Pre-Generating Multi-Difficulty PDE Data For Few-Shot Neural PDE Solvers

Anonymous authors
Paper under double-blind review

Abstract

A key aspect of learned partial differential equation (PDE) solvers is that the main cost often comes from *generating* training data with classical solvers rather than learning the model itself. Another is that there are clear *axes of difficulty*—e.g., more complex geometries and higher Reynolds numbers—along which problems become (1) harder for classical solvers and thus (2) more likely to benefit from neural speedups. Towards addressing this chicken-and-egg challenge, we study *difficulty transfer* on 2D incompressible Navier-Stokes, systematically varying task complexity along geometry (number and placement of obstacles), physics (Reynolds number), and their combination. Similar to how it is possible to spend compute to *pre-train* foundation models and improve their performance on downstream tasks, we find that by classically solving (analogously *pre-generating*) many low and medium difficulty examples and including them in the training set, it is possible to learn high-difficulty physics from far fewer samples. Furthermore, we show that by combining low and high difficulty data, we can spend 8.9× less compute on pre-generating a dataset to achieve the same error as using only high difficulty examples. Our results highlight that *how* we allocate classical-solver compute across difficulty levels is as important as *how much* we allocate overall, and suggest substantial gains from principled curation of pre-generated PDE data for neural solvers.

1 Introduction

Deep learning has emerged as a powerful paradigm for solving PDEs, enabling data-driven surrogate models that can accelerate simulation, inference, and design across diverse scientific domains (Li et al., 2021a; Lu et al., 2021a; Pathak et al., 2022). This has been driven by the development of specialized models such neural operators (Li et al., 2021a; Lu et al., 2021a) and recent transformer-based extensions (Guibas et al., 2022; Brandstetter et al., 2023), which have demonstrated strong performance on benchmark datasets and gained traction in the machine learning for science community. More recently, significant effort has been devoted towards pre-generation of large datasets such as the Well (Ohana et al., 2024) and pre-training specialized foundation models (FMs) (Herde et al., 2024; Hao et al., 2024; Shen et al., 2024; McCabe et al., 2023). The goal of these FMs is to serve as general-purpose foundations for PDE surrogates: delivering fast inference while minimizing—or even eliminating—the need to retrain on new, potentially harder-to-solve PDEs.

An underlying aspect of this line of work has long been the issue that it seeks to solve PDEs faster than classical numerical solvers but requires examples generated by the latter to do so. While such a circuitous setup is justifiable in many of the inverse problem applications that motivate learned solvers, it is still the case that tasks we eventually want to accelerate—practical engineering tasks in difficult-to-simulate regimes—will be exactly those for which it is hard to generate a significant amount. This need to decrease the sample complexity of neural PDE solvers has spurred significant research drawing up transfer learning (Herde et al., 2024), active learning (Bruna et al., 2024; Musekamp et al., 2024), and other *method-centric* approaches (Rotman et al., 2023).

In this paper we take a *data-centric* view, studying how the training data composition of neural PDE solvers affects their performance. We identify that a key feature of PDE data is that most problem settings have multiple axes of difficulty along which classical solving becomes harder, thus making neural PDE solvers both (potentially) more useful but also more difficult to train due to low data availability. Examples of such difficulty axes include domain geometry features, physics parameters such as the Reynolds number (Re) or Debye length, additional terms due to forcing or compressibility, and so on. To understand how easier-to-generate data affects the performance on harder-to-generate target distributions, we consider incompressible Navier-Stokes simulations with difficulty varying along either or both of (1) *geometry*—as defined by the number and complexity of obstacles in the flow—or (2) *physics* in terms of the flow's Re. For simplicity, we use classical simulation costs to divide these two axes into three difficulty categories—easy, medium, and hard—and investigate how mixing in easy and medium data affects performance on the hard distribution.

