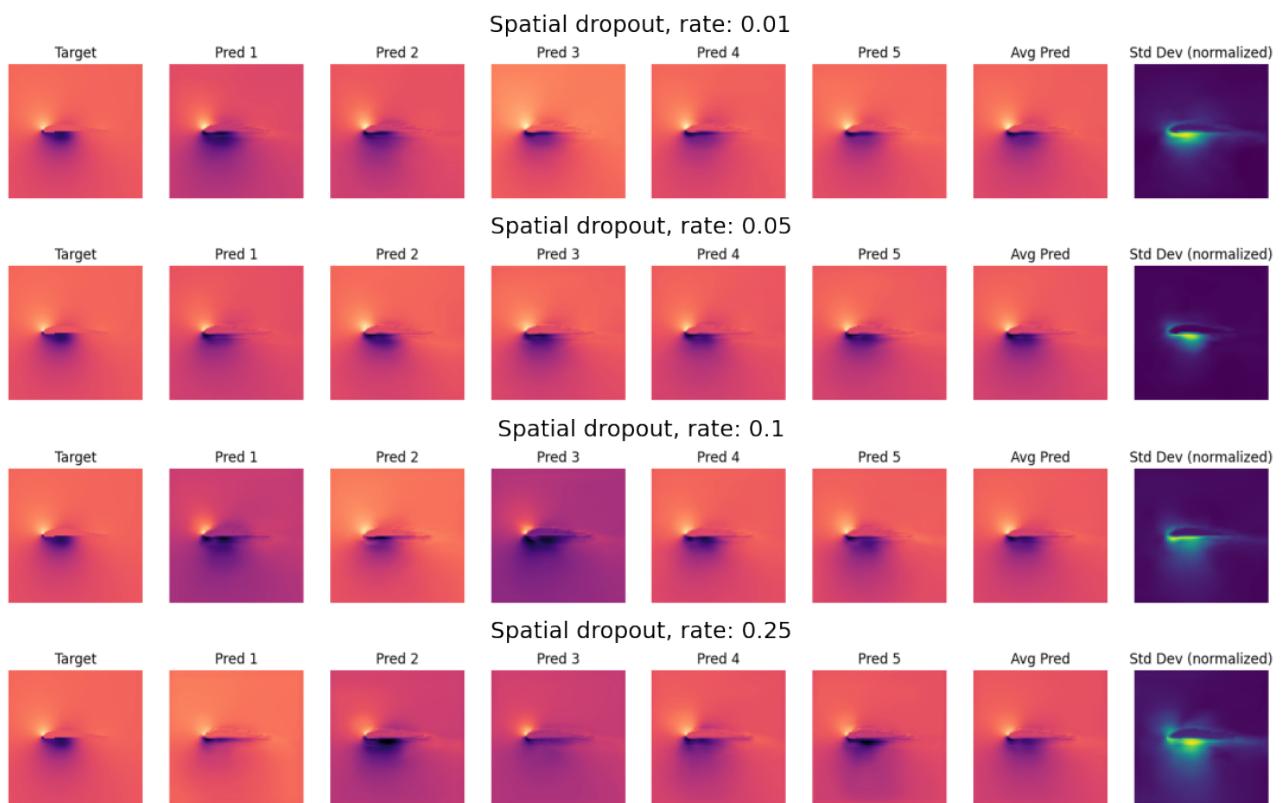
Figure 8. All test samples for a BNN with regular dropout and dropout rate 0.1. For each sample, the target is shown on top, the average prediction in the middle, and the corresponding uncertainty (in different color map) on the bottom.

Figure 9. BNN samples (spatial dropout, different rates) for RANS flow.

Figure 10. BNN samples (regular dropout, different rates) for RANS flow.

Figure 11. MAE vs. uncertainty per test sample for a BNN with spatial dropout rate 0.1: Larger uncertainty typically implies larger error.

