Active Learning for Neural PDE Solvers

Daniel Musekamp[1,3] **Marimuthu Kalimuthu**[1,2,3] **David Holzmüller**[4]
Makoto Takamoto[5] **Mathias Niepert**[1,2,3,5]

[1]University of Stuttgart [2]SimTech [3] IMPRS-IS
[4]INRIA Paris, Ecole Normale Supérieure, PSL University [5]NEC Labs Europe
daniel.musekamp@ki.uni-stuttgart.de

Abstract

Solving partial differential equations (PDEs) is a fundamental problem in engineering and science. While neural PDE solvers can be more efficient than established numerical solvers, they often require large amounts of training data that is costly to obtain. Active Learning (AL) could help surrogate models reach the same accuracy with smaller training sets by querying classical solvers with more informative initial conditions and PDE parameters. While AL is more common in other domains, it has yet to be studied extensively for neural PDE solvers. To bridge this gap, we introduce AL4PDE, a modular and extensible AL benchmark. It provides multiple parametric PDEs and state-of-the-art surrogate models for the solver-in-the-loop setting, enabling the evaluation of existing and the development of new AL methods for PDE solving. We use the benchmark to evaluate batch active learning algorithms such as uncertainty- and feature-based methods. We show that AL reduces the average error by up to 71% compared to random sampling and significantly reduces worst-case errors. Moreover, AL generates similar datasets across repeated runs, with consistent distributions over the PDE parameters and initial conditions. The acquired datasets are reusable, providing benefits for surrogate models not involved in the data generation.

1 Introduction

Neural PDE solvers [48, 26, 6, 11, 25] promise a faster alternative to classical numerical solvers, while being also end-to-end differentiable. A main challenge of neural PDE surrogates is that their training data is often obtained from the same expensive simulators they are intended to replace. Hence, training a surrogate provides a computational advantage only if the generation of the training data set requires fewer simulations than will be saved during inference. Moreover, it is non-trivial to obtain training data covering all challenging dynamical regimes sufficiently. AL is a possible solution to these challenges as it iteratively selects the most informative and diverse training trajectories, thereby reducing the total number of simulations required to reach the same level of accuracy. AL has recently been applied to PDEs in the context of PINNs [56, 45, 1], specific PDE domains [36, 37], or direct prediction models [24, 25]. For example, AL has been applied to the stationary solution of a diffusion problem [5] and to finding extreme events [38]. In multi-fidelity AL, the optimal spatial resolution of the simulation is chosen [22, 23, 57]. Li et al. [24] use an ensemble of FNOs in the single prediction setting. Wu et al. [58] apply AL to stochastic simulations using a spatio-temporal neural process. Hence, AL is still unexplored for a broader class of neural PDE solvers, which currently rely on extensive numerical simulations to generate a sufficient amount of training data. Thus, we introduce AL4PDE, which, contrary to prior benchmarks [48, 11, 12, 28, 27], is the first framework for evaluating and developing AL methods for neural PDE solvers. In addition to various AL algorithms, the extensible framework provides numerical simulators for multiple PDEs (e.g., Navier-Stokes) and neural surrogates (e.g., SineNet [63]). An initial study shows that AL can

D3S3: Data-driven and Differentiable Simulations, Surrogates, and Solvers @ NeurIPS 2024.

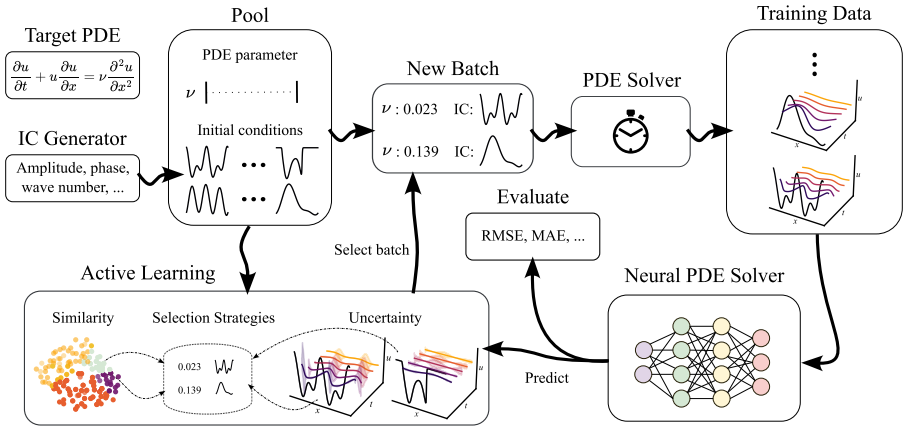

Figure 1: An extensible benchmark framework for pool-based active learning of neural PDE solvers.

increase the data efficiency and especially reduces worst-case errors. The generated data distribution is consistent between random repetitions, showing that AL reliably generates reusable datasets.

2 Background

We seek the solution $\mathbf{u} : [0, T] \times \mathcal{X} \to \mathbb{R}^{N_c}$ of a PDE with a D-dimensional spatial domain $\mathcal{X}$, $\mathbf{x} = [x_1, x_2, \ldots, x_D]^\top \in \mathcal{X}$, temporal domain $t \in [0, T]$, and N_c field variables or channels c [6]:

$$\partial_t \mathbf{u} = F\left(\boldsymbol{\lambda}, t, \mathbf{x}, \mathbf{u}, \partial_{\mathbf{x}} \mathbf{u}, \partial_{\mathbf{x}\,\mathbf{x}} \mathbf{u}, \ldots\right), \qquad (t, \mathbf{x}) \in [0, T] \times \mathcal{X} \tag{1}$$

$$\mathbf{u}(0, \mathbf{x}) = \mathbf{u}^0(\mathbf{x}), \quad \mathbf{x} \in \mathcal{X}; \qquad \mathcal{B}[\mathbf{u}](t, \mathbf{x}) = 0, \quad (t, \mathbf{x}) \in [0, T] \times \partial\mathcal{X} \tag{2}$$

Here, the boundary condition $\mathcal{B}$ (Eq. 2) determines the behavior of the solution at the boundaries $\partial\mathcal{X}$ of the spatial domain $\mathcal{X}$, and the initial condition (IC) $\mathbf{u}^0$ defines the initial state of the system (Eq. 2). The vector $\boldsymbol{\lambda} = (\lambda_1, ..., \lambda_l)^\top \in \mathbb{R}^l$ with $\lambda_i \in [a_i, b_i]$ denotes the PDE parameters. We only consider a single boundary condition (periodic) for simplicity, and thus a single initial value problem can be identified by the tuple $\boldsymbol{\psi} = (\mathbf{u}^0, \boldsymbol{\lambda})$. The inputs to the initial value problem are drawn from p_T, $\boldsymbol{\psi} \sim p_T(\boldsymbol{\psi}) = p_T(\mathbf{u}^0)p_T(\boldsymbol{\lambda})$. The solution $\mathbf{u}$ is uniformly discretized across the spatial dimensions, yielding $N_{\mathbf{x}}$ spatial points in total and the temporal dimension into N_t timesteps. We aim to replace the numerical solver with a neural, autoregressive PDE solver $\mathcal{G}_\theta$ with $\hat{\mathbf{u}}(t + \Delta t, \cdot) = \mathcal{G}_\theta(\hat{\mathbf{u}}(t, \cdot), \boldsymbol{\lambda})$ [26]. The network parameters θ are optimized on training samples $\mathcal{S}_{\text{train}} = \{(\boldsymbol{\psi}_1, \mathbf{u}_1), \ldots, (\boldsymbol{\psi}_{N_{\text{train}}}, \mathbf{u}_{N_{\text{train}}})\}$ using the root mean squared error (RMSE).

3 AL4PDE: An AL Framework for Neural PDE Solvers

The AL4PDE benchmark consists of three major parts: (1) AL algorithms, (2) surrogate models, and (3) PDEs and the corresponding simulators. It follows a modular design to make the addition of new approaches or problems as easy as possible (Fig. 5).

Active Learning Setup AL aims to select the most informative training samples so that the model can reach the same generalization error with fewer calls to the numerical solver. We measure the error using test trajectories on random samples from an input distribution p_T. Fig. 1 shows the full AL cycle. Since it requires retraining the NN(s) after each round, we use batch AL with sufficiently large batches. Specifically, in each round, a batch of simulator inputs $\mathcal{S}_{\text{batch}} = \{\boldsymbol{\psi}_1, ..., \boldsymbol{\psi}_{N_{\text{batch}}}\}$ is selected. We implement *pool-based* AL, which selects from a set of possible inputs $\mathcal{S}_{\text{pool}} = \{\boldsymbol{\psi}_1, ..., \boldsymbol{\psi}_{N_{\text{pool}}}\}$ called "pool". The selected batch $\mathcal{S}_{\text{batch}}$ is then removed from the pool, simulated, and added to the training set $\mathcal{S}_{\text{train}}$. We sample the pool set randomly from the test input distribution p_T. The initial batch is selected randomly. Since neural PDE solvers provide high-dimensional autoregressive rollouts without direct uncertainty predictions, many AL methods cannot be applied straightforwardly. We select AL methods based on uncertainty and features [15]. As a generic baseline, we compare to the selection of a (uniformly) **random** sampling of the inputs, $\boldsymbol{\psi} \sim p_T(\boldsymbol{\psi})$.

Uncertainty-based AL Epistemic uncertainty is often used as a measure of sample informativeness. We adopt the query-by-committee (QbC) approach [47], a simple but effective method that utilizes the variance between the ensemble members' outputs as an uncertainty estimate:

$$a_{\text{QbC}}(\boldsymbol{\psi}_i) := \frac{1}{N_t N_{\mathbf{x}} N_c} \sum_{j=1}^{N_t} \sum_{k=1}^{N_{\mathbf{x}}} \frac{1}{N_m} \sum_{m=1}^{N_m} \|\hat{\mathbf{u}}_{i,m}(t_j, \mathbf{x}_k) - \overline{\hat{\mathbf{u}}_i}(t_j, \mathbf{x}_k)\|_2^2 \,. \tag{3}$$

Here, $\overline{\hat{\mathbf{u}}_i}$ is the mean prediction of all N_m models with $\overline{\hat{\mathbf{u}}_i}(t, \mathbf{x}) = \sum_m \hat{\mathbf{u}}_{i,m}(t, \mathbf{x})/N_m$. The ensemble members produce different outputs $\hat{\mathbf{u}}_i$ due to the inherent randomness resulting from the weight initialization and stochastic optimization. The assumption of QbC is that the variance of the ensemble member predictions correlates positively with the error. A high variance, therefore, points to a region of the input space where we need more data. When given a single-sample acquisition function a, such as the ensemble uncertainty, a simple and common approach to selecting a batch is taking the k most uncertain samples (**Top-K**). However, this does not ensure that the selected batch is diverse. Stochastic batch active learning (**SBAL**) samples inputs $\boldsymbol{\psi}$ from the remaining pool set $\mathcal{S}_{\text{pool}}$ without replacement according to the probability distribution $p_{\text{power}}(\boldsymbol{\psi}) \propto a(\boldsymbol{\psi})^m$, where m is a hyperparameter controlling the sharpness of the distribution [18]. Hence, it also selects samples from regions that are not from the highest mode of the uncertainty distribution and encourages diversity.

Feature-based AL Many deep batch AL methods rely on some feature representation $\phi(\boldsymbol{\psi}) \in \mathbb{R}^p$ of inputs and utilize a distance metric in the feature space as a proxy for the similarity between inputs, which can help to ensure diversity of the selected batch. Moreover, they only need a single model. We compute the trajectory and concatenate the spatially averaged latent features at each timestep. Additionally, Gaussian sketching is applied [15]. In the simpler version of their **Core-Set** algorithm, Sener and Savarese [46] iteratively select the input from the remaining pool with the highest distance to the closest selected or labeled point. While Core-Set produces batches of diverse and informative samples, it does not select samples that are representative of the proposal distribution. To alleviate this issue, Holzmüller et al. [15] propose to replace the greedy Core-Set with **LCMD**, a method inspired by k-medoids clustering. LCMD interprets previously selected inputs as cluster centers, assigns all remaining pool points to their closest center, selects the cluster with the largest sum of squared distances to the center, and from this cluster selects the point that is furthest away from the center. This point then becomes a new center and the process is repeated until the batch is complete.

PDEs and Surrogates We consider 1D and 2D parametric PDEs (full details in Appendix B). The first 1D PDE is the **Burgers**' equation from PDEBench [48] with the kinematic viscosity as the PDE parameter. Secondly, the Kuramoto–Sivashinsky (**KS**) equation from Lippe et al. [26] that demonstrates diverse dynamical behaviors, from fixed points and periodic limit cycles to chaos [16]. Next to the viscosity, the domain length is also varied. Thirdly, to test a multiphysics problem with more parameters, we include the so-called combined equation (**CE**) from Brandstetter et al. [6] although without the forcing term. Depending on the value of the three PDE coefficients, this equation recovers the Heat, Burgers, or the Korteweg-de-Vries PDE. For 2D, we use the compressible Navier-Stokes (**CNS**) equations from PDEBench [48], The ICs are generated from random initial fields. Regarding the surrogates, we include (i) a recent version of U-Net [44] from Gupta and Brandstetter [11], (ii) SineNet [63], an enhancement of modern U-Net that corrects the feature misalignment in the residual connections of U-Net, and (iii) the Fourier neural operator [FNO, 25].