Our first key result is that **adding easy-to-medium difficulty data substantially improves performance** on the hard distribution. Naturally, one might expect that medium difficulty data might be more useful, and our second main result is that there is often **a favorable tradeoff justifying pre-generating medium-difficulty datasets** instead of easy ones when solving cross in two classes of simulation, flow past an object (FPO) and lid-driven cavity (LDC), and using both supervised-only neural PDE solvers—specifically the Factorized Fourier Neural Operator (FFNO) (Tran et al., 2021) and the Convolutional Neural Operator (CNO) (Raonic et al., 2023)—and the current state-of-the-art multi-physics pretrained Poseidon FM (Herde et al., 2024). These complementary settings allow us to assess both specialized neural operators and large pretrained models under controlled difficulty-mixing regimes. In more detail, our contributions are the following:

1. Augmenting hard (e.g., multi-obstacle) training with lower-difficulty data (e.g., zero or one obstacle) substantially improves accuracy on the hard test set. For example, most of the performance of Poseidon-B fine-tuned solely on hard FPO data can be recovered when 90% of it is replaced with easy or medium data , which reduces data-generation time 8.9×.
2. Despite the higher generation cost of medium difficulty (e.g., single-obstacle) examples relative to easy (e.g., no obstacle) ones, for most pre-generation budgets one can obtain a better error by training on fewer examples of the former rather than more of the latter. This demonstrates the importance of optimally selecting the pre-generation simulations.
3. Beyond square obstacles, we show that single simple-obstacle data can improve the few-shot performance of models on flows around more complex shapes from FlowBench (Tali et al., 2024), even when given only a few examples from it. This demonstrates the potential of a single dataset serving as a foundation for few-shot training of learned solvers on *multiple* harder datasets.

We will release all pre-generated datasets and code to reproduce our results. For related work, please see Appendix A.

2 Pre-generating datasets for studying difficulty transfer

As discussed in the introduction, PDE tasks often feature gradations of difficulty that significantly increase the cost of simulation, making neural PDE solvers both more expensive to train (because of the complexity of generating the associated data) and potentially more useful (because of their ability to replace said expensive solves). This increased numerical difficulty can stem from shorter timesteps, higher per-timestep cost (e.g., worse conditioning of linear solves), and meshing challenges. To study how low-to-medium difficulty data can improve few-shot performance on high difficulty data, we consider the 2D incompressible Navier-Stokes (INS) equations of fluid flow. Given a domain $\Omega \subset [0,1]^2$, they govern the velocity $\mathbf{u}(\mathbf{x},t)$ and pressure $p(\mathbf{x},t)$ of a fluid at point $\mathbf{x} \in \Omega^\circ$ and time $t \geq 0$ as follows:

$$\partial_t \mathbf{u} + (\mathbf{u}\cdot\nabla)\mathbf{u} + \nabla p = \nu \Delta \mathbf{u} \qquad \text{and} \qquad \nabla \cdot \mathbf{u} = 0 \tag{1}$$

Different simulation settings can be defined using different domains Ω, different boundary conditions $\mathbf{u}(\mathbf{x},t)$ and $\mathbf{p}(\mathbf{x},t)$ for $\mathbf{x} \in \partial\Omega$, different initial conditions $\mathbf{u}(\mathbf{x},0)$ and $\mathbf{p}(\mathbf{x},0)$ for $\mathbf{x} \in \Omega^\circ$, and different kinematic viscosities $\nu \geq 0$. We focus on two canonical settings: (1) flow past an object (FPO), in which the boundary conditions impose two no slip walls

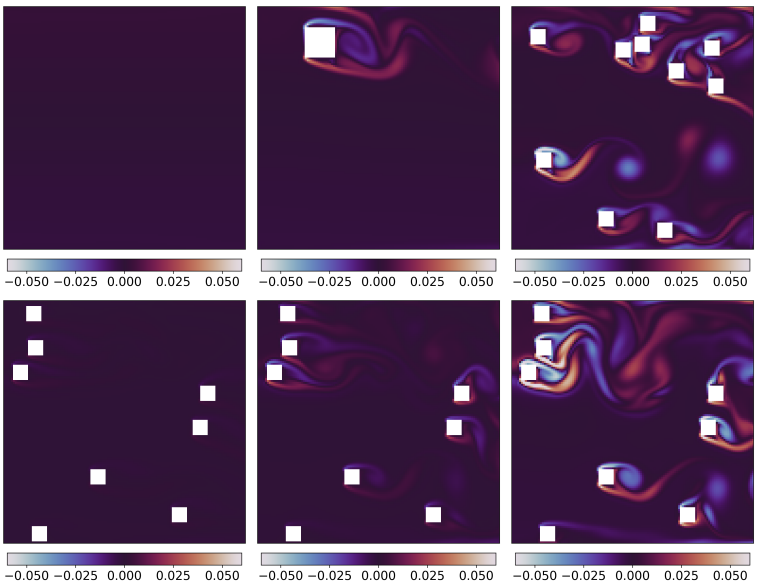


Figure 1: **Top:** vorticity snapshots across increasing geometry difficulty, with flows past zero, one, and multiple (2–10) square obstacles. **Bottom:** snapshots across physics difficulties in the form of low ($[100, 1000]$), medium ($[2000, 4000]$), and high ($[8000, 10000]$) Re bands.

