Leveraging the Stochastic Predictions of Bayesian Neural Networks for Fluid Simulations

Maximilian Müller[*]
University of Tübingen and Tübingen AI Center
maximilian.mueller@wsii.uni-tuebingen.de

Robin Greif
TU Munich and IPP[†]

Frank Jenko
TU Munich and IPP

Nils Thuerey
TU Munich

Abstract

We investigate uncertainty estimation and multimodality via the non-deterministic predictions of Bayesian neural networks (BNNs) in fluid simulations. To this end, we deploy BNNs in two challenging experimental test-cases: 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. In our main experiment, we study BNNs in the context of differentiable solver interactions with turbulent plasma flows. We find that BNN-based corrector networks can stabilize coarse-grained simulations and successfully create diverse trajectories.

1 Introduction

Even though Bayesian neural networks (BNNs) have been studied for a long time in the Machine Learning community [Hinton and van Camp, 1993, MacKay, 1992b], they have only recently received 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. 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. Further, fluid models typically provide a deterministic description of an inherently chaotic and stochastic process [Pope, 2000], with solutions that often contain many bifurcation points [Ko et al., 2008]. 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, Freidberg, 2008]. There, transport is driven by micro-instabilities and the resulting systems are highly turbulent, creating a particularly relevant setup to study multimodality.

In the following, we assess the performance of BNNs on these tasks in two test-cases. We show that a trained BNN produces meaningful uncertainty estimates for complex fluid simulation scenarios,

[*]work done while at LMU Munich and Max Planck Institute for Plasma Physics.
[†]Max Planck Institute for Plasma Physics

Machine Learning and the Physical Sciences workshop, NeurIPS 2022.

like Reynolds-averaged Navier-Stokes flow around airfoils. In addition, our main experiments demonstrate that BNNs successfully generate varied predictions when working in conjunction with a numerical simulator in the plasma turbulence setup.

2 BNNs

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 we leverage the work of Gal and 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 appendix 5.7, we additionally showcase a BNN experiment with *flipout*, a stochastic estimator requiring more substantial changes in the learning process. In all considered BNNs, we either use the mean absolute error (MAE) or mean squared error (MSE) as loss, which in the Bayesian setting correspond to a Laplace and Gaussian likelihood, respectively. In all cases, 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.

3 Experiments

We investigate two scenarios: (1) A static setup, where the learning goal is to infer steady-state Reynolds-averaged Navier-Stokes (RANS) solutions around airfoils and (2) turbulent Hasegawa-Wakatani simulations with a tightly integrated BNN. In the following, we will explain the experimental setup and discuss the corresponding results for both cases. In appendix 5.7 we additionally show results for perturbed buoyancy-driven Navier-Stokes flow.

3.1 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 $\mathbf{v}_i$, $\tilde{\mathbf{v}}_o = \mathbf{v}_o / |\mathbf{v}_i|$, (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, for which 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 and Ba, 2017] to minimize the mean absolute error between the networks output and the simulated target fields. 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 in terms of mean absolute error Thuerey et al. [2018]. Table 1 shows the mean absolute error, averaged over 20 BNN predictions across the full test set. Most BNNs are slightly superior to their non-Bayesian counterparts. Since the non-Bayesian networks are likewise trained with dropout (but

dropout is not extended to the prediction phase), we hypothesize that this is not caused by stronger regularization in BNNs. Instead, we think that the BNN posterior can successfully capture compelling but different solutions, which are combined in an ensemble-like manner during marginalization [Wilson and Izmailov, 2020]. 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 for both spatial and conventional dropout (see figure 1 for 2 examples from the test set). 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, even though the general location is similar and sensible for both implementations. This can be seen in more detail in the appendix in fig. 7, where we show repeated predictions for individual samples. In addition, we find the heuristic *larger uncertainty corresponds to larger MAE* to hold. We illustrate this in figure 8 in the appendix. The full test set with all unseen shapes can also be found in the appendix for both spatial and normal dropout (figures 3 and 4).

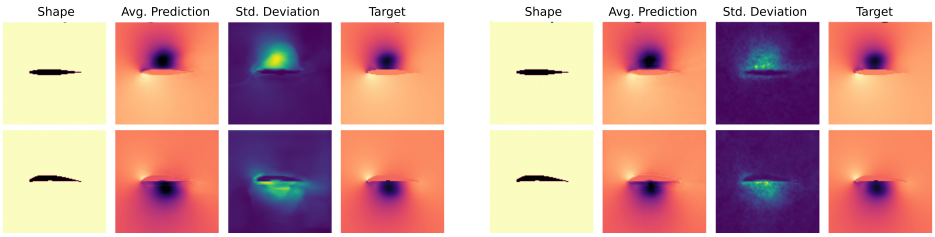

Figure 1: BNN with spatial dropout (left) and conventional dropout (right) acting on unseen shapes. The first column shows the input airfoil shape, the second column the average BNN prediction for the pressure field, the third column the corresponding uncertainty distribution, and the last column the 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.

3.2 Plasma Turbulence Simulations

In our second experiment, we consider 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

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

minimizing

$$\sum_{i=0}^{n-1} \|\mathcal{C}(\mathcal{P}_S(\tilde{\mathbf{s}}_{t+i})|\boldsymbol{w}) - \mathcal{T}\mathbf{r}_{t+i+1}\|_2^2$$

with respect to the weights $\boldsymbol{w}$ of a corrector network $\mathcal{C}$ (in contrast to prior work a BNN). $\mathcal{P}_S$ is 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 (section 5.5). 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. Figure 2 shows the initial ϕ field in row 0 and after 250 units of simulation time in row 1 for different simulation types. The reference in the first column was simulated in a grid of 128×128 cells, downsampled to a resolution of 32×32 and provides the ground truth solution. The second column (*sim-alone*) shows the result of a simulation that was performed entirely in the source domain of size 32×32, without the BNN corrector. It fails to maintain the turbulent state with large-scale structures that are clearly visible in the reference simulation, 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. When simulating with resolutions that are too coarse, energy accumulates at the grid scale, dominating the behavior and leading to the loss of physical meaning. Columns 3, 4 and 5 (*sim-corr-0*, *sim-corr-1* and *sim-corr-2*) were simulated in low resolution , but 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. Importantly, the corrections in the last 3 rows were all performed with the same trained network. All three simulations start with the same initial conditions (that is why at $t = 0$ the fields are identical) and the different evolutions of the trajectories are caused exclusively by the stochastic nature of the corrections of the BNN. 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 3 corrected states at $t = 250$. We quantify this behavior in figure 9 in the appendix, where we show the pairwise L2-distance between the 3 corrected trajectories as a function of simulation time. We further illustrate more time steps of all three fields in figures 10, 11 and 12. The time evolution of these simulations are best inspected in the supplementary video available at https://youtu.be/725ulH9JA-8.