4 Selection of Experiments

We perform an initial study on the behavior of AL using AL4PDE. We use a smaller version of the modern U-Net [11], which is trained using sub-trajectories (two steps) to strike a balance between learning to rollout and fast training. In each AL iteration, the amount of data added is equal to the current training set size [18]. The pool size is fixed to 100,000 candidates and two ensemble members are used to measure uncertainty. We repeat all experiments with five random seeds (Burgers: ten) and report the 95% confidence interval of the mean. Fig. 2 shows the RMSE for the various AL methods and PDEs. AL often reduces the error compared to sampling uniformly at random for the same amount of data. The advantage is especially large for CE, which is likely due to the diverse dynamic regimes found in the PDE. SBAL and LCMD achieve similar errors on all PDEs with the exception of KS, where only SBAL can improve over random sampling. SBAL and LCMD can reach

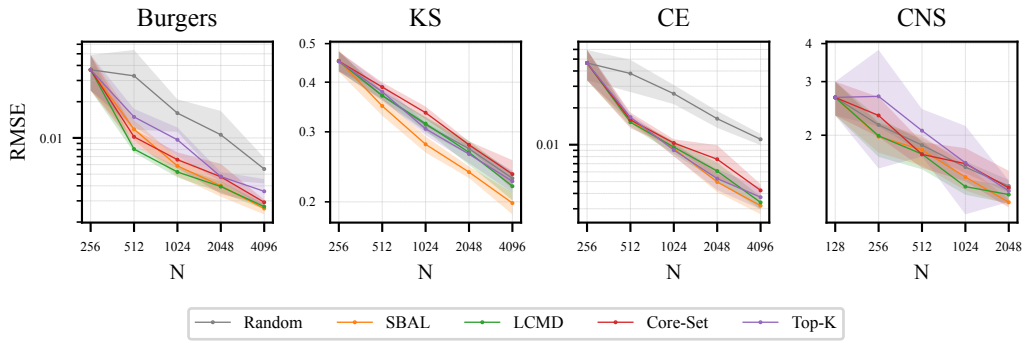

Figure 2: Error over the number of training trajectories (N). AL can reduce the error relative to random sampling of the inputs on all tested PDEs but CNS, where the difference was not significant.

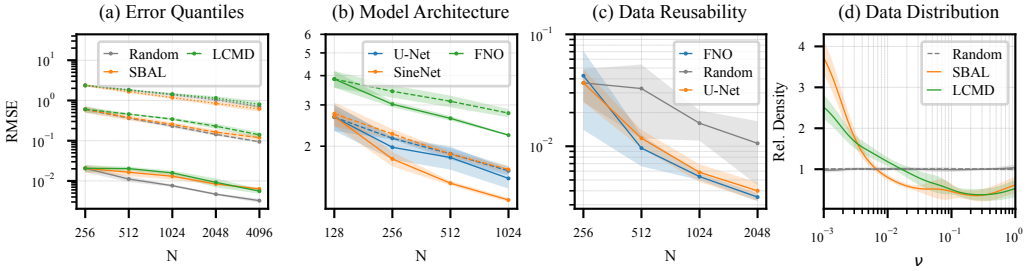

Figure 3: (a) Error quantiles on KS. The 50%, 95%, and 99% quantiles are displayed using full, dashed, and dotted lines, respectively. AL especially improves the higher error quantiles, making the trained model more reliable. (b) Different base models on CNS using SBAL (solid) and random sampling (dashed). SBAL can also improve the accuracy of other models besides the U-Net. (c) Error of the standard U-Net on Burgers, with data selected using FNO or U-Net with SBAL. The selected data is also helpful for a model not used during AL. (d) Marginal distribution of the diffusion parameter of Burgers in the training set generated by AL (relative to the uniform distribution). The shaded area represents the standard deviation between the random seeds. The distribution a small standard deviation, indicating that AL reliably generates similar datasets between independent runs.

lower error values with only a quarter of the data points in the case of CE and Burgers. However, the greedy methods Top-K and Core-Set even increase the error for some PDEs. The difference in the CNS task was not significant, likely due to the performance of the base model training (see Fig. 3b) for a stronger model). Worst-case errors are of special interest when solving PDEs. Since we found the absolute maximum error to be unstable, we show the RMSE quantiles in Fig. 3a). Notably, AL reduces the higher quantiles while the 50 % percentile error is increased. The marginal distributions of the PDE parameter of Burgers equation are shown in Fig. 3d). These distributions are highly similar for different random seeds, and thus, AL reliably selects similar training datasets. The various AL methods generally sample similar parameter values but can differ substantially in certain regions of the parameter space (Appendix G). To investigate the effect of the generated data on other models, we use an FNO ensemble to select the data that we use to train the standard U-Net. Fig. 3c) depicts the error of the U-Net, showing that the selected data is beneficial for models not used for the AL-based data selection. The reusability of the data is especially important since, otherwise, the whole AL procedure would have to be repeated every time a new model is developed.

5 Conclusion

This paper introduces AL4PDE, an extensible framework to develop and evaluate AL algorithms for neural PDE solvers. An initial study shows that existing AL algorithms can already allow a model to reach the same accuracy with up to four times fewer data points, produces consistent as well as reusable datasets, and works well across surrogate architectures.

Acknowledgments and Disclosure of Funding

We acknowledge the support of the German Federal Ministry of Education and Research (BMBF) as part of InnoPhase (funding code: 02NUK078). Marimuthu Kalimuthu and Mathias Niepert are funded by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2075 – 390740016. We acknowledge the support of the Stuttgart Center for Simulation Science (SimTech). The authors thank the International Max Planck Research School for Intelligent Systems (IMPRS-IS) for supporting Marimuthu Kalimuthu, Daniel Musekamp, and Mathias Niepert. Moreover, the authors gratefully acknowledge the computing time provided to them at the NHR Center NHR4CES at RWTH Aachen University (project number p0021158). This is funded by the Federal Ministry of Education and Research, and the state governments participating on the basis of the resolutions of the GWK for national high-performance computing at universities (http://www.nhr-verein.de/unsere-partner).

References

[1] Y. Aikawa, N. Ueda, and T. Tanaka. Improving the efficiency of training physics-informed neural networks using active learning. In *The 37th Annual Conference of the Japanese Society for Artificial Intelligence*, 2023.

[2] C. J. Arthurs and A. P. King. Active training of physics-informed neural networks to aggregate and interpolate parametric solutions to the navier-stokes equations. *J. Comput. Phys.*, 438: 110364, 2021.

[3] J. Ash, S. Goel, A. Krishnamurthy, and S. Kakade. Gone fishing: Neural active learning with fisher embeddings. *Advances in Neural Information Processing Systems*, 34:8927–8939, 2021.

[4] J. T. Ash, C. Zhang, A. Krishnamurthy, J. Langford, and A. Agarwal. Deep batch active learning by diverse, uncertain gradient lower bounds. In *International Conference on Learning Representations*, 2019.

[5] P. Bajracharya, J. Q. Toledo-Marín, G. Fox, S. Jha, and L. Wang. Feasibility study on active learning of smart surrogates for scientific simulations. *arXiv preprint arXiv:2407.07674*, 2024.

[6] J. Brandstetter, D. E. Worrall, and M. Welling. Message passing neural pde solvers. In *International Conference on Learning Representations*, 2021.

[7] J. Brandstetter, R. van den Berg, M. Welling, and J. K. Gupta. Clifford neural layers for PDE modeling. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*, 2023.

[8] G. Dresdner, D. Kochkov, P. C. Norgaard, L. Zepeda-Nunez, J. Smith, M. Brenner, and S. Hoyer. Learning to correct spectral methods for simulating turbulent flows. *Transactions on Machine Learning Research*, 2023.

[9] W. Gao and C. Wang. Active learning based sampling for high-dimensional nonlinear partial differential equations. *Journal of Computational Physics*, 475:111848, 2023.

[10] Y. Geifman and R. El-Yaniv. Deep active learning over the long tail. *arXiv preprint arXiv:1711.00941*, 2017.

[11] J. K. Gupta and J. Brandstetter. Towards multi-spatiotemporal-scale generalized pde modeling. *Transactions on Machine Learning Research*, 2023.

[12] Z. Hao, J. Yao, C. Su, H. Su, Z. Wang, F. Lu, Z. Xia, Y. Zhang, S. Liu, L. Lu, and J. Zhu. PINNacle: A comprehensive benchmark of physics-informed neural networks for solving pdes. *CoRR*, abs/2306.08827, 2023.

[13] S. M. S. Hassan, A. Feeney, A. Dhruv, J. Kim, Y. Suh, J. Ryu, Y. Won, and A. Chandramowlishwaran. Bubbleml: A multiphase multiphysics dataset and benchmarks for machine learning. In *Advances in Neural Information Processing Systems*, 2023.

[14] D. Hendrycks and K. Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[15] D. Holzmüller, V. Zaverkin, J. Kästner, and I. Steinwart. A framework and benchmark for deep batch active learning for regression. *J. Mach. Learn. Res.*, 24:164:1–164:81, 2023.

[16] J. M. Hyman and B. Nicolaenko. The kuramoto-sivashinsky equation: a bridge between pde's and dynamical systems. *Physica D: Nonlinear Phenomena*, 18(1-3):113–126, 1986.

[17] S. Janny, A. Béneteau, M. Nadri, J. Digne, N. Thome, and C. Wolf. EAGLE: large-scale learning of turbulent fluid dynamics with mesh transformers. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*, 2023.

[18] A. Kirsch, S. Farquhar, P. Atighehchian, A. Jesson, F. Branchaud-Charron, and Y. Gal. Stochastic batch acquisition: A simple baseline for deep active learning. *Transactions on Machine Learning Research*, 2023.

[19] S. Lanthaler, R. Molinaro, P. Hadorn, and S. Mishra. Nonlinear reconstruction for operator learning of pdes with discontinuities. In *The Eleventh International Conference on Learning Representations, ICLR*. OpenReview.net, 2023.

[20] S. Lanthaler, Z. Li, and A. M. Stuart. Nonlocality and nonlinearity implies universality in operator learning. *CoRR*, 2024.

[21] H. Lewy, K. Friedrichs, and R. Courant. Über die partiellen differenzengleichungen der mathematischen physik. *Mathematische annalen*, 100:32–74, 1928.

[22] S. Li, W. Xing, R. Kirby, and S. Zhe. Multi-fidelity bayesian optimization via deep neural networks. *Advances in Neural Information Processing Systems*, 33:8521–8531, 2020.

[23] S. Li, R. Kirby, and S. Zhe. Batch multi-fidelity bayesian optimization with deep auto-regressive networks. *Advances in Neural Information Processing Systems*, 34:25463–25475, 2021.

[24] S. Li, X. Yu, W. Xing, M. Kirby, A. Narayan, and S. Zhe. Multi-resolution active learning of fourier neural operators. *arXiv preprint arXiv:2309.16971*, 2023.

[25] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.

[26] P. Lippe, B. Veeling, P. Perdikaris, R. E. Turner, and J. Brandstetter. Pde-refiner: Achieving accurate long rollouts with neural PDE solvers. In *Advances in Neural Information Processing Systems*, 2023.

[27] T. Liu, J. A. L. Benitez, A. Khorashadizadeh, F. Faucher, M. V. de Hoop, and I. Dokmanić. Wavebench: Benchmarking data-driven solvers for linear wave propagation pdes. *Transactions on Machine Learning Research Journal*, 2024.

[28] Y. Luo, Y. Chen, and Z. Zhang. Cfdbench: A comprehensive benchmark for machine learning methods in fluid dynamics. *CoRR*, abs/2310.05963, 2023.

[29] Z. Mao and X. Meng. Physics-informed neural networks with residual/gradient-based adaptive sampling methods for solving partial differential equations with sharp solutions. *Applied Mathematics and Mechanics*, 44(7):1069–1084, 2023.

[30] V. V. Meduri, L. Popa, P. Sen, and M. Sarwat. A comprehensive benchmark framework for active learning methods in entity matching. In *Proceedings of the 2020 ACM SIGMOD international conference on management of data*, pages 1133–1147, 2020.

[31] A. Mehrjou, A. Soleymani, A. Jesson, P. Notin, Y. Gal, S. Bauer, and P. Schwab. Genedisco: A benchmark for experimental design in drug discovery. In *International Conference on Learning Representations*, 2021.

[32] S. C. Mouli, D. C. Maddix, S. Alizadeh, G. Gupta, A. Stuart, M. W. Mahoney, and Y. Wang. Using uncertainty quantification to characterize and improve out-of-domain learning for pdes. In *International Conference on Machine Learning, ICML*, volume abs/2403.10642 of *Proceedings of Machine Learning Research*. PMLR, 2024.

[33] M. Moustapha, S. Marelli, and B. Sudret. Active learning for structural reliability: Survey, general framework and benchmark. *Structural Safety*, 96:102174, 2022.