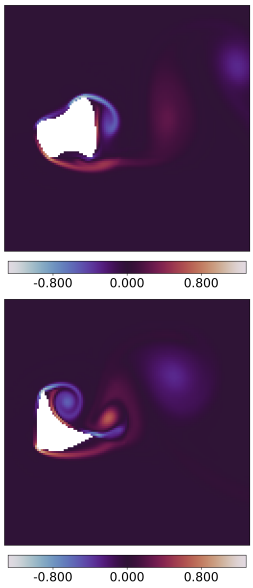


Figure 2: FPO with objects from the FlowBench G1 NURBS data (Tali et al., 2024).

(Dirichlet $u = 0$) around an inlet and an outlet, and (2) lid-driven cavity flow (LDC), which has three no slip walls and a horizontal velocity at the top; in both cases the interior of the domain is at rest to start.

2.1 Difficulty axes

Starting from these basic setups, we vary simulations along three data-difficulty axes: geometry, physics, and their combination. As detailed below, changing the geometry involves modifying the domain Ω and its boundary conditions to add, remove, or change the shape of obstacles, with a greater number of objects or more complex shapes corresponding to greater difficulty. On the other hand, changing the physics involves varying the initial velocity $\mathbf{u}(\mathbf{x}, 0)$ to change the Reynolds number, a dimensionless quantity that when increased typically makes the flow more complex and hard-to-simulate. Figures 1 and 2 illustrate how the vorticity fields of the simulations change with increasing difficulty, while Figure 3 shows the corresponding increase in simulation cost.

We next describe at a high level the settings used to generate the axes' data; further details, including about our OpenFOAM (Jasak et al., 2007) setup, can be found in Appendix B.

1. **Geometry:** A straightforward way to change the problem geometry to increase problem difficulty is by adding or removing obstacles to the flow. In this difficulty axis, we add between zero and ten square obstacles at random, non-overlapping positions. The resulting simulations are categorized as *easy* if they have no obstacles, *medium* if they have one obstacle, and *hard* if they have two or more. The way these changes affect the simulation is illustrated in the top row of Figure 1, and their effect on the FPO generation cost is plotted in Figure 3; in short, more obstacles yield more complex, harder-to-simulate flows.
2. **Physics:** Another way of increasing problem difficulty is to increase the Reynolds number, which is well-known to describe the complexity of a flow. It is defined using a characteristic velocity U and length-scale L to be $\mathrm{Re} = UL/\nu$, so we increase the initial velocity $\mathbf{u}(\mathbf{x}, 0)$ at the inlet (FPO) or the lid (LDC) to make it larger. In particular, we categorize simulations into *easy*, *medium*, and *hard* if the corresponding Re is between $[100, 1000]$, $[2000, 4000]$, and $[8000, 10000]$, respectively; within each band, the Re is sampled from a truncated Gaussian distribution. Figure 1 (bottom) demonstrates how a higher Re induces richer fluid structure, yielding the higher simulation cost (cf. Figure 3).

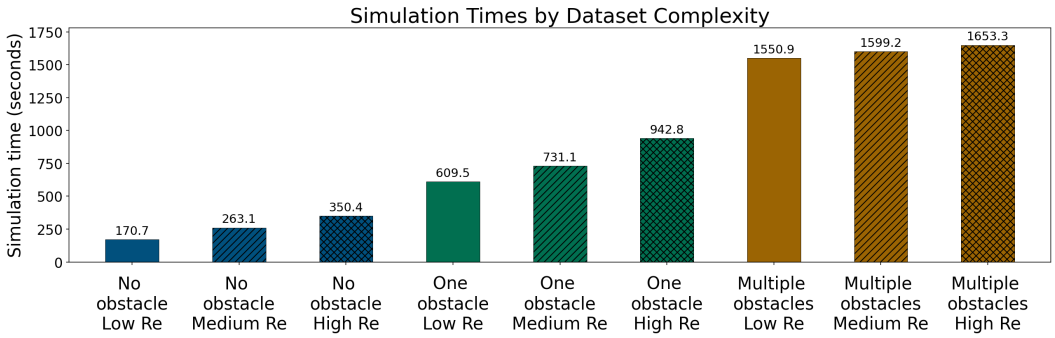


Figure 3: Computational cost of simulating flow past an object (FPO) at different difficulty settings, demonstrating increasing runtime along both the domain geometry axis (increasing number of obstacles) and the physics axis (increasing Reynolds number). The costs reported are averages across thirty simulations.