A.4. Perturbed buoyancy-driven Navier-Stokes flow

The fluid model underlying the simulations are the Navier-Stokes equations in the boussinesq approximation:

$$\frac{\partial u_x}{\partial t} + \mathbf{u} \cdot \nabla u_x = -\frac{1}{\rho} \nabla p$$

$$\frac{\partial u_y}{\partial t} + \mathbf{u} \cdot \nabla u_y = -\frac{1}{\rho} \nabla p + \xi v$$

$$\text{subject to} \quad \nabla \cdot \mathbf{u} = 0,$$

$$\frac{\partial v}{\partial t} + \mathbf{u} \cdot \nabla v = 0$$

with velocity field $\mathbf{u}$ and pressure p. v is a scalar marker field indicating regions of higher temperature (or equivalently higher buoyancy force), and ξ is a measure of the strength of the buoyancy force. Figure 13 shows noise samples $\mathbf{n}$, like they are added to the velocity profiles by the perturbation operator $\mathcal{Q}$:

$$\mathcal{Q}\mathbf{s}_i^t = \mathcal{Q}(\mathbf{d}_i^t, \mathbf{v}_i^t) = (\mathbf{d}_i^t, \mathbf{v}_i^t + \mathbf{n})$$

Figure 12 shows samples of a perturbed trajectory, i.e. where $\mathcal{Q}$ was applied before every solver step. Figure 14 shows the result of a linear regression between standard deviation and mean absolute error for $\lambda = 100$.
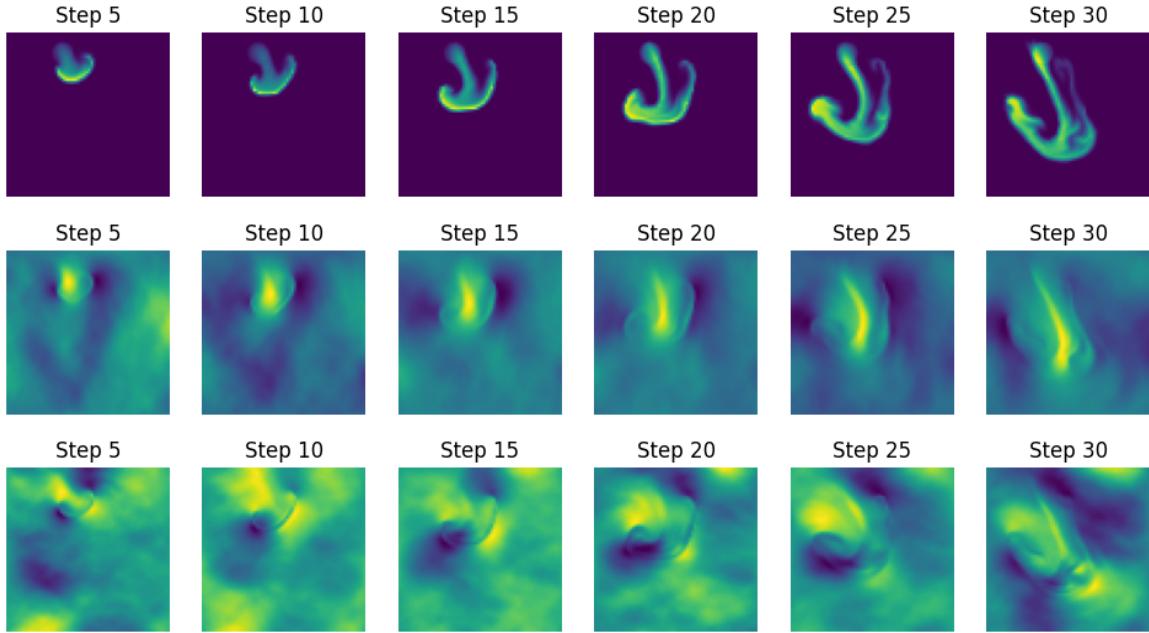