[34] O. Ovadia, E. Turkel, A. Kahana, and G. E. Karniadakis. Ditto: Diffusion-inspired temporal transformer operator. *CoRR*, abs/2307.09072, 2023.

[35] R. Pestourie, Y. Mroueh, T. V. Nguyen, et al. Active learning of deep surrogates for pdes: application to metasurface design. *Computational Materials*, 6(1):164, 2020.

[36] R. Pestourie, Y. Mroueh, C. V. Rackauckas, P. Das, and S. G. Johnson. Data-efficient training with physics-enhanced deep surrogates. In *AAAI 2022 Workshop on AI for Design and Manufacturing (ADAM)*, 2021.

[37] R. Pestourie, Y. Mroueh, C. Rackauckas, P. Das, and S. G. Johnson. Physics-enhanced deep surrogates for partial differential equations. *Nat. Mac. Intell.*, 5(12):1458–1465, 2023.

[38] E. Pickering, S. Guth, G. E. Karniadakis, and T. P. Sapsis. Discovering and forecasting extreme events via active learning in neural operators. *Nature Computational Science*, 2(12):823–833, 2022.

[39] R. Pinsler, J. Gordon, E. Nalisnick, and J. M. Hernández-Lobato. Bayesian batch active learning as sparse subset approximation. *Advances in neural information processing systems*, 32, 2019.

[40] A. F. Psaros, X. Meng, Z. Zou, L. Guo, and G. E. Karniadakis. Uncertainty quantification in scientific machine learning: Methods, metrics, and comparisons. *J. Comput. Phys.*, 477:111902, 2023.

[41] M. A. Rahman, Z. E. Ross, and K. Azizzadenesheli. U-NO: u-shaped neural operators. *Transactions on Machine Learning Research*, abs/2204.11127, 2022.

[42] L. Rauch, M. Aßenmacher, D. Huseljic, M. Wirth, B. Bischl, and B. Sick. Activeglae: A benchmark for deep active learning with transformers. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 55–74. Springer, 2023.

[43] S. Ren, Y. Deng, W. J. Padilla, L. M. Collins, and J. M. Malof. Deep active learning for scientific computing in the wild. *CoRR*, abs/2302.00098, 2023.

[44] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference*, pages 234–241. Springer, 2015.

[45] F. Sahli Costabal, Y. Yang, P. Perdikaris, D. E. Hurtado, and E. Kuhl. Physics-informed neural networks for cardiac activation mapping. *Frontiers in Physics*, 8:42, 2020.

[46] O. Sener and S. Savarese. Active learning for convolutional neural networks: A core-set approach. In *International Conference on Learning Representations*, 2018.

[47] H. S. Seung, M. Opper, and H. Sompolinsky. Query by committee. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 287–294, 1992.

[48] M. Takamoto, T. Praditia, R. Leiteritz, D. MacKinlay, F. Alesiani, D. Pflüger, and M. Niepert. Pdebench: An extensive benchmark for scientific machine learning. In *NeurIPS*, 2022.

[49] M. Takamoto, F. Alesiani, and M. Niepert. Learning neural PDE solvers with parameter-guided channel attention. In *International Conference on Machine Learning, ICML*, volume 202 of *Proceedings of Machine Learning Research*, pages 33448–33467. PMLR, 2023.

[50] A. Thakur. Uncertainty Quantification for Signal-to-Signal Regression-Based Neural Operator Frameworks. 7 2023.

[51] A. P. Toshev, G. Galletti, F. Fritz, S. Adami, and N. A. Adams. Lagrangebench: A lagrangian fluid mechanics benchmarking suite. In *Advances in Neural Information Processing Systems*, 2023.

[52] E. Tsymbalov, M. Panov, and A. Shapeev. Dropout-based active learning for regression. In *Analysis of Images, Social Networks and Texts: 7th International Conference, AIST 2018, Moscow, Russia, July 5–7, 2018, Revised Selected Papers 7*, pages 247–258. Springer, 2018.

[53] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[54] A. Wang, H. Liang, A. McDannald, I. Takeuchi, and A. G. Kusne. Benchmarking active learning strategies for materials optimization and discovery. *Oxford Open Materials Science*, 2 (1):itac006, 2022.

[55] T. Weber, E. Magnani, M. Pförtner, and P. Hennig. Uncertainty quantification for fourier neural operators. In *ICLR 2024 Workshop on AI4DifferentialEquations In Science*, 2024.

[56] C. Wu, M. Zhu, Q. Tan, Y. Kartha, and L. Lu. A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 403:115671, 2023.

[57] D. Wu, R. Niu, M. Chinazzi, Y. Ma, and R. Yu. Disentangled multi-fidelity deep bayesian active learning. In *International Conference on Machine Learning*, pages 37624–37634. PMLR, 2023.

[58] D. Wu, R. Niu, M. Chinazzi, A. Vespignani, Y.-A. Ma, and R. Yu. Deep bayesian active learning for accelerating stochastic simulation. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 2559–2569, 2023.

[59] T. Wu, W. Neiswanger, H. Zheng, S. Ermon, and J. Leskovec. Uncertainty quantification for forward and inverse problems of pdes via latent global evolution. In *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI*, pages 320–328. AAAI Press, 2024.

[60] Y. Wu and K. He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.

[61] Y. Yang and M. Loog. A benchmark and comparison of active learning for logistic regression. *Pattern Recognition*, 83:401–415, 2018.

[62] X. Zhan, H. Liu, Q. Li, and A. B. Chan. A comparative survey: Benchmarking for pool-based active learning. In *IJCAI*, pages 4679–4686, 2021.

[63] X. Zhang, J. Helwig, Y. Lin, Y. Xie, C. Fu, S. Wojtowytsch, and S. Ji. Sinenet: Learning temporal dynamics in time-dependent partial differential equations. *CoRR*, abs/2403.19507, 2024.

ACTIVE LEARNING FOR NEURAL PDE SOLVERS
SUPPLEMENTAL MATERIAL

A Additional Background on Related Work

In this section, we elaborate on related works that tackle active learning in relevant settings and problems discussed here. Moreover, we summarize related work on uncertainty quantification and SciML benchmarks closely related to the proposed AL4PDE benchmark.

A.1 General Active Learning

Most AL algorithms are evaluated on classic image classification datasets [3, 4] and many benchmarks also consider the more common classification setting [42, 61, 62]. There is also work on specialized tasks such as entity matching [30], structural integrity [33], material science [54], or drug discovery [31]. Holzmüller et al. [15] present a benchmark for AL of single-output, tabular regression tasks. Wu et al. [56] study different adaptive and non-adaptive methods for selecting collocation points for PINNs. Ren et al. [43] benchmark pool-based AL methods on simulated, mostly tabular regression tasks.

In terms of deep active learning methods for regression, there are multiple approaches: Query-by-committee [47] uses ensemble prediction variances as uncertainties. Tsymbalov et al. [52] use Monte Carlo dropout to obtain uncertainties; however, their method is only applicable by training with dropout. Approaches based on last-layer Bayesian linear regression [39, 3] are often convenient since they do not require ensembles or dropout. These methods are applicable in principle in our setting but lose their original Bayesian interpretation since the last layer of a neural operator is applied multiple times during the autoregressive rollout. Distance-based methods like Core-Set [46, 10] and the clustering-based LCMD [15] exhibit better runtime complexity than last-layer Bayesian methods while sharing their other advantages [15]. Since these algorithms just require some distance function between two input points, we can adapt them to the neural PDE solver setting.

Physics-Informed Neural Networks and Neural Operators. In the context of neural PDE solvers, AL has primarily been applied to select the so-called collocation points of PINNs. A typical approach here would be to sample these collocation points based on the residual error directly [2, 9, 29, 56]. While this strategy can be effective, it differs from standard AL since it uses the "label", i.e., the residual loss, when selecting data points. Aikawa et al. [1] use a Bayesian PINN to select points based on uncertainty, whereas Sahli Costabal et al. [45] employ a PINN ensemble for AL of cardiac activation mapping. Pestourie et al. [35] use AL to approximate Maxwell equations using ensemble-based uncertainty quantification for metamaterial design. Uncertainty-based AL was also employed for diffusion, reaction-diffusion, and electromagnetic scattering [37].

A.2 Uncertainty Quantification (UQ)

Uncertainty quantification has been studied in the context of SciML simulations. Psaros et al. [40] provide a detailed overview of UQ methods in SciML, specifically for PINNs and DeepONets. However, effective and reliable UQ methods for neural operators (i.e., mapping between function spaces) and high dimensionality of data, which is common in PDE solving, remain challenging.

Neural Operators. LE-PDE-UQ [59] deals with a method to estimate the uncertainty of neural operators by modeling the dynamics in the latent space. The model has been shown to outperform other UQ approaches, such as Bayes layer, Dropout, and L2 regularization on Navier-Stokes turbulent flow prediction tasks. Unlike the considered setting in our case, the model utilizes a history of 10 timesteps and has been tested only on a fixed PDE parameter. Hence, it is unclear whether the robustness of this approach remains when these settings change.

Mouli et al. [32] aim to develop a cost-efficient method for uncertainty quantification of parametric PDEs, specifically one that works well in the out-of-domain test settings of PDE parameters. First, the study shows the challenges of existing UQ methods, such as the Bayesian neural operator (BayesianNO) for out-of-domain test data. It then shows that ensembling several neural operators is an effective strategy for UQ that is well-correlated with prediction errors and proposes diverse neural operators (DiverseNO) as a cost-effective way to estimate uncertainty with just a single model based on FNO outputting multiple predictions.

Thakur [50] studies UQ in the context of neural operators and develops a probabilistic FNO model to quantify aleatoric and epistemic uncertainties. [55] study UQ for FNO and propose a Laplace approximation for the Fourier layer to effectively compute uncertainty.

A.3 Further Scientific Machine Learning Benchmarks

In recent years, various benchmarks and datasets for SciML have been published. We outline some of the major open-source benchmarks below.

PDEBench [48] is a large-scale SciML benchmark of 1D to 3D PDE equations modeling hydrodynamics ranging from Burgers' to compressible and incompressible Navier-Stokes equations. PDEArena [11] is a modern surrogate modeling benchmark including PDEs such as incompressible Navier-Stokes, Shallow Water, and Maxwell equations [7]. CFDBench [28] is a recent benchmark comprising four flow problems, each with three different *operating parameters*, the specific instantiations of which include varying boundary conditions, physical properties, and geometry of the fluid. The benchmark compares the generalization capabilities of a range of neural operators and autoregressive models for each of the said *operating parameters*. LagrangeBench [51] is a large-scale benchmark suite for modeling 2D and 3D fluid mechanics problems based on the Lagrangian specification of the flow field. The benchmark provides both datasets and baseline models. For the former, it introduces seven datasets of varying Reynolds numbers by solving a weak form of NS equations using smoothed particle hydrodynamics. For the latter, efficient JAX implementations of GNN baseline models such as Graph Network-based Simulator and (Steerable) Equivariant GNN are included. EAGLE [17] introduces an industrial-grade dataset of non-steady fluid mechanics simulations encompassing 600 geometries and 1.1 million 2D meshes. In addition, to effectively process a dataset of this scale, the benchmark proposes an efficient multi-scale attention model, mesh transformer, to capture long-range dependencies in the simulation. BubbleML [13] is a thermal simulations dataset comprising boiling scenarios that exhibit multiphase and multiphysics phase change phenomena. It also consists of a benchmark validating the dataset against U-Nets and several variants of FNO.

B Additional Task Details

In the following section, we will discuss the tasks considered in detail. Table 1 shows the temporal and spatial resolution of the considered PDEs. The test data consists of 2048 trajectories generated with inputs from p_T.

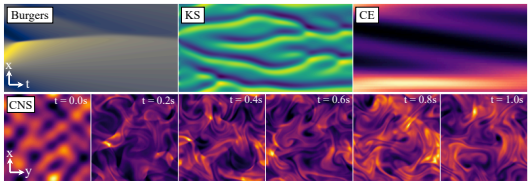

Figure 4: Example trajectories of the PDEs.

PDE	T in s	Sim. Res. (N_t, $N_\mathbf{x}$, [$N_\mathbf{y}$])	Train. Res. (N_t, $N_\mathbf{x}$, [$N_\mathbf{y}$])
Burgers	2	(201, 1024)	(41, 256)
KS	40	(801, 512)	(41, 256)
CE	4	(501, 64)	(51, 64)
CNS	1	(21, 128, 128)	(21, 64, 64)

Table 1: Discretizations of the PDEs.