As mentioned before, we combine the geometry and physics axes to obtain our third difficulty axis; in the latter case we use low Re flows with no objects as the easy examples and medium Re flows with one object as the medium examples. In all cases, we treat "easy/medium/hard" as a relative, cost-based notion of difficulty: configurations that are cheaper to solve (e.g., low-Re, simple geometries) form the easy tier, while those that require substantially more wall-clock time (e.g., high-Re, multi-obstacle flows) form the hard tier (see Figure 3).

Lastly, we also enhance our investigation of geometry difficulty by changing the object complexity, specifically by incorporating simulations from FlowBench (Tali et al., 2024). Specifically, we use their G1 dataset of FPO simulations of flows around Non-Uniform Rational B-Splines (NURBS), two examples of which are depicted in Figure 2. Because FlowBench is external, we do not measure the classical solve difficulty and treat no-obstacle and single (square) obstacle flows from the geometry axis as the easy and medium difficulty settings, respectively.

2.2 Pre-generated datasets

For each of the above settings and difficulty axes (excluding FlowBench) we generate $n = 6,400$ simulations with different randomly sampled initial conditions. Specifically, following Herde et al. (2024) we sample Re between $[100, 1000]$ (or higher if we are varying along the physics axis, as described above) and use that to set the inlet (FPO) or lid (LDC) velocity. We store the solution $\boldsymbol{y}_t^i = (\mathbf{u}_i(\mathbf{x}, t), p_i(\mathbf{x}, t))$ of each simulation $i = 1, \ldots, n$ at $T = 20$ timesteps $t = 1, \ldots, T$ on a regular grid of points $\mathbf{x} \in \Omega$. Starting with this data, we hold out a subset of $N = 100$ trajectories and set the goal of a learned PDE solver as using the remaining data to learn a model that, given the initial conditions $\boldsymbol{y}_0^i = (\mathbf{u}_i(\mathbf{x}, 0), p_i(\mathbf{x}, 0))$ of a held-out trajectory i, predicts a trajectory $\hat{\boldsymbol{y}}_t^i$ where $t \in [1, T]$ and $\hat{\boldsymbol{y}}_t^i = (\hat{\mathbf{u}}_i(\mathbf{x}, t), \hat{p}_i(\mathbf{x}, t))$. Following Raonic et al. (2023); Herde et al. (2024), we measure its success at doing so using the mean relative L1 error (nMAE):

$$\text{nMAE} = \sum_{i=1}^{N} \sum_{t=1}^{T} \left\| \boldsymbol{y}_t^i - \hat{\boldsymbol{y}}_t^i \right\|_1 \Big/ \sum_{i=1}^{N} \sum_{t=1}^{T} \left\| \boldsymbol{y}_t^i \right\|_1 \tag{2}$$

3 Empirical results

We now turn to our empirical investigation, in which we evaluate several supervised and foundation models while varying the difficulty composition of their training and fine-tuning data along the difficulty axes described in Section 2.1. This results in three key takeaways: mixing in lower difficulty data can be sufficient for strong performance (Section 3.1), it can be beneficial to mix in a few medium difficulty examples rather than many easy ones (Section 3.2), and there is potential for "foundation datasets" that have strong few-shot performance on diverse data, as suggested via few-shot evaluations on FlowBench (Section 3.3). Crucially, throughout we are interested in the model's performance a *target distribution* consisting only