Figure 12. Trajectories of marker field, $x-$component and $y-$component of velocity profiles for dynamically perturbed Navier-Stokes flow.
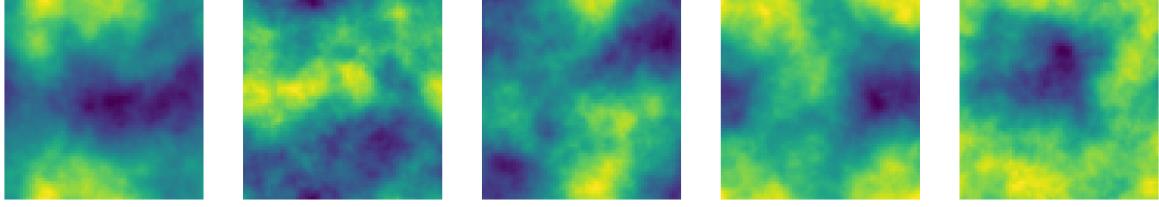
Figure 13. Samples of the noise fields which are added to the velocity fields by the perturbation operator $\mathcal{Q}$.
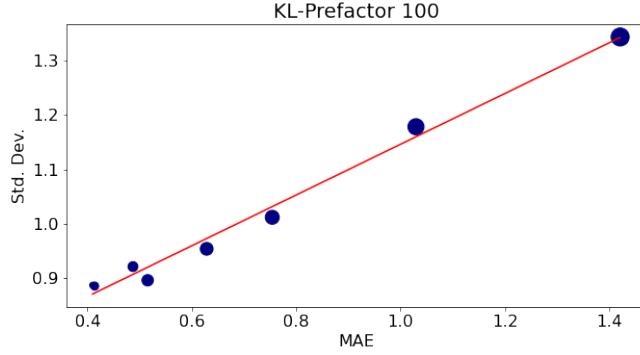

Figure 14. For $\lambda = 100$, a linear regression between standard deviation and mean absolute error yields a slope of 0.47 and an intercept of 0.68.