B.1 Burgers' Equation

The 1D Burgers' equation is written as

$$\partial_t u + u \partial_x u = (\nu/\pi) \partial_{xx} u. \tag{4}$$

The spatial domain is set to $x \in [0, 1]$. Following the parameter spacing of the PDE parameters values in PDEBench [48] and CAPE [49], we draw them on a logarithmic scale, i.e., we first draw $\boldsymbol{\lambda}_{i,\text{normed}}$ uniformly from $[0, 1)$ and then transform the parameter to its domain $[a_i, b_i)$ using

$$\boldsymbol{\lambda}_i = a_i \exp(\log(b_i/a_i)\, \boldsymbol{\lambda}_{i,\text{normed}}). \tag{5}$$

The parameter space is set to $\nu \in [0.001, 1)$. We use the FDM-based JAX simulator and the initial condition generator from PDEBench [48]. The ICs are constructed based on a superposition of

sinusoidal waves [48],

$$\mathbf{u}^0(x) = \sum_{i=1}^{N_w} A_i \sin(2\pi k_i x/L + \phi_i) \tag{6}$$

where the wave number k_i is an integer sampled uniformly from $[1, 5)$, amplitude A_i is sampled uniformly from $[0, 1)$, and phase ϕ_i from $[0, 2\pi)$. The number of waves N_w is set to 2. Windowing is applied afterward with a probability of 10%, where all parts of the IC are set to zero outside of $[x_L, x_R]$. x_L is drawn uniformly from $[0.1, 0.45)$ and x_R from $[0.55, 0.9)$. Lastly, the sign of $\mathbf{u}^0$ is flipped for all entries with a probability of 10%.

B.2 Kuramoto-Sivashinsky (KS)

The 1D KS equation reads as

$$\partial_t u + u\partial_x u + \partial_{xx} u + \nu \partial_{xxxx} u = 0 \quad x \in [0, L]. \tag{7}$$

The ICs are generated using the superposition of sinusoidal waves (Eq. (6)), but k_i is sampled from $[1, 10)$, A_i from $[-1, 1)$ and ϕ_i from $[0, 2\pi)$. No windowing or sign flips are applied. The total number of waves N_w in this case is set to 10. Since we cannot omit the first part of the simulations as Lippe et al. [26], we reduce the simulation time to 40s, but allow for more variance in the ICs to reach the chaotic behavior easier by increasing the number of wave functions of the IC. The trajectories are obtained using JAX-CFD [8]. The PDE parameters are drawn uniformly from their range (no logarithmic scale). The domain length L is chosen from $L \in [0.1, 100)$ and the viscosity from $\nu \in [0.5, 4)$.

B.3 Combined Equation (CE)

We adopt the *combined equation* albeit without the *forcing* term and the corresponding numerical solver from Brandstetter et al. [6].

$$\partial_t u + \partial_x \left(\alpha u^2 - \beta \partial_x u + \gamma \partial_{xx} u \right) = 0 \tag{8}$$

As for the IC, the domain of k_i is set to $[1, 3)$ and for A_i it is set as $[-0.4, 0.4)$. The number of waves N_w is set to 5, and no windowing or sign flips are applied either. The PDE parameters are also drawn uniformly from their range. The parameter space is defined to be $\alpha \in [0, 3)$, $\beta \in [0, 0.4)$ and $\gamma \in [0, 1)$. Depending on the choice of the PDE coefficients (α, β, γ), this equation recovers the Heat (0, 1, 0), Burgers (0.5, 1, 0), or the Korteweg-de-Vries (3, 0, 1) PDE. The spatial domain is set to $x \in [0, 16]$.

B.4 Compressible Navier-Stokes (CNS)

The 2D CNS equations from PDEBench [48] are written as

$$\partial_t \rho + \nabla \cdot (\rho \mathbf{v}) = 0, \tag{9a}$$

$$\rho(\partial_t \mathbf{v} + \mathbf{v} \cdot \nabla \mathbf{v}) = -\nabla p + \eta \triangle \mathbf{v} + (\zeta + \eta/3) \nabla(\nabla \cdot \mathbf{v}), \tag{9b}$$

$$\partial_t(\epsilon + \rho v^2/2) + \nabla \cdot [(p + \epsilon + \rho v^2/2)\mathbf{v} - \mathbf{v} \cdot \sigma'] = \mathbf{0}, \tag{9c}$$

where σ' is the viscous tensor. The equation has four channels (density ρ, velocity x-component v_x and y-component v_y as well as the pressure p. The spatial domain is set to $\mathbf{x} \in [0, 1] \times [0, 1]$ We use the JAX simulator and IC generator from PDEBench [48] for CNS equations. The PDE parameters are drawn in logarithmic scale as in Eq. (5) with $\eta, \zeta \in [10^{-4}, 10^{-1})$. The IC generator for the pressure, density, and velocity channels is also based on the superposition of sinusoidal functions. However, the velocity channels are renormalized so that the IC has a given input Mach number, which is drawn from $m \in [0.1, 1)$. Secondly, we constrain the density channel to be positive by

$$\mathbf{u}_\rho = \rho_0 (1 + \Delta_\rho \, \mathbf{u}'_\rho \, / \max_x (|\, \mathbf{u}'_\rho(x)|) \tag{10}$$

where ρ_0 is sampled from $[0.1, 10)$ and Δ_ρ from $[0.013, 0.26)$. The pressure channel p is similarly transformed using $\Delta_p \in [0.04, 0.8)$. The offset p_0 is defined relatively to ρ_0 as $p_0 = T_0 \rho_0$ with $T_0 \in [0.1, 10)$. The compressibility is reduced using a Helmholtz-decomposition [48]. A windowing is applied with a probability of 50% to a channel.

C Additional Model and Training Details

This section describes the baseline surrogate models used in more detail, lists the hyperparameters, and explains various training methods. First, we provide a short description of the base models used. Then, we explain the training methods and list the hyperparameters.

C.1 Fourier Neural Operators (FNOs)

We use the FNO [25] implementation provided by PDEBench [48]. FNOs are based on spectral convolutions, where the layer input is transformed using a Fast Fourier Transformation (FFT), multiplied in the Fourier space with a weight matrix, and then transformed back using an inverse FFT. Following the recent observations made in [19, 20] that only a small fixed number of modes are sufficient to achieve the needed expressivity of FNO, we retain only a limited number of low-frequency Fourier modes and discard the ones with higher frequencies. The raw PDE parameter values are appended as additional constant channels to the model input [49].

C.2 U-shaped Networks (U-Nets)

U-Net [44] is a common architecture in computer vision, particularly for perception and semantic segmentation tasks. The structure resembles an hourglass, where the inputs are first successively downsampled at multiple levels and then gradually, with the same number of levels, upsampled back to the original input resolution. This structure allows the model to capture and process spatial information at multiple scales and resolutions. The U-Net used in this paper is based on the modern U-Net version of Gupta and Brandstetter [11], which differs from the original U-Net [44] by including improvements such as group normalization [60]. The model is conditioned on the input PDE parameter values, where they are transformed into vectors using a learnable Fourier embedding [53] and a projection layer and are then added to the convolutional layers' inputs in the *up* and *down* blocks.

C.3 SineNet

U-Nets were originally designed for semantic segmentation problems in medical images [44]. Due to its intrinsic capabilities for multi-scale representation modeling, U-Nets have been widely adopted by the SciML community for PDE solving [48, 11, 26, 41, 34]. One of the important components of U-Nets to recover high-resolution details in the upsampling path is by the fusion of feature maps using skip connections. This does not cause an issue for semantic segmentation tasks since the desired output for a given image is a segmentation mask. However, in the context of time-dependent PDE solving, specifically for advection-type PDEs modeling transport phenomena, this is not well-suited since there will be a "lag" in the feature maps of the downsampling path since the upsampling path is expected to predict the solution $\mathbf{u}$ for the next timestep. This detail was overlooked in U-Net adaptations for time-dependent PDE solving. SineNet is a recently introduced image-to-image model that aims to mitigate this problem by stacking several U-Nets, called *waves*, drastically reducing the feature misalignments. More formally, SineNet learns the mapping

$$\boldsymbol{x}_t = P(\{\boldsymbol{u}_{t-h+1}, \ldots, \boldsymbol{u}_t\})$$

$$\boldsymbol{u}_{t+1} = Q(\boldsymbol{x}_{t+1})$$

$$\boldsymbol{x}_{t+\Delta_k} = V_k(\boldsymbol{x}_{t-\Delta_{k-1}}), \qquad k = 1, \ldots, K$$

Unlike the original SineNet, our adaptation uses only one temporal step as a context to predict the solution for the subsequent timestep.

C.4 Hyperparameters and Training Protocols

During AL, we use m=1 for power sampling and a prediction batch size for the pool of 200. The features of all inputs are projected using the sketch operator to a dimension of 512. Table 2 lists the model hyperparameters.

U-Net	
Activation	GELU [14]
Conditioning	Fourier [53]
Channel multiplier	[1, 2, 2, 4]
Hidden Channels	16
# Params (1D)	3,378,865
# Params (2D)	9,182,036
FNO	
Activation	GELU [14]
Conditioning	As additional channel [49]
Layers	4
Width (1D)	64
Width (2D)	32
Modes	20
# Params (1D)	353,154
# Params (2D)	3,286,310
SineNet	
Activation	GELU [14]
Conditioning	Fourier [53]
Hidden Channels	32
Waves	4
# Params (2D)	5,020,840

Table 2: Model hyperparameters.

The inputs are channel-wise normalized using the standard deviation of the different channels on the initial data set. The outputs are denormalized accordingly. The input only consists of the current state $\mathbf{u}_t$, not including data from prior timesteps. All models are used to predict the difference to the current timestep (for U-Net, the outputs are multiplied with a fixed factor of 0.3 following Lippe et al. [26].

The neural network parameters θ are minimized using the root mean squared error (RMSE) on the training samples,

$$\mathcal{L}_{\text{RMSE}}(\mathbf{u}, \hat{\mathbf{u}}) = \sqrt{\frac{1}{N_t N_{\mathbf{x}} N_c} \sum_{i=1}^{N_t} \sum_{j=1}^{N_{\mathbf{x}}} \|\mathbf{u}(t_i, \mathbf{x}_j) - \hat{\mathbf{u}}(t_i, \mathbf{x}_j)\|_2^2}. \tag{11}$$

We employ one- and two-step training strategies during the training phase and a complete rollout of the trajectories during validation. The training is performed with a cosine schedule, which reduces the learning rate from 10^{-3} to 10^{-5}. The batch size is set to 512 (CNS: 64). For the FNO model in the 2D experiment, we found it better to use the teacher-forcing schedule from [49]. We found it necessary to add gradient clipping to prevent a sudden divergence in the training curve. To account for the very different gradient norms among problems, we set the upper limit to 5 times the highest gradient found in the first five epochs. Afterward, the limit is adapted using a moving average.

C.5 Hardware and Runtime

The experiments were performed on NVIDIA GeForce RTX 4090 GPUs (one per experiment). Table 3 shows the runtime and GPU memory during training.

D Framework Overview

The framework has three major components: `Model`, `BatchSelection`, and `Task`. `Task` acts as a container of all the PDE-specific information and contains the `Simulator`, `PDEParamGenerator`, and `ICGenerator` classes. `PDEParamGenerator` and `ICGenerator` can draw samples from the test

	Burgers	KS	CE	CNS
Runtime in h				
Random	15.1	12.9	16.8	37.9
SBAL	22.9	20.1	25.6	54.4
LCMD	14.5	13.9	17.2	39.0
Core-Set	14.5	13.4	16.6	39.7
Top-K	22.1	29.8	26.4	55.5
Training Memory in GB				
All	8.16	8.18	4.47	7.29

Table 3: Total runtime of the different AL methods and the memory during training (since all methods train the same model, the memory usage during training is identical).

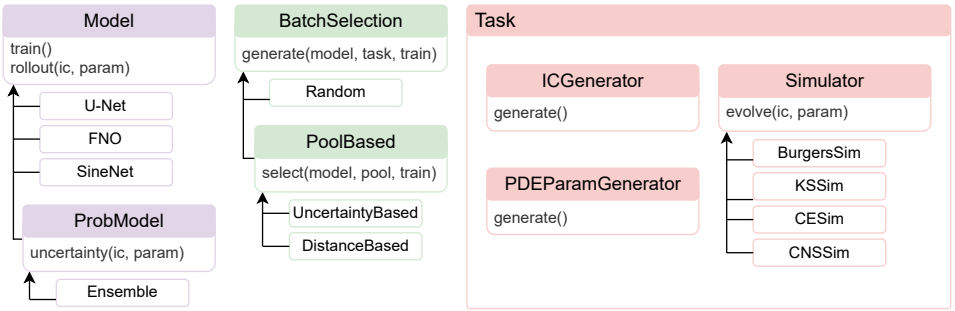

Figure 5: Structural overview of the AL4PDE benchmark.

input distribution p_T. The inputs are first drawn from a normalized range and then transformed into the actual inputs. Afterward, the inputs can be passed to the simulator to be evolved into a trajectory. Listing 1 shows the pseudocode of the (random) data generation pipeline. In order to implement a new PDE, a user has to implement a new subclass of `Simulator` overwrite the `__call__` function and, if desired, add a new `ICGenerator`.

Listing 2 shows the interface for the `Model` and `ProbModel` classes. `Model` provides functions to rollout a surrogate and deals with the training and evaluation. In order to add a new surrogate, a user has to overwrite the `forward` method. The `rollout` function also allows to get the internal model features for distance-based acquisition functions. `ProbModel` is an extension of the `Model` class, which adds the possibility of getting an uncertainty estimate. After training the model, the `BatchSelection` class is called in order to select a new set of inputs. The most important subclass is the `PoolBased` class, which deals with managing the pool and provides the `select_next` method, which a new pool-based method has to overwrite.