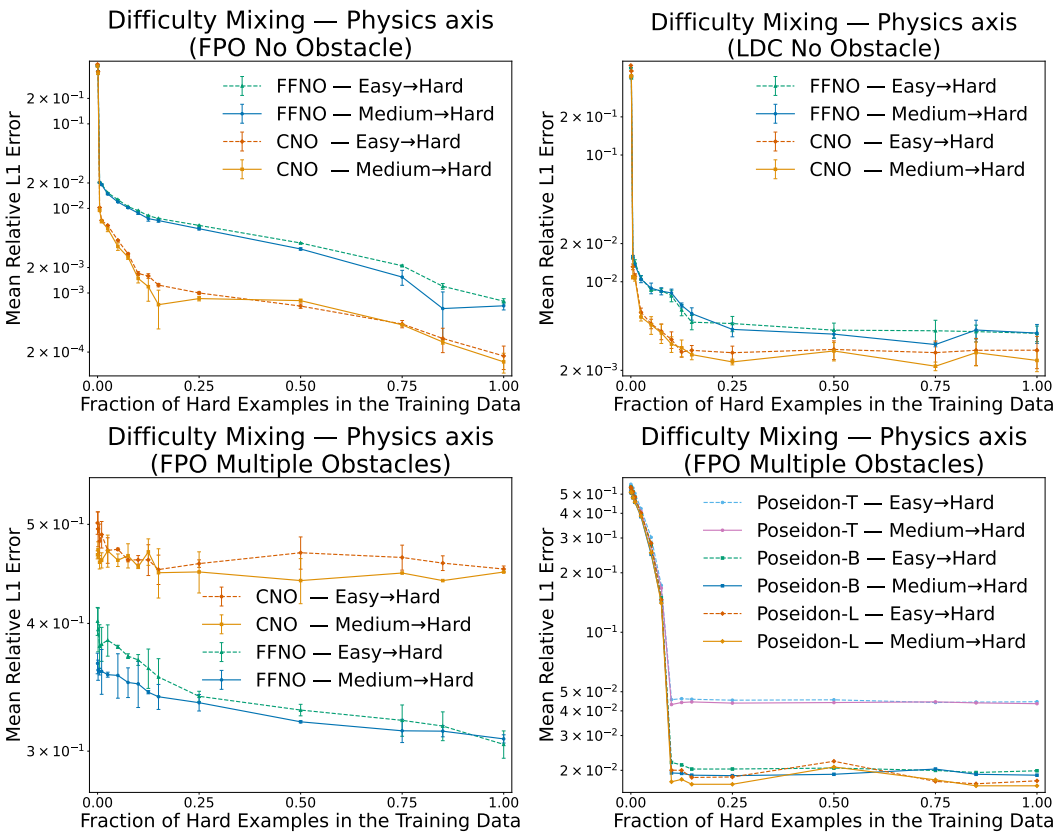


Figure 4: Performance on hard (high Re) examples while varying the data composition. We fix the total number of training examples to 800 and show the error of various models as the fraction of the data consisting of high Re ($\in [8000, 10000]$) examples increases. Here the easy examples and medium examples are low Re ($\in [100, 1000]$) and medium Re ($\in [2000, 4000]$), respectively. The two row evaluates supervised models on no-obstacle FPO (left) and LDC (right), the bottom left evaluates supervised models on flows past multiple objects, and the right evaluates multiple Poseidon FMs on flows past multiple objects. Across all results we observe that a small fraction of lower difficulty examples is able to recover much of the performance of neural PDE solvers trained on solely hard (target) examples.

of the relevant axis's hard examples, which we estimate by evaluating on a held out set. For example, if we train on mixture of no-obstacle, single-obstacle, and multi-obstacle training examples, we report performance on a test set drawn from only the latter's distribution.

The specific supervised models we consider are the Convolutional Neural Operator (CNO) (Raonic et al., 2023) and the Factorized Fourier Neural Operator (FFNO) (Tran et al., 2021), which have demonstrated strong performance on several benchmarks (Ohana et al., 2024; Tali et al., 2024; Dauner et al., 2024; Takamoto et al., 2022; Koehler et al., 2024). These two models are trained from scratch on the different training mixtures we consider. To see whether our findings continue to hold in the higher performance regimes enabled by large-scale multi-physics pretraining, we also consider the Poseidon family of FMs trained on diverse PDE families (Herde et al., 2024), evaluating three variants: Tiny (21M parameters), Base (158M), and Large (629M). Unlike CNO and FFNO, in this case we train or *fine-tune* the models on our training mixure starting from a model checkpoint pretrained on diverse PDE families. Training details of all models are reported in the Appendix.