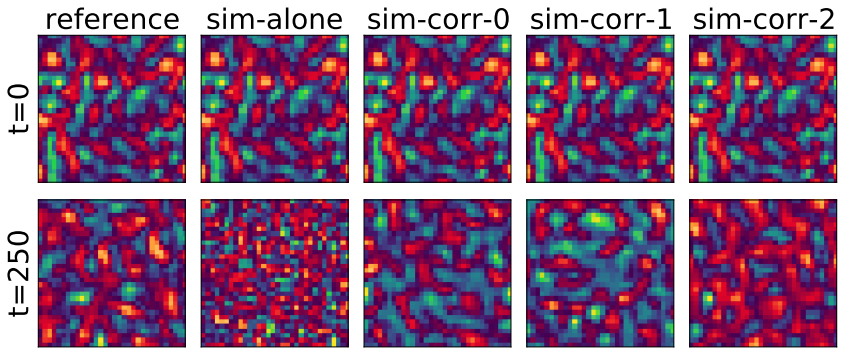

Figure 2: A BNN (spatial dropout, rate 0.01) trained in the loop is able to stabilize turbulent Hasegawa-Wakatani simulations, as shown here with normalized ϕ fields. The stochastic nature of the corrections allows unrolling different trajectories from the same frame.

4 Conclusions

In our first experiment were able to show that BNNs can reach slightly superior performance when used as pure surrogate for RANS-flow simulations around airfoils and provide sensible uncertainty information. We found qualitative differences in the uncertainty estimates when comparing spatial dropout to conventional dropout implementations, with spatial dropout providing smoother and

larger variations. In our main experiment, we deployed a Bayesian *solver-in-the-loop* framework to infer highly turbulent Hasegawa-Wakatani simulations. We could show that BNNs can successfully stabilize the simulations and the stochasticity of the trained network can be leveraged to generate a variety of qualitatively different yet sensible trajectories, starting from the same initial conditions.

Broader Impact

Obtaining sensible uncertainty estimates for physical simulations is a long-standing goal. Our first experiment shows that using simple dropout-BNNs as surrogate models can already provide accurate solutions with sensible uncertainty estimates. The smoother and larger variations of spatial dropout implementations are especially desirable when realistic, individually sampled solutions are required, rather than the marginal prediction. Obtaining stable, yet varied trajectories, like we did in our main experiment, is of interest in setups like the considered turbulent plasma physics simulations. There, the global statistical quantities, like average heat flux and particle transport, are typically much more important than specific microscopic states. This poses an interesting avenue for future work, namely whether the solution space obtained by the BNN simulations can be leveraged to estimate the relevant, large-scale statistics more reliably than with traditional methods, which can be helpful in fusion research.

References

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

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

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

Y. Gal and Z. Ghahramani. 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, page 1050–1059. JMLR.org, 2016.

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

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

G. E. Hinton and D. van Camp. 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, pages 5–13, New York, NY, USA, Aug. 1993. Association for Computing Machinery. ISBN 978-0-89791-611-0. doi: 10.1145/168304.168306. URL `https://doi.org/10.1145/168304.168306`.

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

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

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

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

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

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

D. J. C. MacKay. 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.

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

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

N. Thuerey, K. Weissenow, H. Mehrotra, N. Mainali, L. Prantl, and X. Hu. 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.

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

M. Wakatani and A. Hasegawa. 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.

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

F. Wenzel, K. Roth, B. S. Veeling, J. Swiatkowski, L. Tran, S. Mandt, J. Snoek, T. Salimans, R. Jenatton, and S. Nowozin. 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.

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

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] We describe the detailed experimental setup and, where applicable, the assumptions made. We specify the used fluid models throughout the paper. We do not state assumptions made implicitly in the fluid models.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] We mention possible applications of our work in the Broader Impact statement.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] The code for the first experiment will be made available upon acceptance. The code for the second experiment is still work in progress.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] In section 3 we provide the training details.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] Given the limited computational resources, we decided to perform a parameter sweep over the dropout rate rather than repeated runs for multiple seeds.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [No]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [N/A]

(c) Did you include any new assets either in the supplemental material or as a URL? [N/A]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

5 Appendix

The appendix is structured as follows: In section 5.1 and 5.2 we provide information on the network architectures. In section 5.3 we show additional plots for the RANS-flow experiment. In section 5.4 we show details of the turbulent plasma flow experiment. The solver-in-the-loop training algorithm is explained in more detail in section 5.5. In section 5.6 we provide more details on BNNs and the flipout estimator, which is used in the experiment presented in section 5.7.

5.1 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.

5.2 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.

5.3 RANS-Flow

For completeness, we provide all 90 test samples together with their marginal prediction and the corresponding uncertainty. Figure 3 illustrates the spatial dropout implementation with rate 0.01, and figure 4 the conventional dropout implementation with rate 0.1. Further, we show repeated samples for varying dropout rates, again for spatial dropout in figure 5 and conventional dropout in figure 6. Figure 8 shows the uncertainty-MAE relation of all test samples for a BNN with spatial dropout rate 0.1.