A.5. Turbulent plasma simulations

The Hasegawa-Wakatani system (Hasegawa & Wakatani, 1983; Wakatani & Hasegawa, 1984) is a simple fluid model relevant for the description of plasma turbulence. It assumes a constant magnetic field in z-direction, $\mathbf{B} = B\hat{\mathbf{z}}$ and can be written as
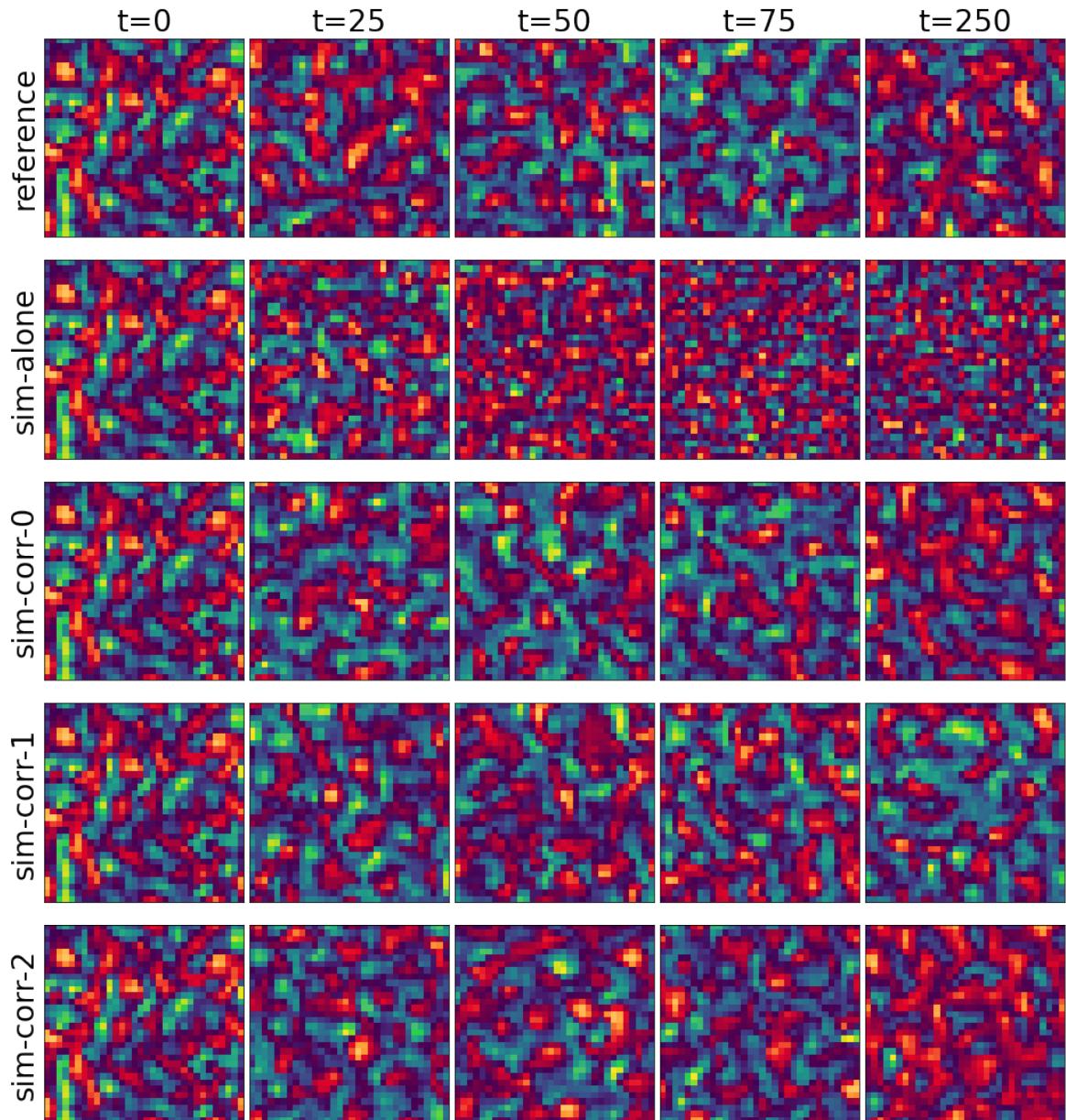
$$\frac{\partial}{\partial t}\nabla_\perp^2 \tilde{\phi} + (\hat{\mathbf{z}} \times \nabla_\perp) \cdot \nabla_\perp \nabla_\perp^2 \tilde{\phi} = \alpha(\tilde{\phi} - \tilde{n}) + D^\phi \tag{4}$$
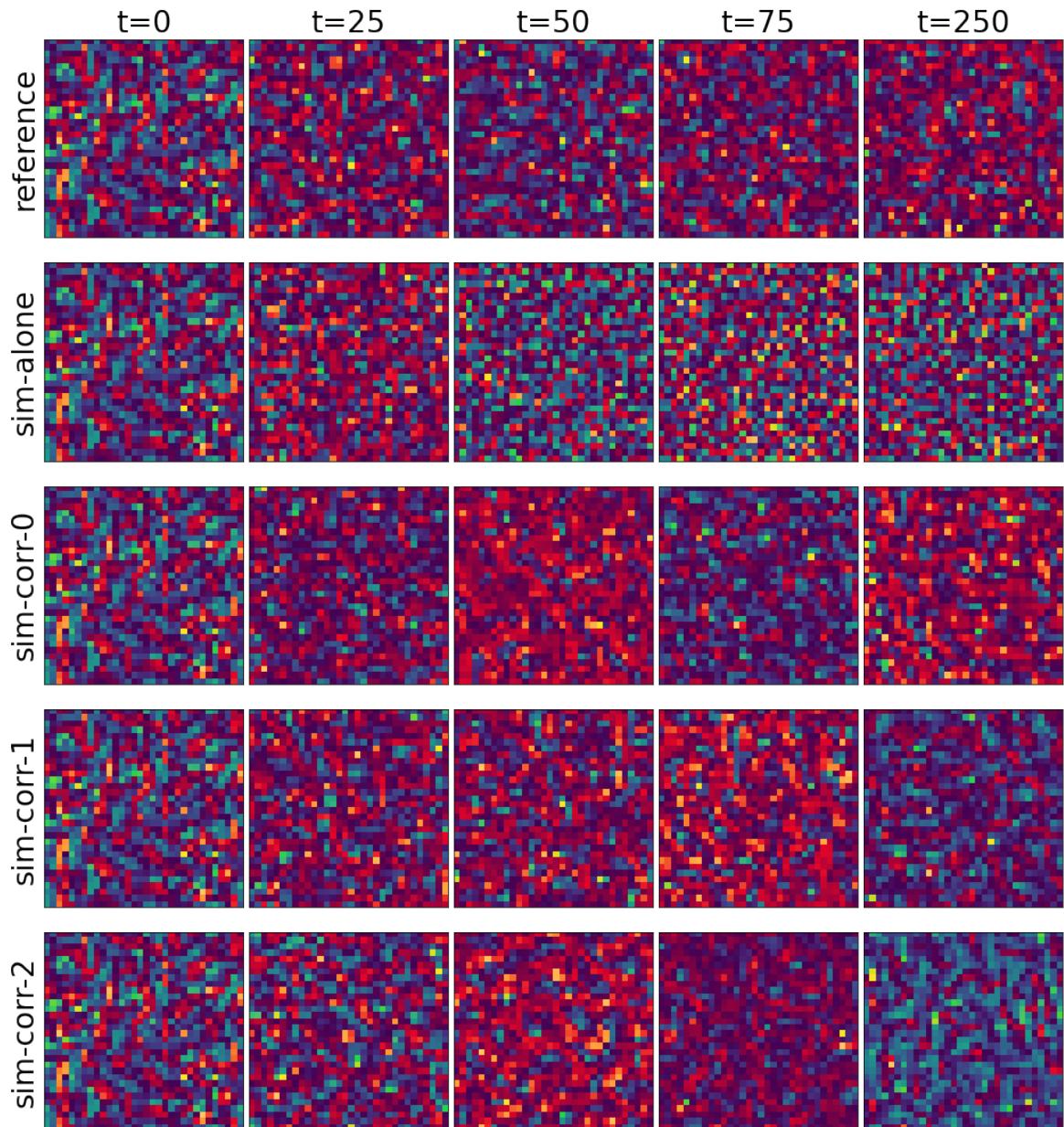
$$\frac{\partial}{\partial t}\tilde{n} + (\hat{\mathbf{z}} \times \nabla_\perp) \cdot \nabla_\perp \tilde{n} = \alpha(\tilde{\phi} - \tilde{n}) + D^n \tag{5}$$