The code is available on GitHub at https://github.com/dmusekamp/al4pde.

```python
class PDEParamGenerator:

    def get_normed_pde_params(self, n):
        # Generates the random PDE parameters in a normed space
        # (e.g. between 0 and 1).

    def get_pde_params(self, pde_params_normed):
        # Transforms the normed parameters to their true value.


class ICGenerator:

    def initialize_ic_params(self, n):
        # Generates the random parameters of an IC (e.g. Mach number).

    def generate_initial_conditions(self, ic_params, pde_params)
        # Transforms the IC parameters and PDE parameters to the IC.


class Simulator:

    def __call__(self, ic, pde_params, grid):
        # Evolves the IC for a given PDE parameter.


# generate pde parameters
pde_params_normed = pde_gen.get_normed_pde_params(n)
pde_params = pde_gen.get_pde_params(pde_params_normed)

# generate ICs
ic_params = ic_gen.initialize_ic_params(n)
ic_gen.generate_initial_conditions(ic_params, pde_params)

trajectories = sim(ic, pde_param, grid)
```

Listing 1: Interface and example code for generating inputs and simulation.

```python
class Model(nn.Module):

    def init_training(self, al_iter):
        # Reset model, optimizer, scheduler, ...

    def forward(self, xx, grid, param, return_features):
        # Predic next state.

    def rollout(self, xx, grid, final_step, param,  return_features):
        # Autoregressive rollout of the model until timestep final_step.

    def evaluate(self, step, loader, prefix):
        # Evaluate the model on the given dataset (e.g. validation, train).

    def train_single_epoch(self, current_epoch, total_epoch, num_epoch):
        # Train the model for one epoch.

    def train_n_epoch(self, al_iter, num_epoch):
        # Train the model .


class ProbModel(Model):

    def uncertainty(self, xx, grid, param):
        # Get uncertainty over next state.

    def unc_roll_out(self, xx, grid, final_step, param, return_features):
        # Compute prediction and uncertainty of the rollout.


class BatchSelection:

    def generate(self, prob_model, al_iter, train_loader):
        # Selects new inputs and passes them to the simulator.


class PoolBased(BatchSelection):

    def select_next(self, step, prob_model, ic_pool, pde_param_pool,
        ic_train, pde_param_train, grid, al_iter):
        # Selects new input from (ic_pool, pde_param_pool).


for al_iter in range(num_al_iter):
    # retrain model
    prob_model.train_n_epoch(al_iter, num_epoch)

    # select next inputs
    batch_sel.generate(prob_model, al_iter, train_loader)
```

Listing 2: Interface and example code for the neural operator models and AL methods.

E Additional Experiments

In this section, we provide additional experiments and ablation studies.

E.1 Timing Experiment

The main experiments only provide the error over the number of data points since we use problems with rather fast solvers to accelerate the benchmarking of the AL methods. Additionally, a more lightweight model, trained for a shorter time, might be enough for data selection even if it does not reach the best possible accuracy. To investigate AL in terms of time efficiency gains, we perform one experiment on the Burgers' PDE, for which the numerical solver is the most expensive among all 1D PDEs due to its higher resolution.

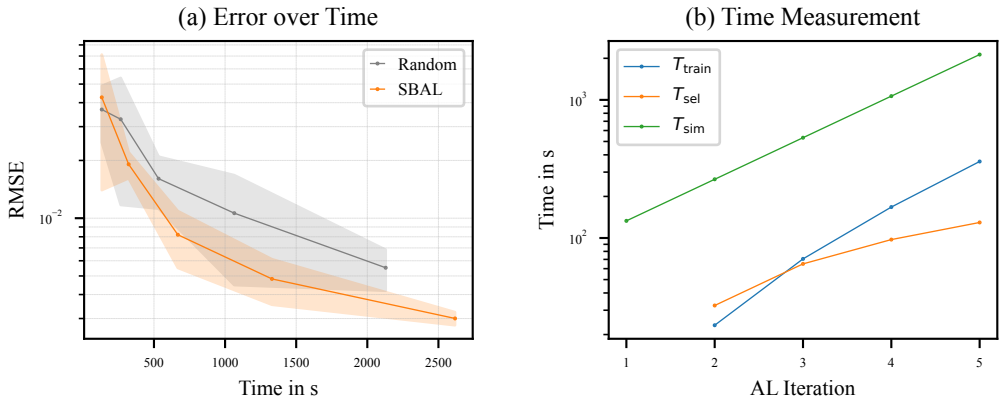

Figure 6: (a) Error of the standard U-Net on Burgers over the required total time. Using smaller FNOs to select the data, SBAL can provide smaller errors in the same amount of time. (b) Cumulative training, selection, and simulation times necessary to reach the given active learning iteration (e.g., time to select data for iteration 2 counted in iteration 2) for the timing experiment.

A realistic time measurement for the simulator of Burgers' equation is challenging. Firstly, we observed that we can reach the shortest time per trajectory by setting the batch size to 4096 (0.52 seconds). Therefore, we use this as the fixed time per trajectory. The actual simulation times per AL iteration are higher since we start with batch sizes below this saturation point. Secondly, the simulation step size is adapted to the PDE parameter value due to the CFL condition [21]. Therefore, it would be beneficial to batch similar parameter values together and also to consider the parameter simulation costs in the acquisition function. Fig. 6b) shows training, selection, and simulation times.

The FNO surrogate used for selection is only trained for 20 epochs with a batch size of 1024. We use one-step training, and the learning rate of 0.001 is not annealed. The model itself has a width of 20 and uses 20 modes, resulting in 36,706 parameters. During selection, a batch size of 32,768 is used.

We train a regular U-Net on the AL collected data, which allows us to use a small, lightweight model for data selection only and an expensive one to evaluate the data selected. Fig. 6a) shows the accuracy of the evaluation U-Net over the cumulative time consumed for training the selection model, selecting the inputs, and simulation. For the random baseline, only the simulation time is considered. On Burgers, AL provides better accuracy for the same time budget.

E.2 Different Error Function

It is important to consider error metrics for surrogate model training besides the RMSE [48]. Thus, we explore the impact of AL on the mean absolute error (MAE) as an example of an alternative metric. As depicted in Fig. 7, SBAL, when using the absolute difference between the models as the uncertainty, can also successfully reduce the MAE. However, the MAE does not improve greatly relative to random sampling when the standard variance between the models is used. Hence, it is crucial to tailor the AL method to the relevant metric.

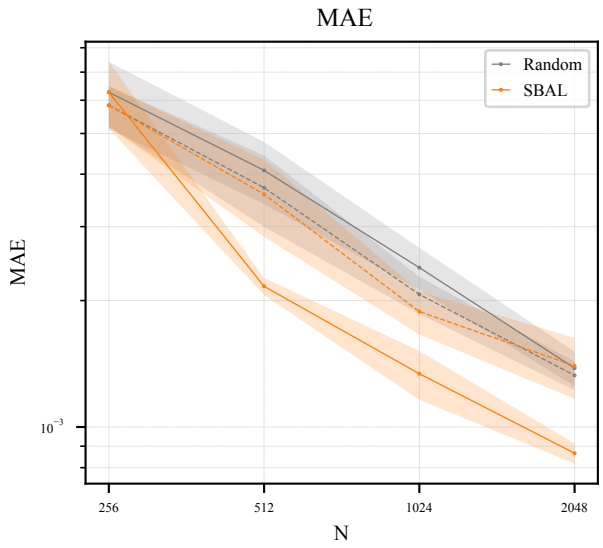

Figure 7: AL with the MAE as the objective on Burgers, compared to the MAE of the same setup trained with the RMSE (dashed). Considering the desired error metric in the uncertainty estimate and training loss is essential.

E.3 Additional Ablations

We ablate different design choices for the considered AL algorithms. For the SBAL algorithm, we investigate the ensemble size (Fig. 8a) next to the choice of the base model architecture in Fig. 3. Consistent with prior work [38], choosing an ensemble size of two models is already sufficient (Fig. 8a). In general, the average uncertainty and error of a trajectory with two ensemble members are correlated with a Pearson coefficient of 0.41 on CE in the worst case up to 0.94 on CNS (Table 8). Fig. 8b) compares different feature choices for the LCMD algorithm, which are used to calculate the

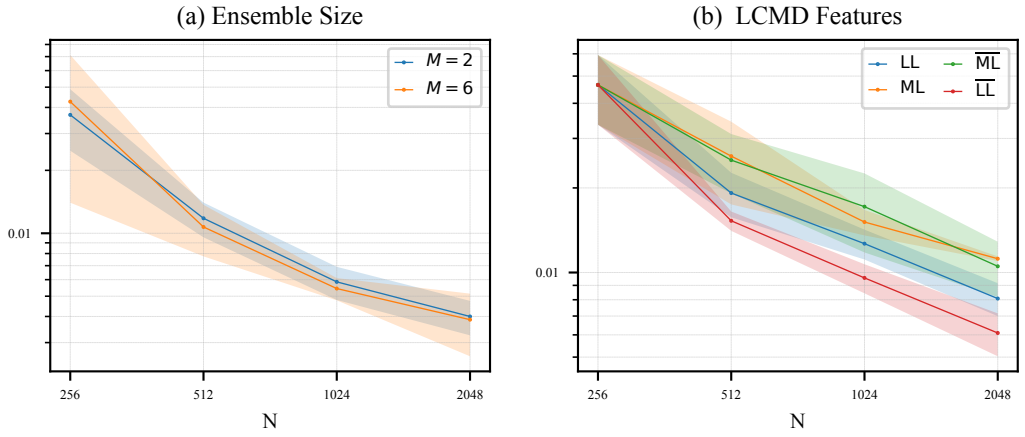

Figure 8: (a) Number of models M in the ensemble on Burgers. SBAL works reliably with only two models. (b) Comparison of different feature vectors LCMD on CE. Shown are the last layer feature map (LL), its spatial average ($\overline{\text{LL}}$), as well as the features of the mid layer (ML) and its spatial average ($\overline{\text{ML}}$). Averaging the feature maps improves the error, indicating the importance of considering the model invariances.

distances. Using the spatial average of the last layer features produces higher accuracy than using the full feature vector or the features from the bottleneck step in the middle of the U-Net. Thus, it

is indeed important for distance-based selection to consider the equivariances of the problem in the distance function.

F Detailed Results

Tables 4, 5, 6 and 7 list the results from the main experiments. Table 8 shows the Pearson and Spearman coefficient of the average uncertainty per trajectory with the average error per trajectory. Among the PDEs, the Pearson correlation coefficient is the lowest on CE. The Spearman coefficient, which measures the correlation in terms of the ranking, is above 0.73 on average for all experiments.

Iteration	1	2	3	4	5
			RMSE $\times 10^{-2}$		
Random	3.684 ± 1.203	3.278 ± 2.107	1.607 ± 0.485	1.062 ± 0.614	0.552 ± 0.133
SBAL	3.684 ± 1.203	1.179 ± 0.223	0.586 ± 0.106	0.400 ± 0.075	$\mathbf{0.259 \pm 0.028}$
LCMD	3.684 ± 1.203	$\mathbf{0.808 \pm 0.053}$	$\mathbf{0.521 \pm 0.052}$	$\mathbf{0.394 \pm 0.043}$	0.269 ± 0.014
Core-Set	3.684 ± 1.203	1.021 ± 0.160	0.659 ± 0.100	0.476 ± 0.134	0.292 ± 0.015
Top-K	3.684 ± 1.203	1.494 ± 0.250	0.964 ± 0.258	0.477 ± 0.044	0.360 ± 0.096
			50% Quantile $\times 10^{-2}$		
Random	0.182 ± 0.015	$\mathbf{0.122 \pm 0.015}$	$\mathbf{0.083 \pm 0.010}$	$\mathbf{0.058 \pm 0.005}$	$\mathbf{0.044 \pm 0.007}$
SBAL	0.182 ± 0.015	0.178 ± 0.032	0.105 ± 0.011	0.078 ± 0.011	0.054 ± 0.006
LCMD	0.182 ± 0.015	0.129 ± 0.014	0.101 ± 0.015	0.068 ± 0.008	0.050 ± 0.006
Core-Set	0.182 ± 0.015	0.169 ± 0.017	0.133 ± 0.013	0.094 ± 0.014	0.063 ± 0.008
Top-K	0.182 ± 0.015	0.197 ± 0.020	0.176 ± 0.024	0.109 ± 0.010	0.078 ± 0.012
			95% Quantile $\times 10^{-2}$		
Random	1.468 ± 0.136	0.834 ± 0.125	$\mathbf{0.502 \pm 0.037}$	$\mathbf{0.343 \pm 0.014}$	$\mathbf{0.255 \pm 0.025}$
SBAL	1.468 ± 0.136	1.054 ± 0.248	0.544 ± 0.065	0.409 ± 0.064	0.269 ± 0.026
LCMD	1.468 ± 0.136	$\mathbf{0.669 \pm 0.069}$	0.503 ± 0.091	0.347 ± 0.030	0.259 ± 0.020
Core-Set	1.468 ± 0.136	0.865 ± 0.123	0.662 ± 0.090	0.503 ± 0.113	0.336 ± 0.034
Top-K	1.468 ± 0.136	1.273 ± 0.177	1.045 ± 0.200	0.575 ± 0.064	0.449 ± 0.077
			99% Quantile $\times 10^{-2}$		
Random	6.315 ± 0.838	3.327 ± 0.724	1.653 ± 0.111	0.968 ± 0.046	0.649 ± 0.027
SBAL	6.315 ± 0.838	3.169 ± 0.945	1.360 ± 0.213	0.987 ± 0.239	0.599 ± 0.056
LCMD	6.315 ± 0.838	$\mathbf{1.802 \pm 0.157}$	$\mathbf{1.223 \pm 0.237}$	$\mathbf{0.819 \pm 0.108}$	$\mathbf{0.573 \pm 0.041}$
Core-Set	6.315 ± 0.838	2.461 ± 0.500	1.756 ± 0.360	1.153 ± 0.295	0.703 ± 0.056
Top-K	6.315 ± 0.838	4.456 ± 1.685	3.251 ± 1.039	1.347 ± 0.129	1.048 ± 0.326

Table 4: Error metrics on the Burgers' equation.