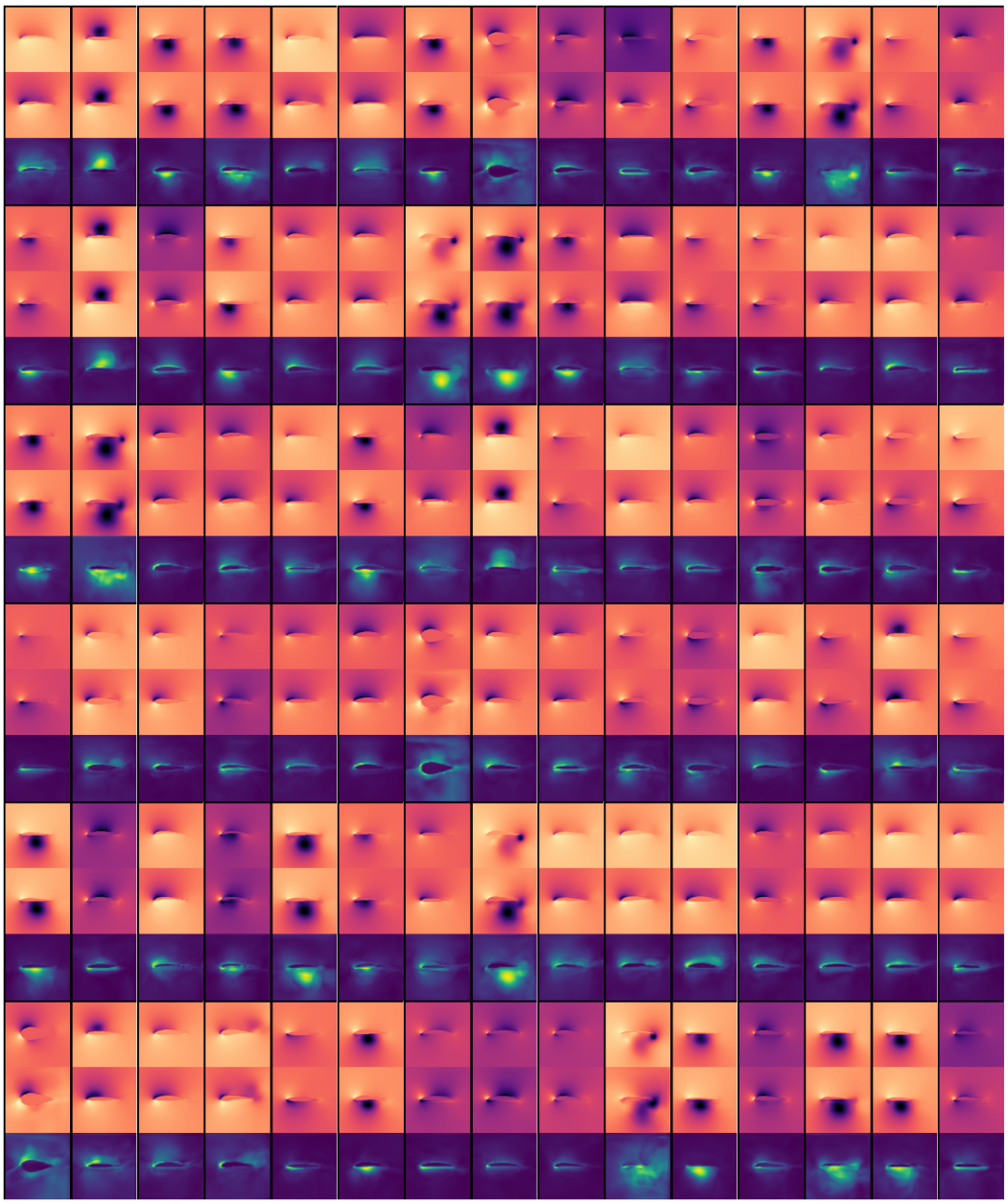

Figure 3: All test samples for a BNN with spatial dropout and dropout rate 0.01. For each sample, the target is shown on top, the average prediction in the middle, and the corresponding uncertainty (in different color map) on the bottom.