where x, y and t are normalized to work as dimensionless spatial and temporal coordinates, $\tilde{\phi}(x, y)$ is the normalized electrostatic potential, and $\tilde{n}(x, y)$ the normalized density. Equation 4 and equation 5 are cross-coupled through the adiabaticity parameter $\alpha \propto \frac{T L_n}{n_0 e^2 c_s}$. The expressions D^ϕ and D^n are non-physical terms added for numerical stability. Their role is to dissipate energy accumulating on grid scale through a so-called hyper-diffusivity. For details on the implementation, refer to Greif et al. (2022). $\nabla_\perp$ is to be understood as $(\partial x, \partial y, 0)$. Even though it might not be obvious at first glance, $\nabla_\perp^2 \phi$ describes the fluid vorticity. This is because the fluid velocity can be approximated as the $\mathbf{E} \times \mathbf{B}$-drift and is hence $v_D \propto \mathbf{E} \times \mathbf{B}$. The vorticity can thus be written as

$$\mathbf{\Omega} = \nabla \times \mathbf{v}_D \propto \nabla \times (-\nabla \phi \times \hat{\mathbf{z}}) = \nabla^2 \phi \hat{\mathbf{z}}. \tag{6}$$

A more thorough introduction to the Hasegawa-Wakatani model, together with a detailed derivation of the equations, is available in chapter 2.5 of (Balescu, 2005), while (Camargo et al., 1995) provides good insights into numerical experiments with the model.