Iteration	1	2	3	4	5
			RMSE		
Random	0.452 ± 0.026	$\underline{0.370 \pm 0.012}$	0.312 ± 0.013	0.272 ± 0.010	0.229 ± 0.010
SBAL	0.452 ± 0.026	$\mathbf{0.347 \pm 0.020}$	$\mathbf{0.281 \pm 0.010}$	$\mathbf{0.236 \pm 0.008}$	$\mathbf{0.200 \pm 0.012}$
LCMD	0.452 ± 0.026	0.370 ± 0.009	0.315 ± 0.013	0.266 ± 0.019	$\underline{0.219 \pm 0.018}$
Core-Set	0.452 ± 0.026	0.389 ± 0.011	0.335 ± 0.013	0.278 ± 0.006	0.235 ± 0.020
Top-K	0.452 ± 0.026	0.378 ± 0.018	$\underline{0.305 \pm 0.011}$	$\underline{0.264 \pm 0.014}$	0.225 ± 0.015
			50% Quantile		
Random	0.021 ± 0.005	$\mathbf{0.011 \pm 0.002}$	$\mathbf{0.008 \pm 0.001}$	$\mathbf{0.005 \pm 0.001}$	$\mathbf{0.003 \pm 0.001}$
SBAL	0.021 ± 0.005	$\underline{0.016 \pm 0.004}$	$\underline{0.013 \pm 0.003}$	$\underline{0.008 \pm 0.001}$	0.006 ± 0.001
LCMD	0.021 ± 0.005	0.020 ± 0.003	0.016 ± 0.003	0.009 ± 0.003	$\underline{0.006 \pm 0.001}$
Core-Set	0.021 ± 0.005	0.022 ± 0.003	0.021 ± 0.002	0.014 ± 0.002	0.009 ± 0.002
Top-K	0.021 ± 0.005	0.020 ± 0.003	0.018 ± 0.002	0.012 ± 0.003	0.010 ± 0.002
			95% Quantile		
Random	0.603 ± 0.106	$\mathbf{0.363 \pm 0.020}$	$\mathbf{0.231 \pm 0.024}$	$\mathbf{0.143 \pm 0.011}$	$\mathbf{0.094 \pm 0.006}$
SBAL	0.603 ± 0.106	$\underline{0.376 \pm 0.060}$	$\underline{0.255 \pm 0.031}$	$\underline{0.163 \pm 0.022}$	$\underline{0.119 \pm 0.018}$
LCMD	0.603 ± 0.106	0.458 ± 0.024	0.344 ± 0.024	0.230 ± 0.035	0.140 ± 0.023
Core-Set	0.603 ± 0.106	0.501 ± 0.025	0.425 ± 0.034	0.295 ± 0.021	0.213 ± 0.053
Top-K	0.603 ± 0.106	0.458 ± 0.017	0.340 ± 0.026	0.257 ± 0.039	0.188 ± 0.016
			99% Quantile		
Random	2.368 ± 0.153	1.844 ± 0.105	1.382 ± 0.117	$\underline{1.040 \pm 0.092}$	$\underline{0.708 \pm 0.048}$
SBAL	2.368 ± 0.153	$\mathbf{1.655 \pm 0.137}$	$\mathbf{1.177 \pm 0.100}$	$\mathbf{0.844 \pm 0.103}$	$\mathbf{0.619 \pm 0.093}$
LCMD	2.368 ± 0.153	$\underline{1.811 \pm 0.056}$	1.440 ± 0.097	1.151 ± 0.123	0.802 ± 0.149
Core-Set	2.368 ± 0.153	1.920 ± 0.077	1.571 ± 0.090	1.230 ± 0.046	0.982 ± 0.202
Top-K	2.368 ± 0.153	1.860 ± 0.126	$\underline{1.356 \pm 0.092}$	1.138 ± 0.086	0.873 ± 0.119

Table 5: Error metrics on KS.

Iteration	1	2	3	4	5
			RMSE $\times 10^{-2}$		
Random	4.651 ± 1.293	3.814 ± 1.121	2.609 ± 0.466	1.630 ± 0.257	1.108 ± 0.117
SBAL	4.651 ± 1.293	1.597 ± 0.083	0.931 ± 0.125	$\mathbf{0.496 \pm 0.087}$	$\mathbf{0.318 \pm 0.048}$
LCMD	4.651 ± 1.293	$\mathbf{1.528 \pm 0.121}$	0.957 ± 0.114	0.609 ± 0.107	0.338 ± 0.041
Core-Set	4.651 ± 1.293	1.596 ± 0.235	1.033 ± 0.076	0.761 ± 0.230	0.424 ± 0.053
Top-K	4.651 ± 1.293	1.678 ± 0.099	$\mathbf{0.904 \pm 0.101}$	0.529 ± 0.103	0.373 ± 0.077
			50% Quantile $\times 10^{-2}$		
Random	0.238 ± 0.025	$\mathbf{0.166 \pm 0.036}$	$\mathbf{0.125 \pm 0.021}$	0.083 ± 0.005	0.065 ± 0.004
SBAL	0.238 ± 0.025	0.200 ± 0.024	0.125 ± 0.009	$\mathbf{0.076 \pm 0.008}$	$\mathbf{0.052 \pm 0.004}$
LCMD	0.238 ± 0.025	0.171 ± 0.007	0.128 ± 0.015	0.083 ± 0.008	0.054 ± 0.004
Core-Set	0.238 ± 0.025	0.224 ± 0.070	0.168 ± 0.020	0.143 ± 0.059	0.083 ± 0.009
Top-K	0.238 ± 0.025	0.211 ± 0.019	0.155 ± 0.016	0.111 ± 0.015	0.073 ± 0.008
			95% Quantile $\times 10^{-2}$		
Random	2.373 ± 0.220	1.619 ± 0.222	1.090 ± 0.050	0.695 ± 0.039	0.516 ± 0.019
SBAL	2.373 ± 0.220	1.723 ± 0.126	$\mathbf{0.980 \pm 0.070}$	$\mathbf{0.510 \pm 0.036}$	$\mathbf{0.313 \pm 0.014}$
LCMD	2.373 ± 0.220	$\mathbf{1.485 \pm 0.121}$	1.038 ± 0.087	0.609 ± 0.061	0.361 ± 0.020
Core-Set	2.373 ± 0.220	1.902 ± 0.379	1.389 ± 0.126	1.102 ± 0.469	0.598 ± 0.095
Top-K	2.373 ± 0.220	1.901 ± 0.100	1.236 ± 0.099	0.739 ± 0.151	0.416 ± 0.039
			99% Quantile $\times 10^{-2}$		
Random	10.192 ± 1.523	7.260 ± 1.226	4.741 ± 0.281	2.893 ± 0.227	1.870 ± 0.099
SBAL	10.192 ± 1.523	4.756 ± 0.215	$\mathbf{2.701 \pm 0.251}$	$\mathbf{1.433 \pm 0.070}$	$\mathbf{0.896 \pm 0.053}$
LCMD	10.192 ± 1.523	$\mathbf{4.198 \pm 0.103}$	2.787 ± 0.210	1.631 ± 0.178	0.991 ± 0.038
Core-Set	10.192 ± 1.523	5.056 ± 0.827	3.526 ± 0.212	2.638 ± 1.069	1.446 ± 0.290
Top-K	10.192 ± 1.523	5.382 ± 0.373	3.174 ± 0.181	1.756 ± 0.448	0.972 ± 0.092

Table 6: Error metrics on CE.

Iteration	1	2	3	4	5
			RMSE		
Random	2.662 ± 0.339	2.162 ± 0.029	1.856 ± 0.106	1.572 ± 0.072	1.362 ± 0.065
SBAL	2.662 ± 0.339	$\mathbf{1.979 \pm 0.226}$	1.790 ± 0.203	1.458 ± 0.140	$\mathbf{1.205 \pm 0.027}$
LCMD	2.662 ± 0.339	1.991 ± 0.293	1.734 ± 0.189	$\mathbf{1.356 \pm 0.081}$	1.277 ± 0.083
Core-Set	2.662 ± 0.339	2.322 ± 0.350	$\mathbf{1.731 \pm 0.168}$	1.613 ± 0.202	1.343 ± 0.186
Top-K	2.662 ± 0.339	2.684 ± 1.129	2.070 ± 0.368	1.623 ± 0.524	1.313 ± 0.106
			50% Quantile		
Random	0.506 ± 0.119	$\mathbf{0.447 \pm 0.156}$	$\mathbf{0.356 \pm 0.111}$	$\mathbf{0.266 \pm 0.087}$	$\mathbf{0.209 \pm 0.034}$
SBAL	0.506 ± 0.119	0.480 ± 0.116	0.543 ± 0.344	0.336 ± 0.063	0.295 ± 0.053
LCMD	0.506 ± 0.119	0.574 ± 0.361	0.412 ± 0.234	0.317 ± 0.065	0.312 ± 0.085
Core-Set	0.506 ± 0.119	0.562 ± 0.154	0.411 ± 0.085	0.433 ± 0.191	0.408 ± 0.120
Top-K	0.506 ± 0.119	0.653 ± 0.165	0.521 ± 0.133	0.483 ± 0.174	0.400 ± 0.065
			95% Quantile		
Random	4.421 ± 0.630	3.491 ± 0.154	2.828 ± 0.314	2.317 ± 0.207	1.927 ± 0.170
SBAL	4.421 ± 0.630	3.308 ± 0.550	2.936 ± 0.370	2.310 ± 0.349	$\mathbf{1.821 \pm 0.128}$
LCMD	4.421 ± 0.630	$\mathbf{3.263 \pm 0.561}$	$\mathbf{2.758 \pm 0.351}$	$\mathbf{2.025 \pm 0.177}$	2.003 ± 0.326
Core-Set	4.421 ± 0.630	4.235 ± 0.899	2.952 ± 0.375	2.690 ± 0.396	2.189 ± 0.437
Top-K	4.421 ± 0.630	5.009 ± 2.402	3.891 ± 0.921	2.911 ± 1.392	2.238 ± 0.289
			99% Quantile		
Random	11.378 ± 1.863	9.135 ± 0.253	7.754 ± 0.507	6.620 ± 0.340	5.735 ± 0.320
SBAL	11.378 ± 1.863	8.295 ± 1.062	$\mathbf{7.195 \pm 0.786}$	6.058 ± 0.573	$\mathbf{4.933 \pm 0.112}$
LCMD	11.378 ± 1.863	$\mathbf{8.196 \pm 0.926}$	7.229 ± 0.609	$\mathbf{5.569 \pm 0.362}$	5.265 ± 0.399
Core-Set	11.378 ± 1.863	9.739 ± 1.416	7.263 ± 0.707	6.646 ± 0.794	5.404 ± 0.722
Top-K	11.378 ± 1.863	11.424 ± 5.585	8.531 ± 1.478	6.466 ± 2.101	5.237 ± 0.417

Table 7: Error metrics on CNS.

Iteration	1	2	3	4
		Pearson		
KS	87.1 ± 3.8	84.9 ± 2.3	78.0 ± 5.4	80.5 ± 3.7
CE	49.2 ± 16.2	62.0 ± 14.6	41.3 ± 22.1	73.8 ± 20.9
CNS	78.2 ± 6.4	78.9 ± 18.0	90.8 ± 2.7	94.3 ± 2.0
Burgers $M = 2$	92.0 ± 6.3	71.3 ± 27.1	71.4 ± 11.5	67.9 ± 18.4
Burgers $M = 6$	89.5 ± 8.2	60.9 ± 26.7	67.9 ± 19.6	
		Spearman		
KS	86.4 ± 2.8	83.0 ± 2.8	83.9 ± 4.2	82.7 ± 0.4
CE	87.4 ± 1.7	83.9 ± 2.1	81.2 ± 1.0	80.5 ± 1.5
CNS	94.6 ± 2.4	93.4 ± 2.3	91.1 ± 3.9	93.4 ± 1.6
Burgers $M = 2$	87.5 ± 2.7	83.2 ± 11.0	75.2 ± 5.0	73.7 ± 5.3
Burgers $M = 6$	90.3 ± 0.9	84.5 ± 2.3	80.8 ± 2.2	

Table 8: Correlation coefficients in percent between the error and the uncertainty averages per trajectory, including the standard deviation. Computed for SBAL on the main experiments as well as the ensemble size ablation experiment.