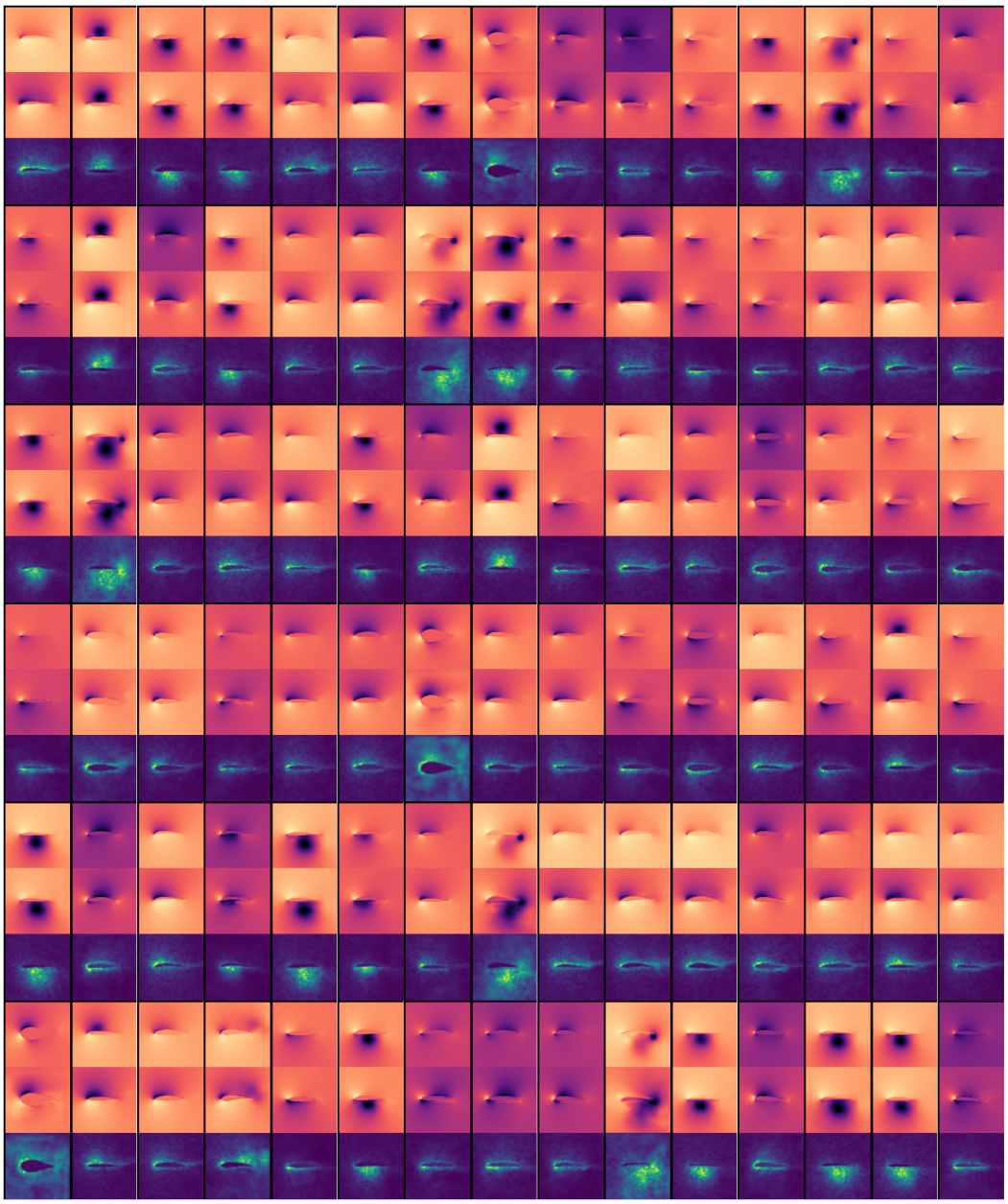

Figure 4: All test samples for a BNN with regular dropout and dropout rate 0.1. For each sample, the target is shown on top, the average prediction in the middle, and the corresponding uncertainty (in different color map) on the bottom.

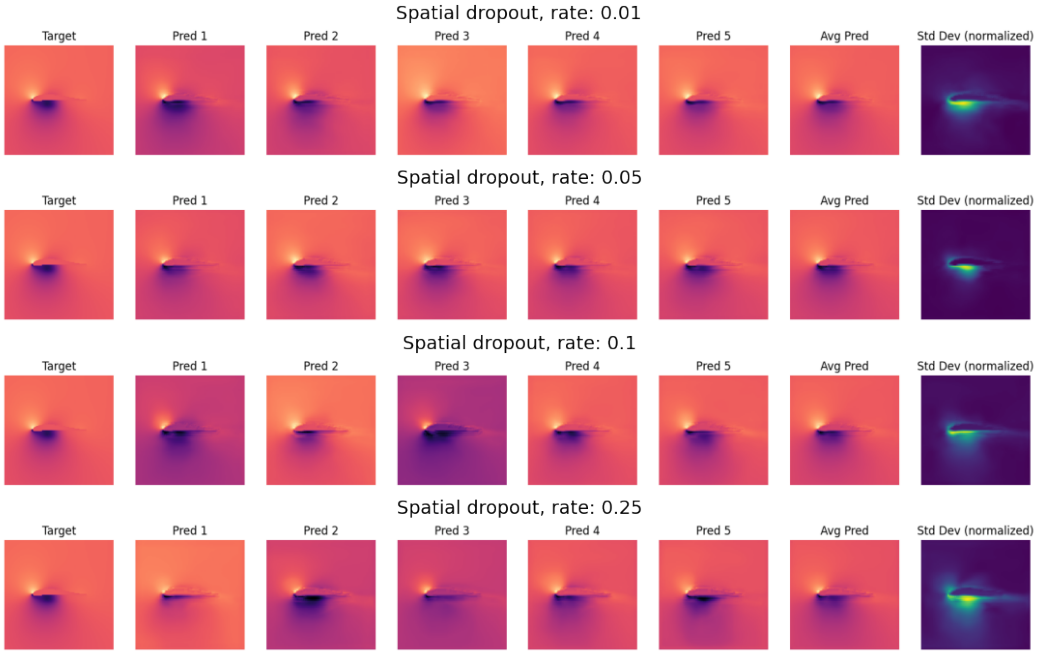

Figure 5: BNN samples (spatial dropout, different rates) for RANS flow.

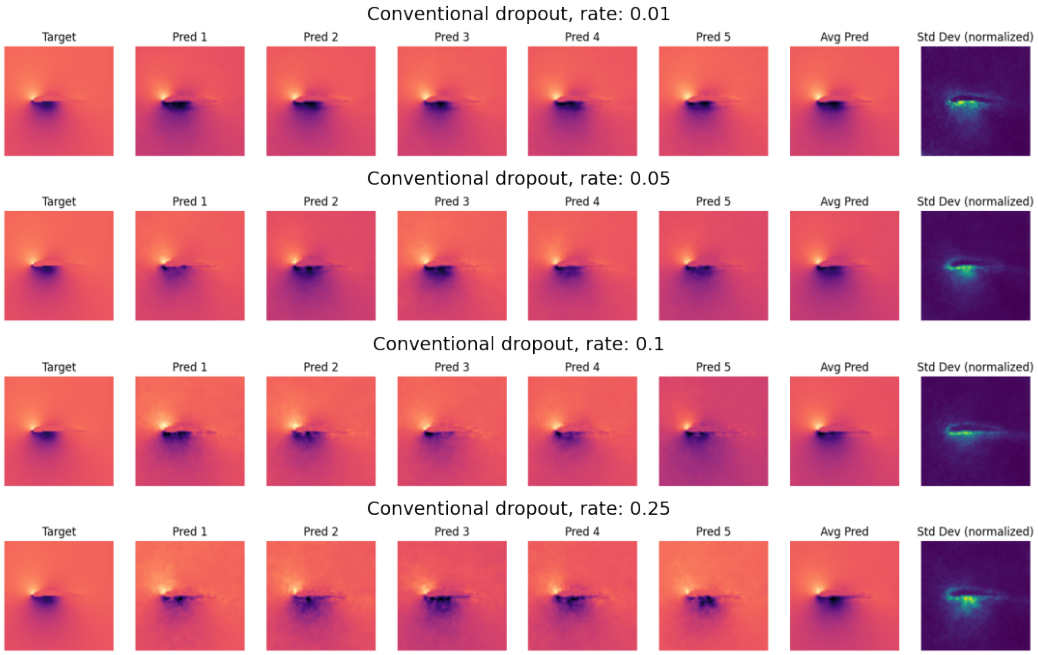

Figure 6: BNN samples (regular dropout, different rates) for RANS flow.