Figure 15. Density n

Figure 16. Vorticity Ω

A.6. Solver-in-the-loop Training Algorithm

The solver-in-the-loop setup can be formalized as follows. One considers two different discretization schemes of the same PDE $\mathcal{P}$: A fine reference discretization $\mathcal{P}_R$ with solutions $\mathbf{r} \in \mathcal{R}$ from the *reference manifold* and a more coarse source discretization $\mathcal{P}_S$ with solutions $\mathbf{s} \in \mathcal{S}$. $\mathbf{r}$ and $\mathbf{s}$ are states at certain instances in time, but with different *spatial* resolution. Evolutions of the PDE consist then of a more exact reference sequence $\{\mathbf{r}_t, \mathbf{r}_{t+\Delta t}, ..., \mathbf{r}_{t+k\Delta t}, \}$ and a more coarse source sequence $\{\mathbf{s}_t, \mathbf{s}_{t+\Delta t}, ..., \mathbf{s}_{t+k\Delta t}, \}$, respectively. Furthermore, a mapping operator $\mathcal{T}$ is defined such that it transforms a phase space point from the reference manifold to the source manifold.

In our case, where we consider the manifolds to differ only spatially, $\mathcal{T}$ can be seen as a downsampling operator, and we can write $\mathbf{s}_{t_0} = \mathcal{T}\mathbf{r}_{t_0}$ for the initial state. A trajectory in the phase space, i.e. the evolution of a starting point $\mathbf{r}_t$, is obtained by evaluating $\mathcal{P}_R$ iteratively. A state after k steps of equal step size Δt is then $\mathbf{r}_{t+k}$. Importantly, $\mathcal{P}_S(\mathcal{T}\mathbf{r}_t) \neq \mathcal{T}\mathbf{r}_{t+1}$. In other words, applying a solver step in the fine reference manifold and downsampling afterwards is not the same as downsampling first and applying the solver step in the coarse source manifold. The numerical error in the case where one applies the solver step in the coarse source manifold is typically larger, and this is precisely what we want to mitigate here. The learning goal is thus to train a correction operator $\mathcal{C}(\mathbf{s}|\theta)$, such that a corrected solution $\mathcal{C}(\mathbf{s})$ is closer to the downsampled solution than an unmodified state, e.g. in terms of the $L2$-norm:

$$\|\mathcal{C}(\mathcal{P}_S(\mathcal{T}\mathbf{r}_{t_0})) - \mathcal{T}\mathbf{r}_{t_1}\| < \|\mathcal{P}_S(\mathcal{T}\mathbf{r}_{t_0}) - \mathcal{T}\mathbf{r}_{t_1}\|$$

In our case, $\mathcal{C}(\mathbf{s}|\theta)$ is a neural network that receives a state $\mathbf{s}$ as input and corrects it to $\tilde{\mathbf{s}}$. Further, repeated applications of the solver are denoted exponentially:

$$\mathbf{s}_{t+n} = \mathcal{P}_S(\mathcal{P}_S(...\mathcal{P}_S(\mathcal{T}\mathbf{r}_t))) = \mathcal{P}_S^n(\mathcal{T}\mathbf{r}_t)$$

A fully corrected trajectory, where the correction is applied in every iteration, is then given by $\tilde{\mathbf{s}}_{t+n} = (\mathcal{C}\mathcal{P}_S)^n(\mathcal{T}\mathbf{r}_t)$. The solver-in-the-loop now leverages the differentiable physics pipeline that is available through *PhiFlow* and integrates the solver into the training loop when training $\mathcal{C}$. Thus, the corrector $\mathcal{C}$ receives states $\tilde{\mathbf{s}}$ that it has corrected previously and can backpropagate gradients through the solver steps. The objective function is thus

$$\underset{\theta}{\arg\min} \sum_{i=0}^{n-1} \|\mathcal{C}(\mathcal{P}_S(\tilde{\mathbf{s}}_{t+i})|\theta) - \mathcal{T}\mathbf{r}_{t+i+1}\|_2^2. \tag{7}$$