G Overview of the Generated Datasets

In the following sections, we show visual examples of the data selected by random sampling and SBAL, and the marginal distributions of all PDE and IC parameters afterwards.

G.1 Example Trajectories

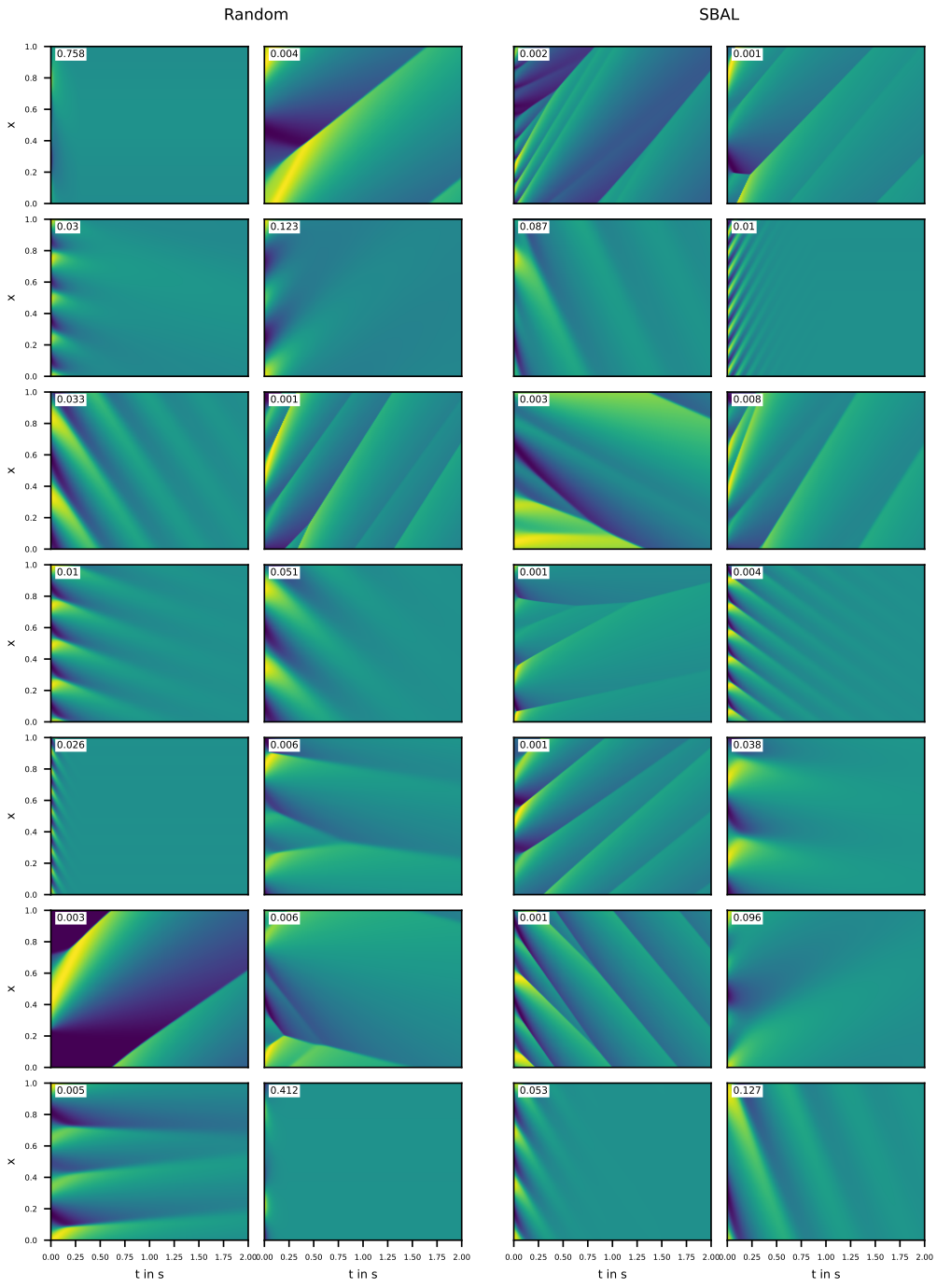

Figure 9: Example ground truth trajectories of random and SBAL on Burgers. The number on the top left of the trajectories shows the PDE parameter ν.

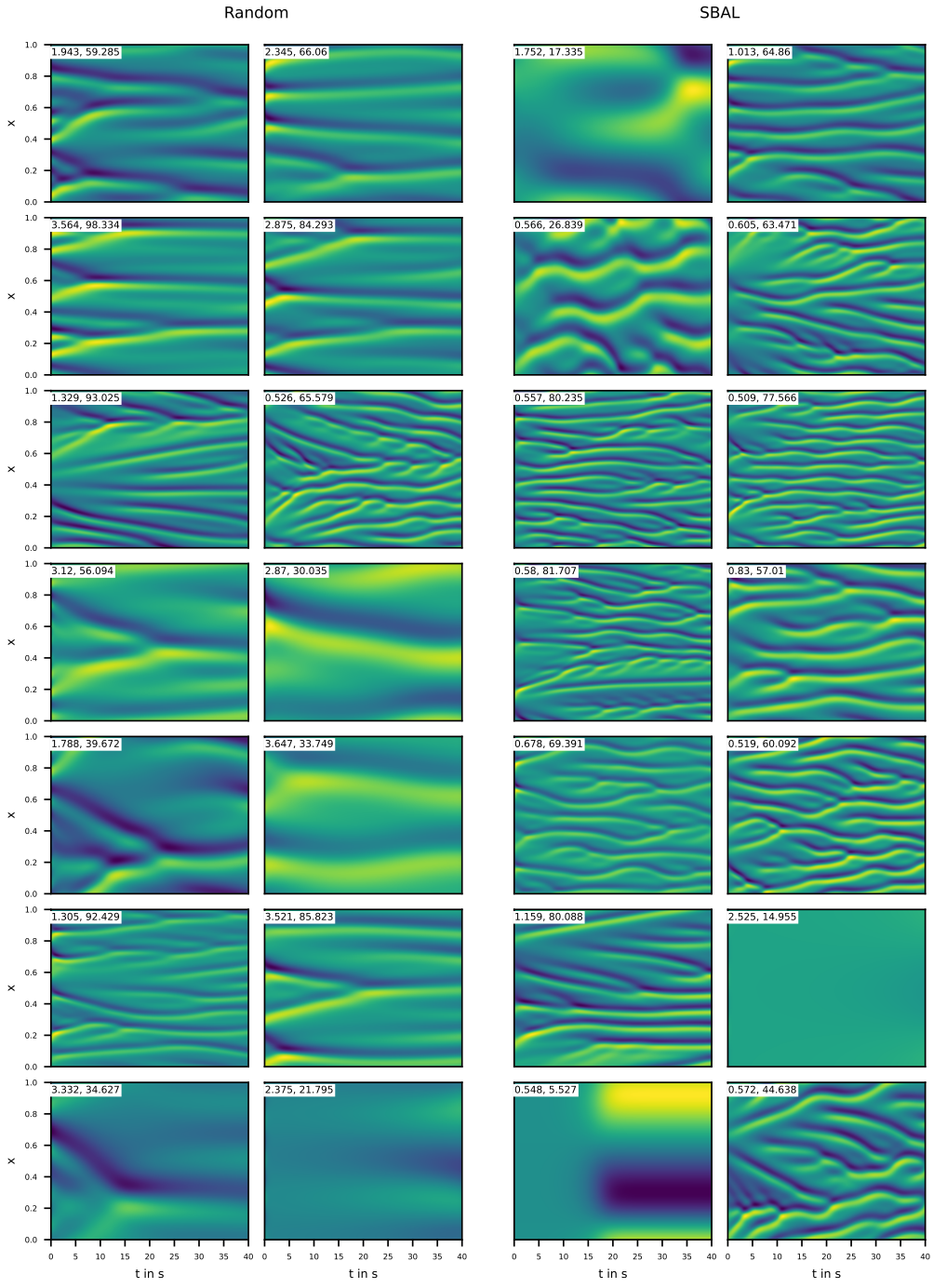

Figure 10: Example ground truth trajectories of random and SBAL on **KS**. The number on the top left of the trajectories shows the parameters (ν, L). The x-axis is shown in normalized values between 0 and 1 independent of the variable domain length L.

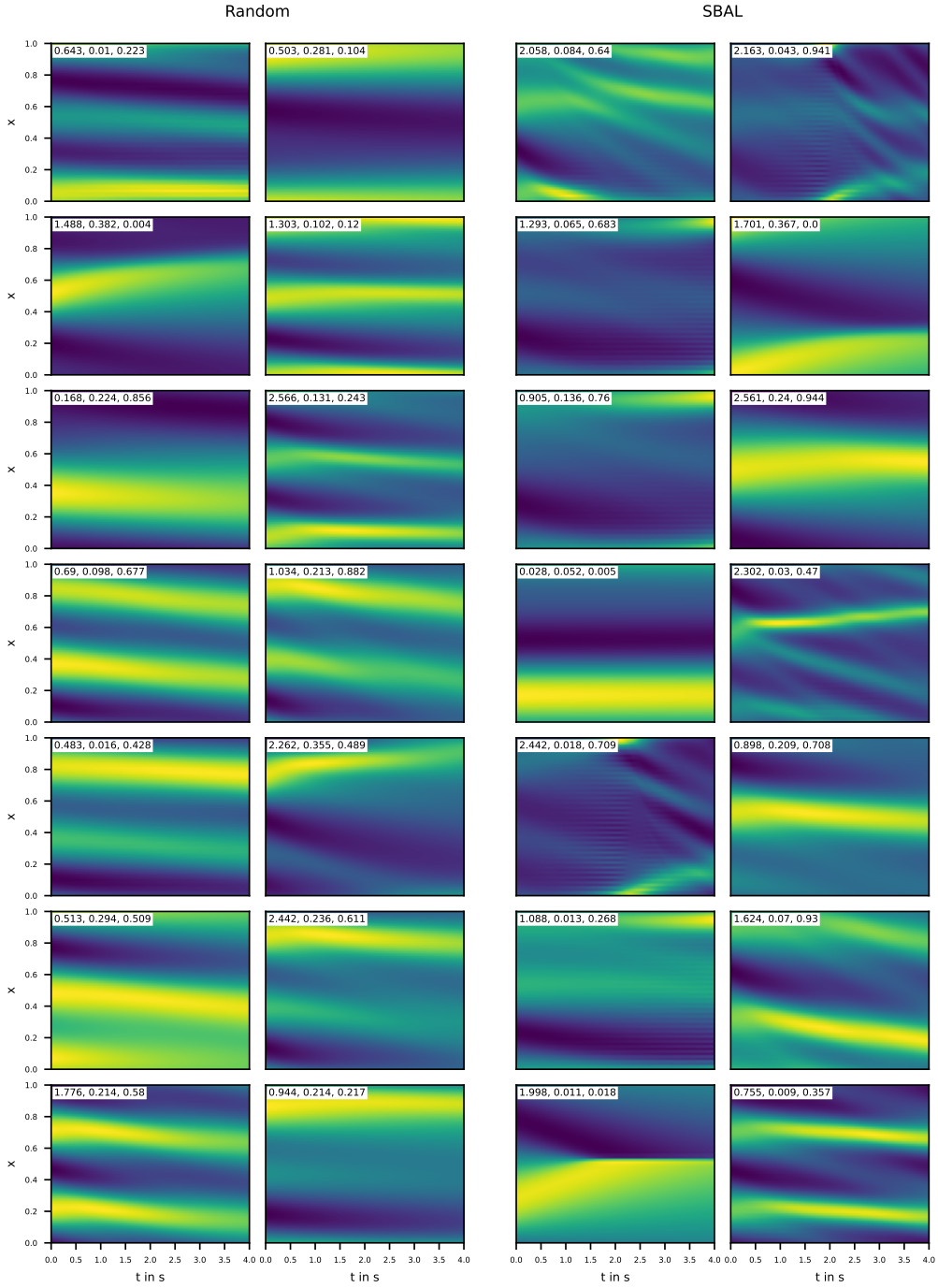

Figure 11: Example ground truth trajectories of random and SBAL on CE. The numbers on the top left of the trajectories shows the PDE parameters (α, β, γ).