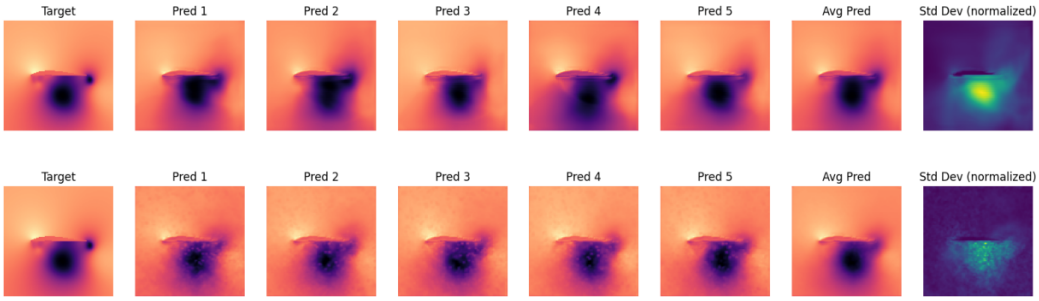

Figure 7: Repeated samples from BNN with spatial dropout (top) and conventional dropout (bottom) for a specific (hard) test case. The individual predictions of the BNN with spatial dropout are smoother and show larger variations compared to the conventional dropout case, where the difference between predictions is on a smaller scale.

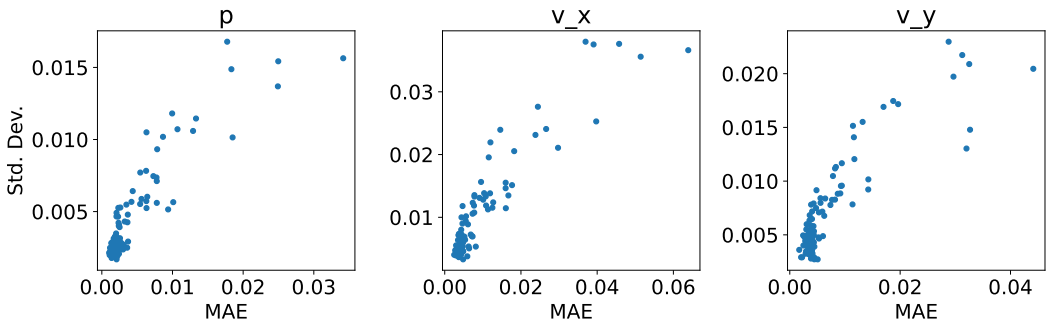

Figure 8: MAE vs. uncertainty per test sample for a BNN with spatial dropout rate 0.1: Larger uncertainty typically implies larger error.

5.4 Turbulent plasma simulations

The Hasegawa-Wakatani system Hasegawa and Wakatani [1983], Wakatani and 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

$$\frac{\partial}{\partial t}\nabla_\perp^2\tilde{\phi} + (\hat{\mathbf{z}} \times \nabla_\perp) \cdot \nabla_\perp \nabla_\perp^2 \tilde{\phi} = \alpha(\tilde{\phi} - \tilde{n}) + D^\phi \tag{1}$$

$$\frac{\partial}{\partial t}\tilde{n} + (\hat{\mathbf{z}} \times \nabla_\perp) \cdot \nabla_\perp \tilde{n} = \alpha(\tilde{\phi} - \tilde{n}) + D^n \tag{2}$$

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 1 and equation 2 are cross-coupled through the adiabaticity parameter $\alpha \propto \frac{TL_n}{n_0 e^2 c_s}$. 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

$$\mathbf{\Omega} = \nabla \times \mathbf{v}_D \propto \nabla \times (-\nabla\phi \times \hat{\mathbf{z}}) = \nabla^2\phi\hat{\mathbf{z}}. \tag{3}$$

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.

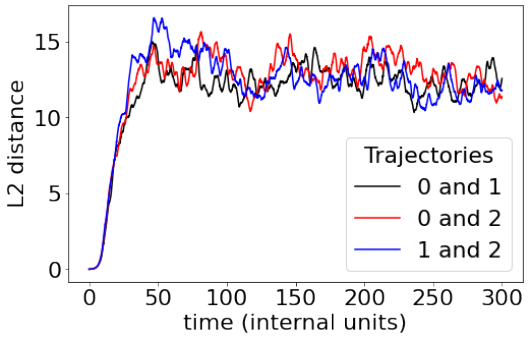

Figure 9: $L2$-distance between 3 BNN-corrected density trajectories: The initial correlation fades out after only about 30 internal time units (which is about 10 times the autocorrelation time).