The subtlety of *equation 7* is that the states $\tilde{\mathbf{s}}_{t+k}$ themselves depend on the corrected function since they are computed via $\tilde{\mathbf{s}}_{t+k} = (\mathcal{C}\mathcal{P}_S)^k(\mathcal{T}\mathbf{r}_t)$, leading to a recurrent optimization problem. In practice, one samples short simulation intervals of length $L \approx 5$ instead of the full trajectory and uses batches of data. One starts with a collection of reference states, which are downsampled to the source domain. Then, each downsampled state evolves into a trajectory of size L (the look-ahead) via recurrent application of $\mathcal{C}\mathcal{P}_S$. Each trajectory is compared to the corresponding reference trajectory, and the loss is computed. Backpropagation then allows to propagate gradients of the loss with respect to the network's parameters through all solver steps, like it is illustrated in figure 17. In pseudo-code, the optimization procedure can be written as

---

Algorithm 1 Solver-in-the-loop training

Result: Trains a corrector $\mathcal{C}(\theta)$ in the solver-in-the-loop framework.

1. Inputs: A full, fine-grained reference trajectory. $\{\mathbf{r}_0, \mathbf{r}_1, ..., \mathbf{r}_N\}$

2. Set learning rate η, look-ahead L, batch size B and initialize $\boldsymbol{\theta}$ randomly.

while θ *not converged* **do**

  Set $\Delta\theta \longleftarrow 0$.

  Sample a set S of B random integers from the interval $[0, N - L]$.

  for *each j in S* **do**

    Obtain initial state in coarse-grained source domain by downsampling: $\mathbf{s}_j = \mathcal{T}\mathbf{r}_j$.

    Compute L-look-ahead trajectory by applying solver and corrector iteratively and summarize to loss

$$\mathcal{L} = \sum_{i=0}^{L-1} \|\mathcal{C}(\mathcal{P}_S(\tilde{\mathbf{s}}_{j+i})|\theta) - \mathcal{T}\mathbf{r}_{j+i+1}\|_2^2. \tag{8}$$

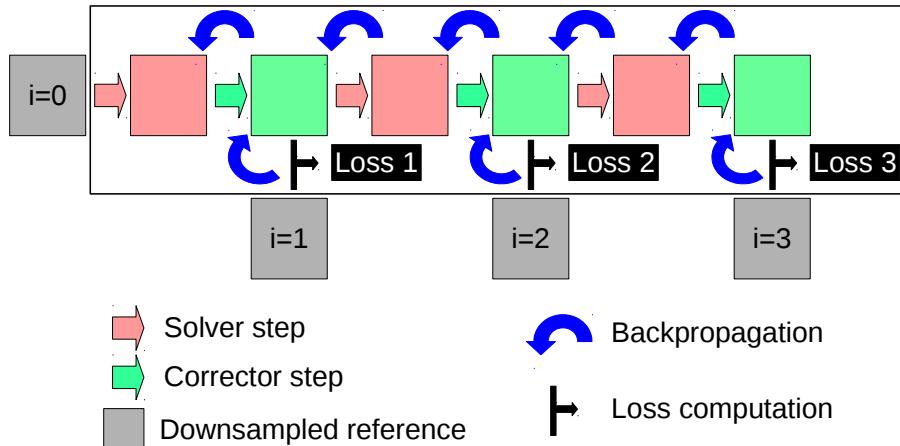   Compute derivative of loss w.r.t. θ by backpropagation through solver:
    $\Delta\boldsymbol{\theta} \longleftarrow \Delta\boldsymbol{\theta} + \frac{\partial}{\partial\boldsymbol{\theta}}\mathcal{L}$.

  end

  Update $\boldsymbol{\theta} \longleftarrow \boldsymbol{\theta} + \eta\Delta\boldsymbol{\theta}$.

end

Figure 17. Sketch of the solver-in-the-loop with a look-ahead of $L = 3$ and batch size $B = 1$: Starting from a downsampled reference at $i = 0$, solver and corrector are applied iteratively 3 times. Each corrected frame is compared to the corresponding downsampled reference frame and the respective losses are computed. Those losses are then backpropagated through the solver and the corrector in order to update the parameters of the corrector network.