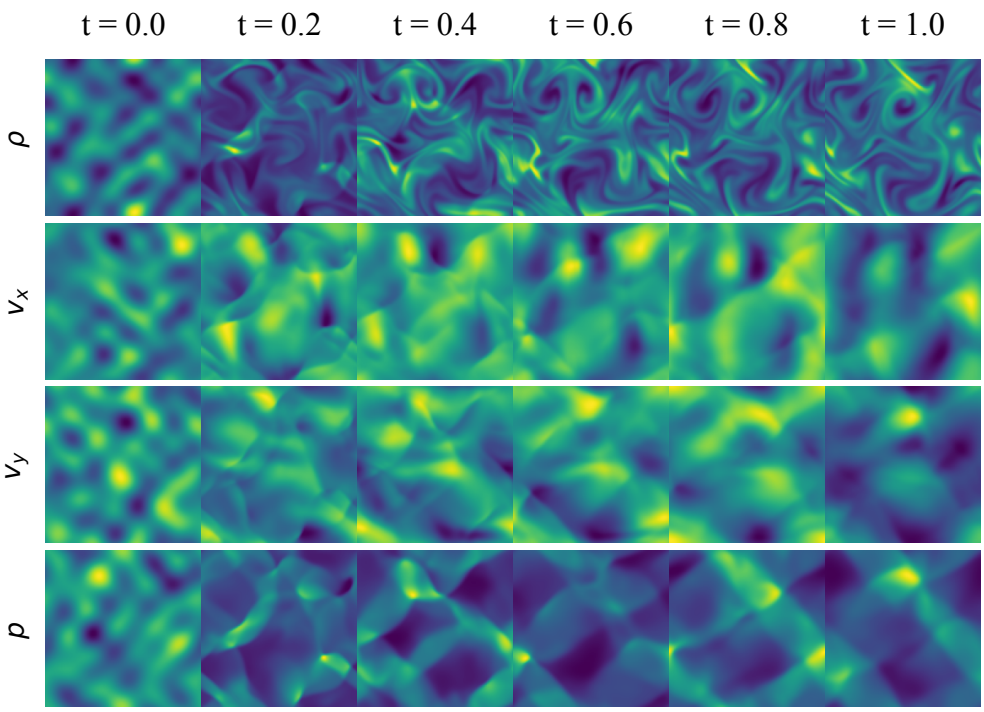

Figure 12: Example ground truth trajectory of CNS.

G.2 IC Parameter Marginal Distributions

Figures 13, 14, 15 and 16 show the marginal distributions of the random parameters of the IC generators, i.e. the random variables drawn which are then transformed using a deterministic function to the actual IC. For example, the KS IC generator draws amplitudes and phases from a uniform distribution and uses them afterward for the superposition of sine waves. If multiple numbers are drawn from each type of variable, we put them together, e.g., in the case of KS, multiple amplitudes are drawn for the different waves, but Fig. 14 only shows the distribution of all amplitude variables mixed. The distribution curves for continuous variables are computed using kernel density estimation. The shaded areas (vertical lines for discrete variables) show the standard deviation between the marginal distributions of different random seeds.

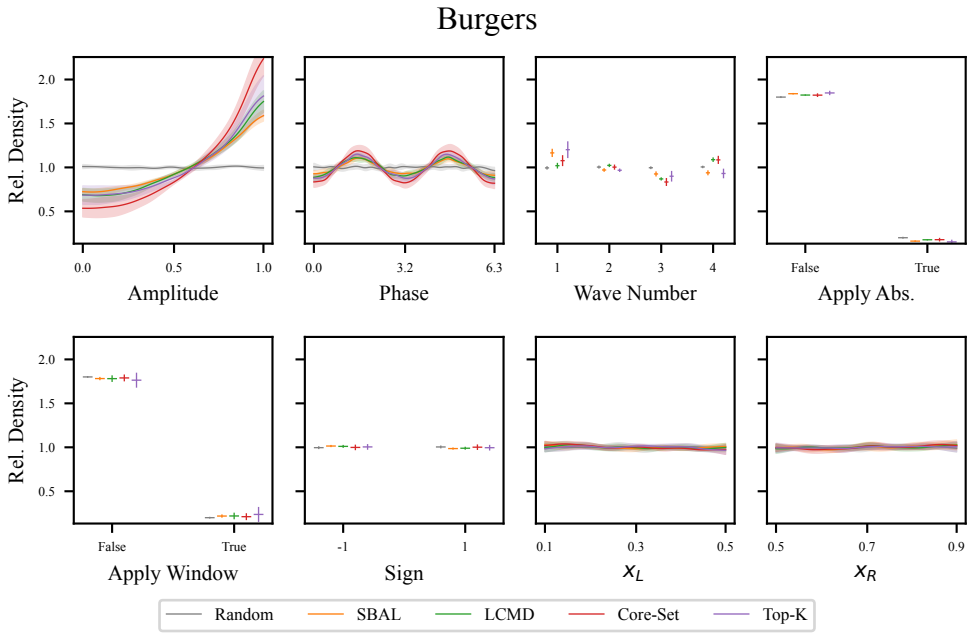

Figure 13: Marginal distribution of the parameters of the ICs sampled by the AL methods for Burgers. Displayed as the ratio to the density of the uniform distribution.

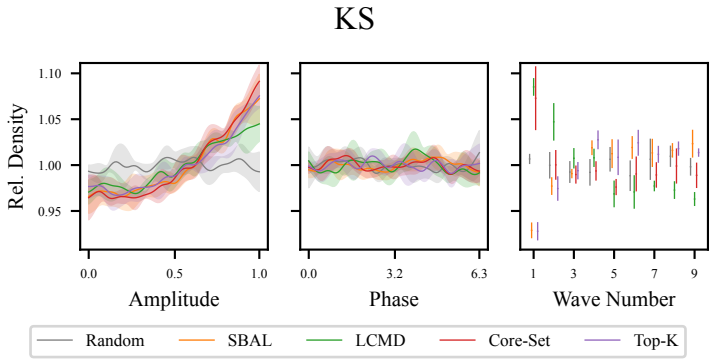

Figure 14: Marginal distribution of the parameters of the ICs sampled by the AL methods for KS. Displayed as the ratio to the density of the uniform distribution.

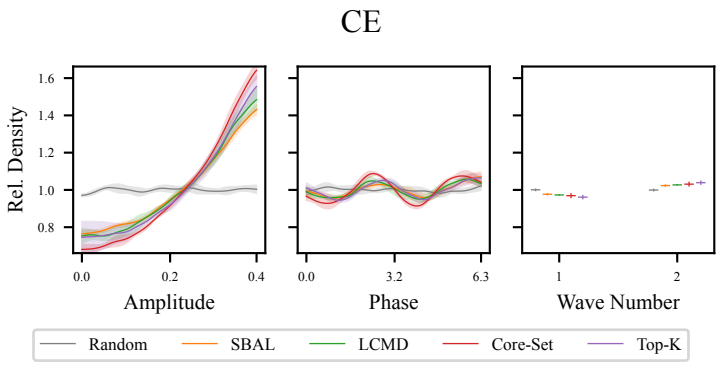

Figure 15: Marginal distribution of the parameters of the ICs sampled by the AL methods for CE. Displayed as the ratio to the density of the uniform distribution.

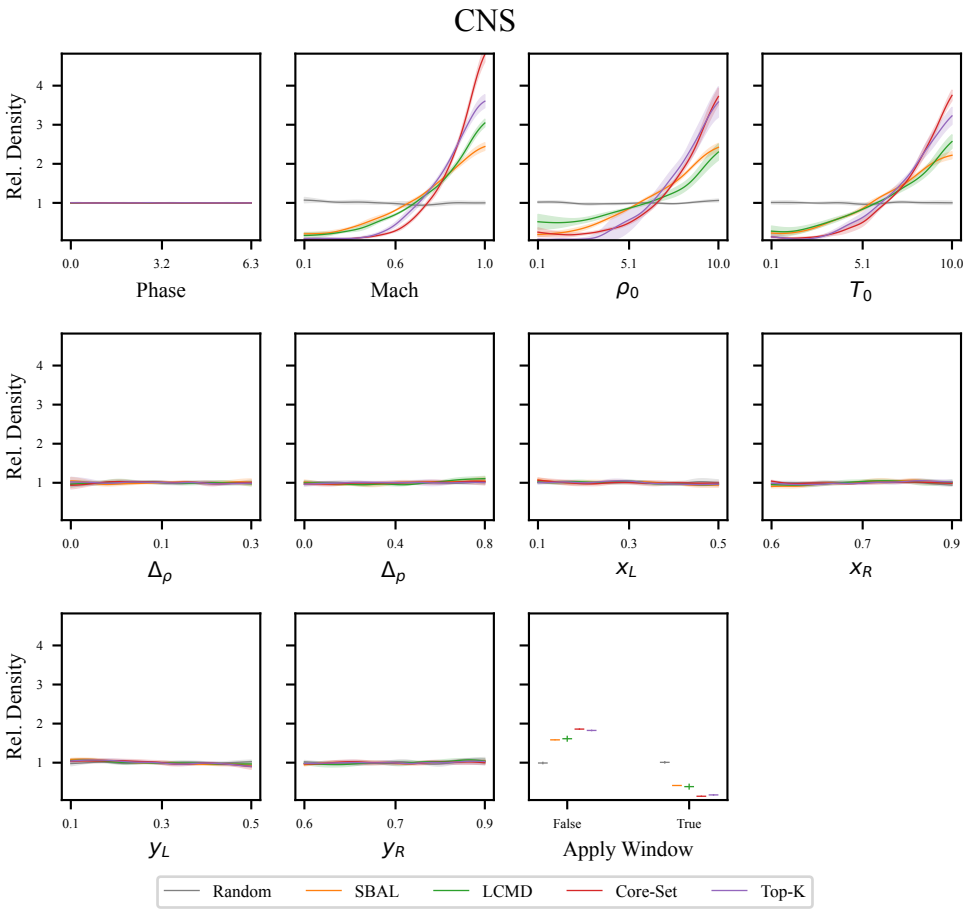

Figure 16: Marginal distribution of the parameters of the ICs sampled by the AL methods for 2D CNS. Displayed as the ratio to the density of the uniform distribution.

G.3 PDE Parameter Marginal Distributions

Similarly, Fig. 17 shows the KDE estimates of the dataset after the final AL iteration for the PDE parameters.

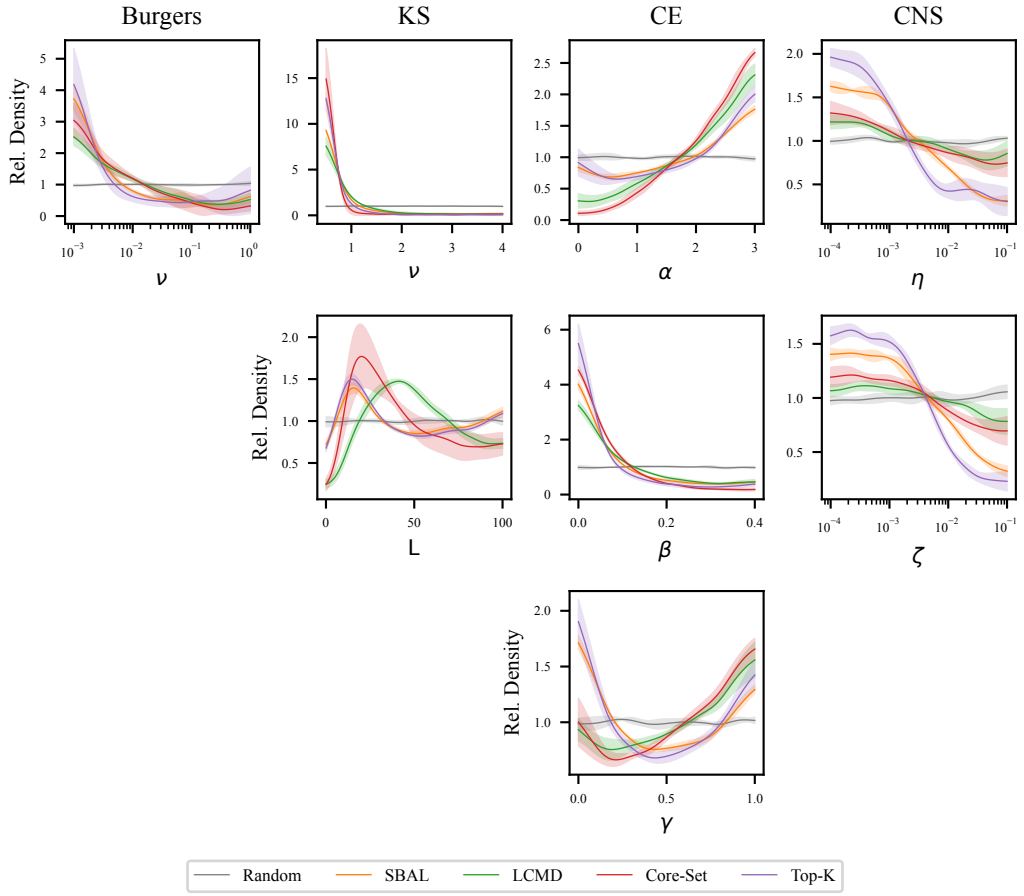

Figure 17: Marginal distribution of the PDE parameters, including the standard deviation between different runs. Displayed as the ratio to the density of the test distribution.