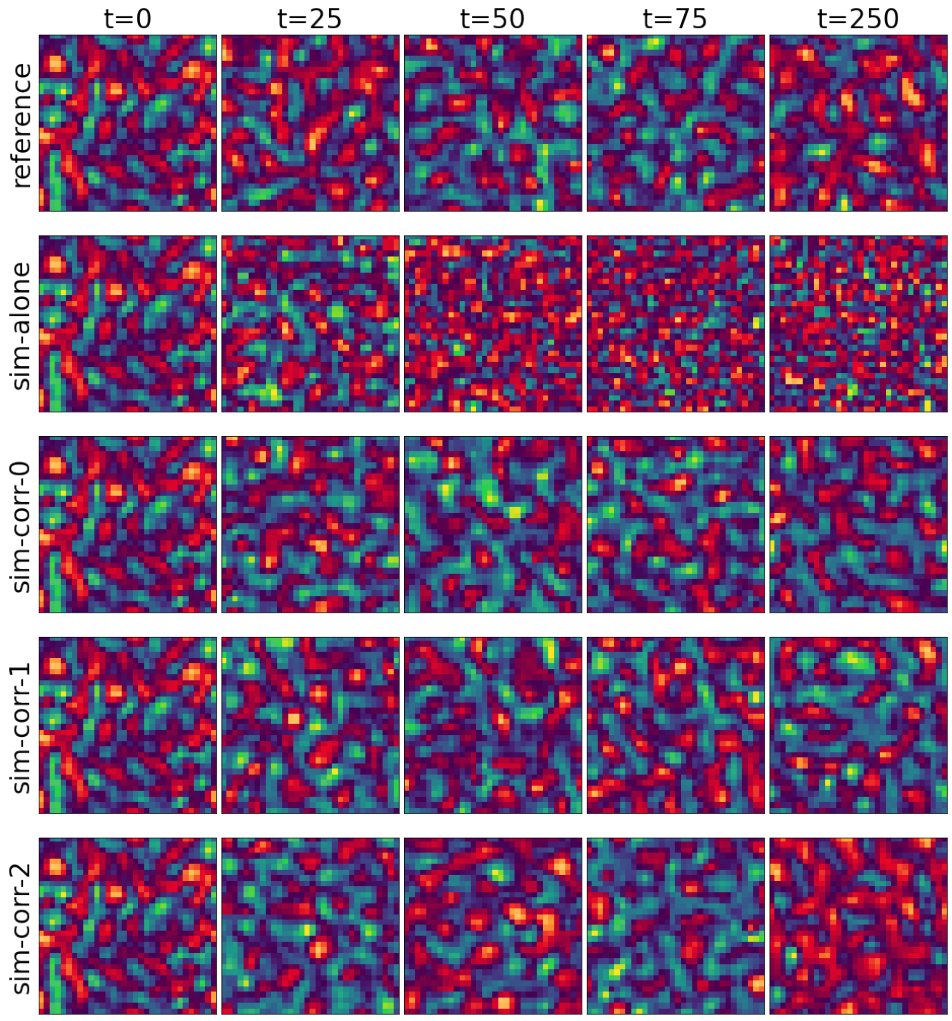

Figure 10: ϕ field

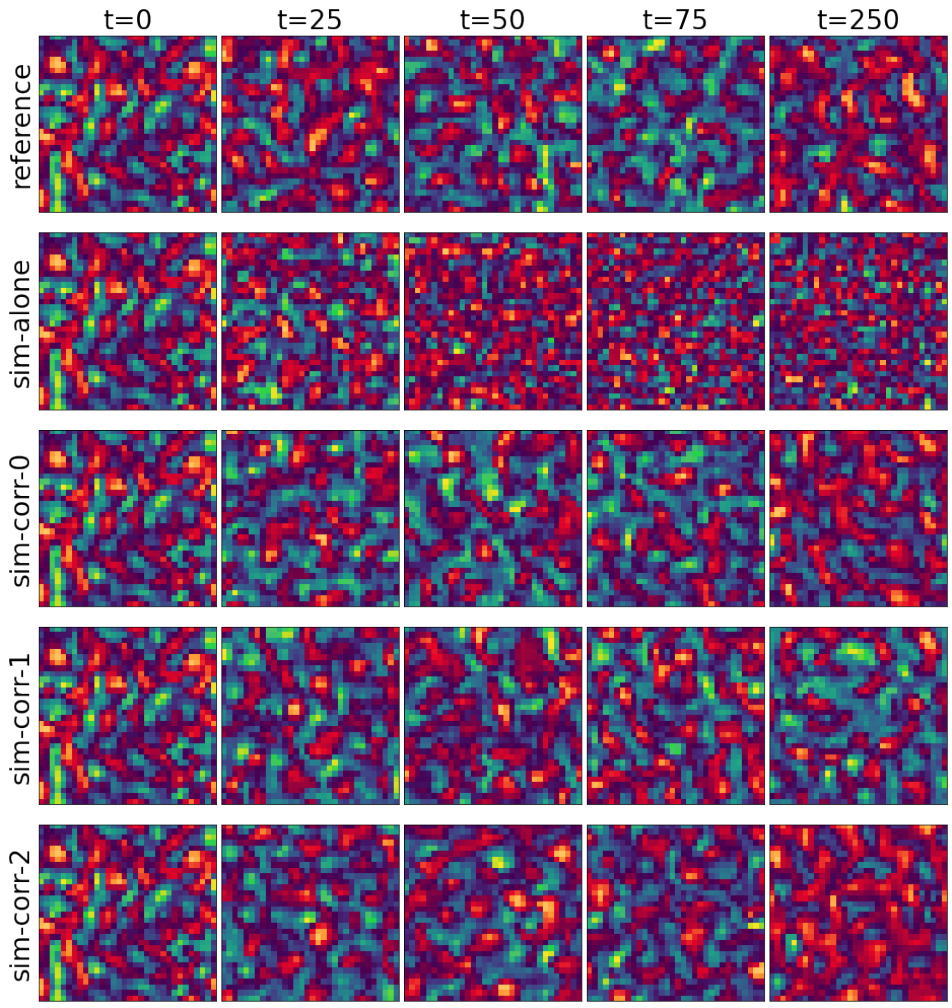

Figure 11: Density n

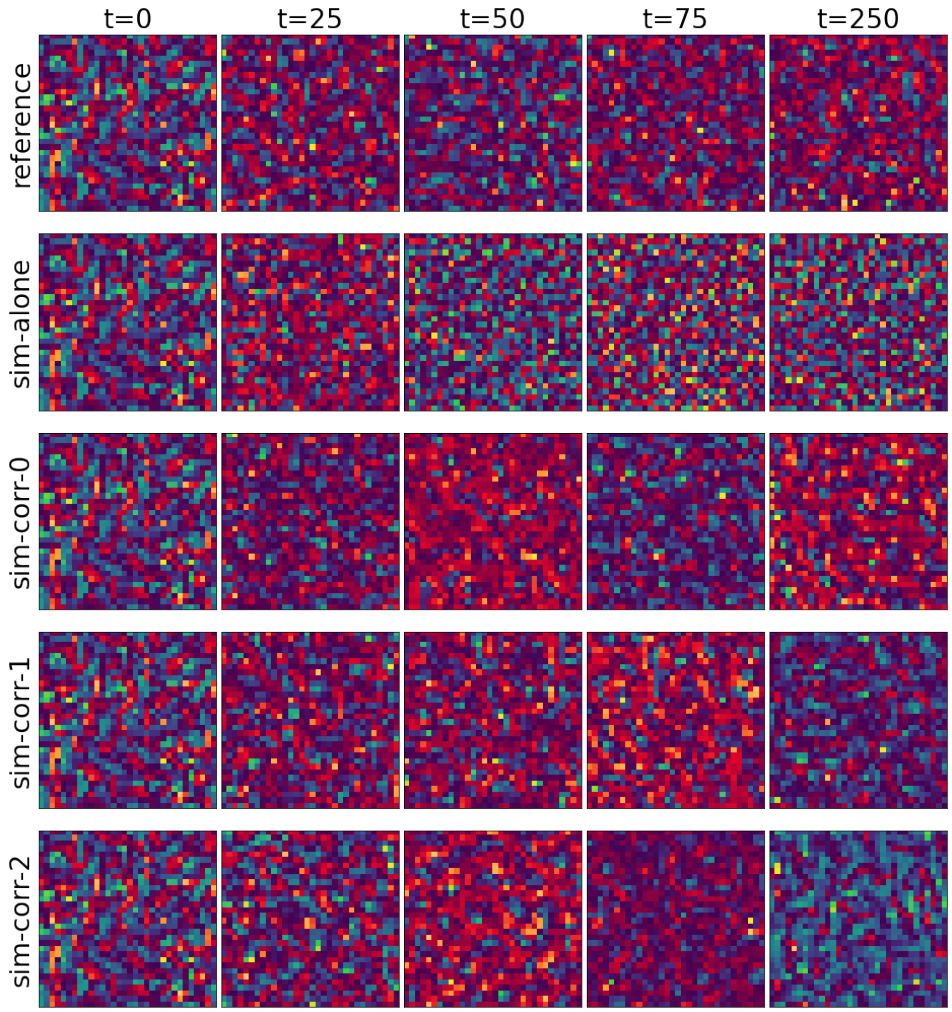

Figure 12: Vorticity Ω

5.5 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

$$\arg\min_{\theta} \sum_{i=0}^{n-1} \|\mathcal{C}(\mathcal{P}_S(\tilde{\mathbf{s}}_{t+i})|\theta) - \mathcal{T}\mathbf{r}_{t+i+1}\|_2^2 \,. \tag{4}$$

The subtlety of $equation\ 4$ 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 13. 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 $\boldsymbol{\theta}$ *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

$$\mathcal{L} = \sum_{i=0}^{L-1} \|\mathcal{C}(\mathcal{P}_S(\tilde{\mathbf{s}}_{j+i})|\theta) - \mathcal{T}\mathbf{r}_{j+i+1}\|_2^2 . \tag{5}$$

    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

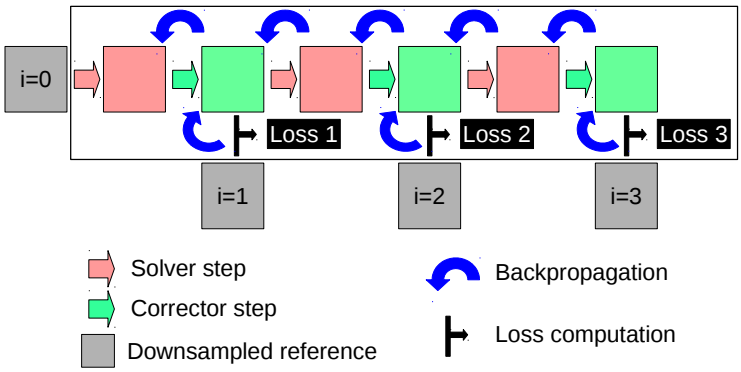

Figure 13: Sketch of the solver-in-the-loop with a look-ahead of $L = 3$ and batch size $B = 1$: Starting from a downsampled reference at $i = 0$, solver and corrector are applied iteratively 3 times. Each corrected frame is compared to the corresponding downsampled reference frame and the respective losses are computed. Those losses are then backpropagated through the solver and the corrector in order to update the parameters of the corrector network.

5.6 Flipout BNNs

BNNs based on variational inference [Kingma et al., 2015] aim at minimizing the KL-divergence between true and approximate posterior, or equivalently maximize the evidence lower bound (ELBO):

$$\mathbb{E}_{\boldsymbol{w}\sim q_{\boldsymbol{\theta}}}[\log p(\boldsymbol{Y}|\boldsymbol{w}, \boldsymbol{X})] - \frac{1}{\lambda} D_{\mathrm{KL}}\left(q_\theta(\boldsymbol{w}) \| p(\boldsymbol{w})\right) \tag{6}$$

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 6 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 $\lambda \to 1$ we have a Bayesian network where the log-likelihood and the KL-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 6, and in particular derivatives thereof, sub-sampling and Monte-Carlo techniques are typically leveraged together with a reparametrization trick [Kingma and Welling,

2014], that allows to backpropagate gradients through distributions. One particular stochastic estimator is the *flipout* estimator [Wen et al., 2018], which computes decorrelated stochastic gradient estimates of equation 6 with few perturbation samples. Gal and Ghahramani [2016] showed that under mild conditions, the optimization procedures of a BNN, i.e. minimizing Equation 6, and training a Neural Network with dropout and weight decay are equivalent.

5.7 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 16. 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 5.7 together with a formal description and a visualization (figure 17) of the perturbation operator $\mathcal{Q}$. The learning goal in this setup is to predict the velocity profiles advanced by n simulation steps from the current marker profile, i.e. minimizing

$$\mathbb{E}_{\boldsymbol{w}\sim q_{\boldsymbol{\theta}}}\left[\sum_{i=1}^{N-n}\sum_{t=1}^{T}\left\|f(\mathbf{d}_i^t|\boldsymbol{w}) - \mathbf{v}_{i+n}^t\right\|_1\right] - \frac{1}{\lambda}D_{\mathrm{KL}} \tag{7}$$

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 14, 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 16 in the appendix). In such cases, the network is typically not capable of reasonably approximating the target distribution. In figure 14 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, see fig. 18) 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 be seen qualitatively in figure 15. 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 15, 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

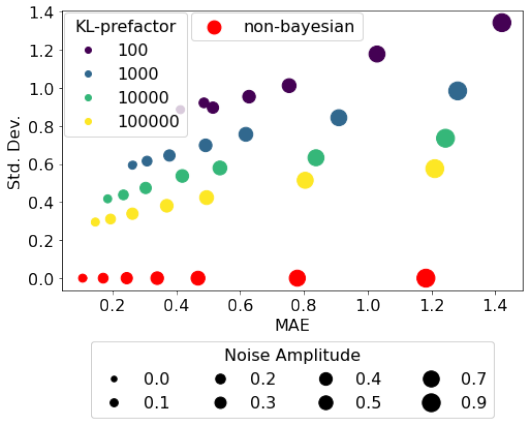

Figure 14: Uncertainty vs. mean absolute error for perturbed buoyancy-driven Navier-Stokes data. The performance of the conventional, non-Bayesian network is shown in red.

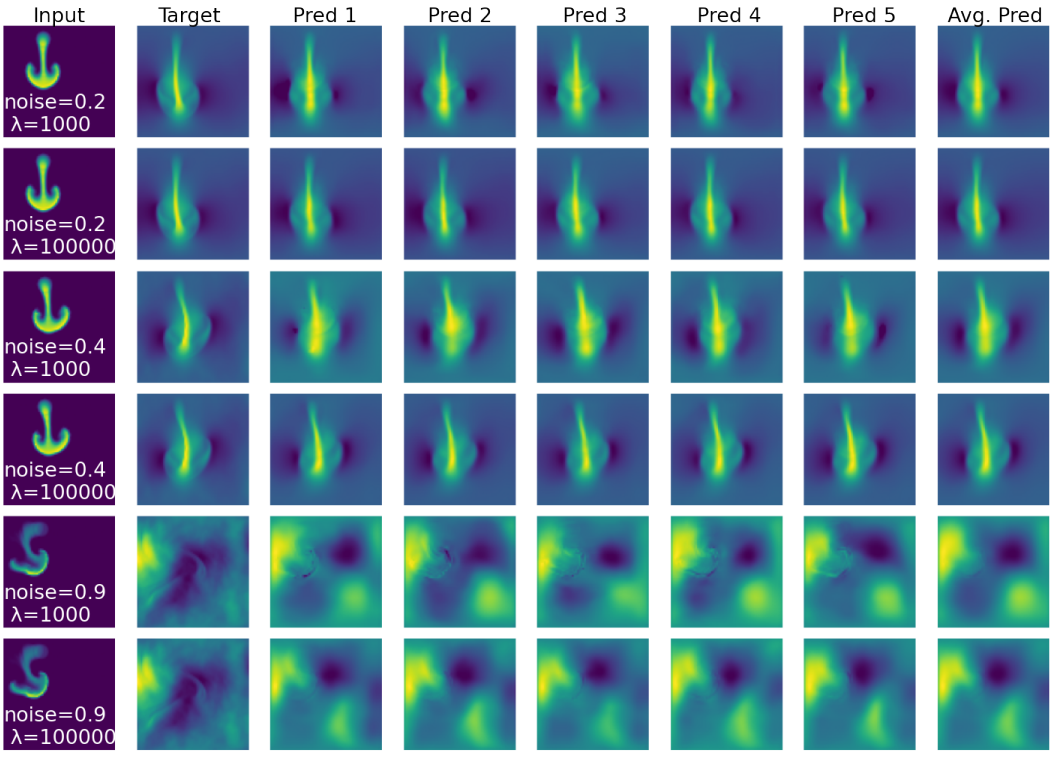

Figure 15: Repeated samples for different noise-per-frame training setups with $\lambda \in \{10^3, 10^5\}$ and noise amplitude $\in \{0.2, 0.4, 0.9\}$.

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. This might be due to the implicitly assumed Laplace likelihood not being able to truly distinguish between ambiguous solutions and instead averaging over them.

Fluid Model. 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 17 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 16 shows samples of a perturbed trajectory, i.e. where $\mathcal{Q}$ was applied before every solver step. Figure 18 shows the result of a linear regression between standard deviation and mean absolute error for $\lambda = 100$.

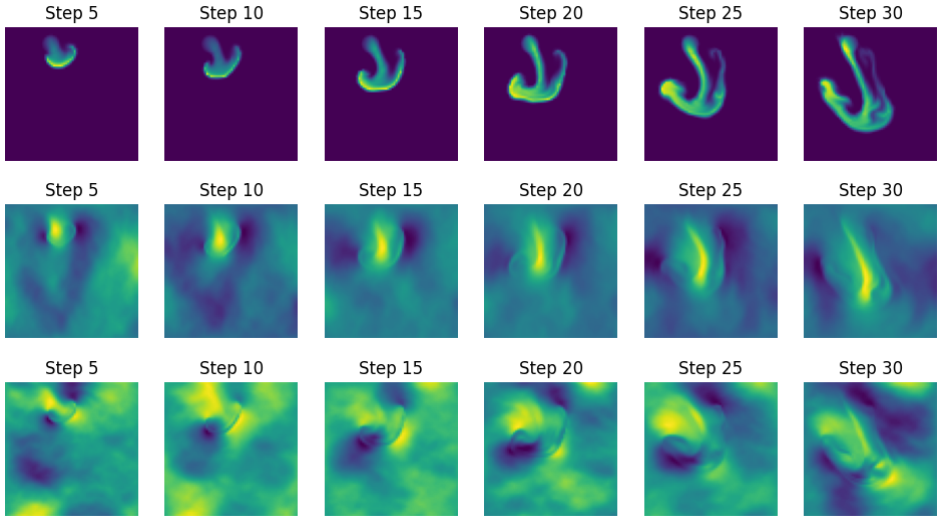

Figure 16: Trajectories of marker field, $x-$component and $y-$component of velocity profiles for dynamically perturbed Navier-Stokes flow.

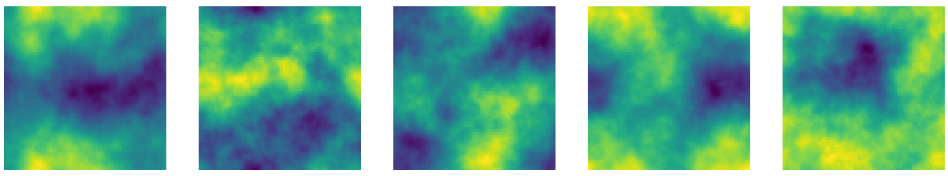

Figure 17: Samples of the noise fields which are added to the velocity fields by the perturbation operator $\mathcal{Q}$.

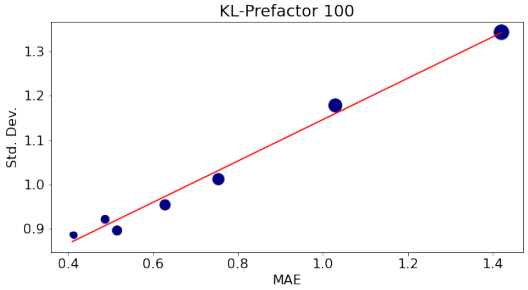

Figure 18: For $\lambda = 100$, a linear regression between standard deviation and mean absolute error yields a slope of 0.47 and an intercept of 0.68.