3.1 Training on simpler examples goes a long way

We start with **difficulty-mixing** evaluations, in which we fix the *total* number of training points to $n = 800$ and vary the proportion allocated to hard examples from the target distribution. Our main finding is that adding a small set of hard examples to otherwise lower

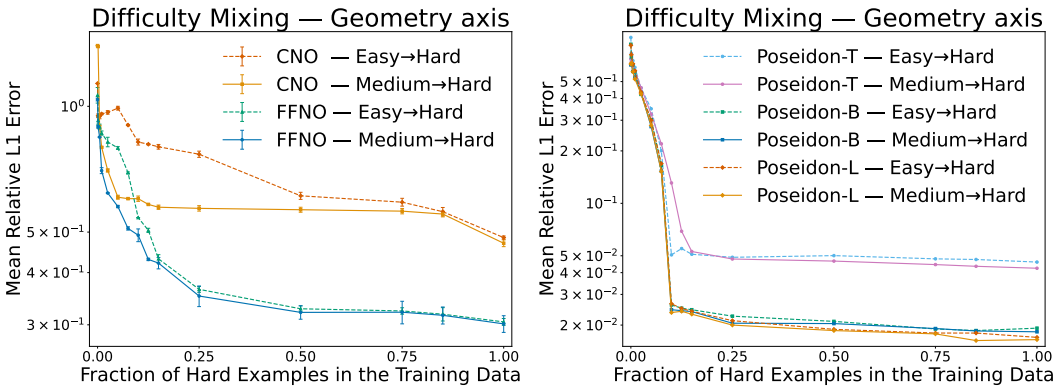


Figure 5: Performance on hard (multi-obstacle) FPO while varying data composition. The total number of training examples is fixed to 800 and we evaluate using varying fractions of zero obstacle (easy) and single obstacle (medium) simulations in the training data. As with varying Re, for both supervised models (left) and Poseidon FMs (right), a small number of lower difficulty examples suffices to recover most of the performance of models trained on entirely hard examples.

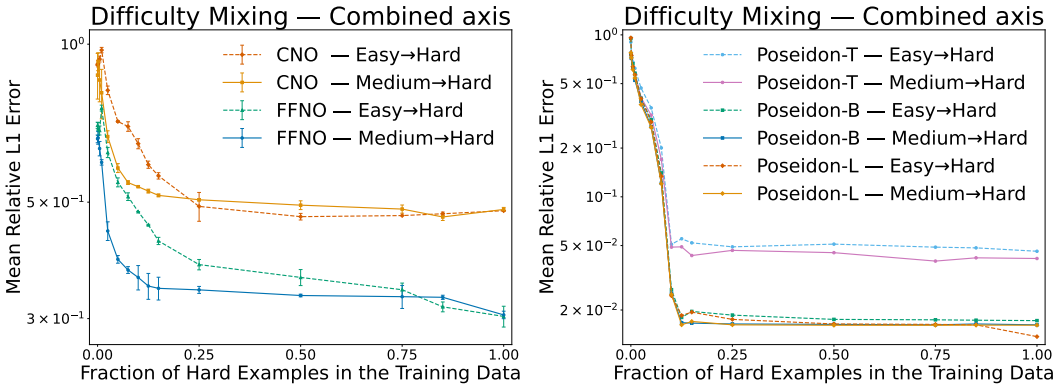


Figure 6: Performance on hard (multi-obstacle and high Re) FPO while varying data composition along both physics and geometry. The total number of training examples is fixed to 800 and we evaluate using varying fractions of zero obstacle low Re flow (easy) and single obstacle medium Re flow (medium) simulations in the training data. As with varying Re and geometry separately, for both supervised models (left) and Poseidon FMs (right), a small number of lower difficulty examples suffices to recover most of the performance of models trained on entirely hard examples.

difficulty (easy and medium) training data is sufficient to recover most of the performance of training on a dataset where all 800 examples are hard. Below we discuss how this manifests along specific difficulty axes. Note that the total number $n = 800$ of training points was determined by training only on hard data using several candidate budgets n and finding that the test error plateaued after around 800 examples; we standardize this budget throughout this subsection, although as discussed in Figure 11 our main finding holds for other budgets as well.

1. **Physics axis** (Fig. 4): While models trained on lower difficulty examples do poorly on the hard (high Re) test examples, replacing just 10% of them by target distributions examples recovers most of the benefit of training fully on the latter. Notably, using the numbers in Figure 3 we see that the former involves 8.9× less compute time. At 10% hard examples, Poseidon-B typically reduces error by about 96% at 10% hard data, while CNO and FFNO show $\approx 98\%$ reductions in the no-obstacle and $\approx 6\%$ in the multi-obstacle setting. Increasing the proportion of hard examples provides incremental gains until 25% and plateaus after.
2. **Geometry axis** (Fig. 5): The same pattern when composing multi-object FPO with flows past zero or one objects: the main improvement for CNO, FFNO, and Poseidon is obtained when only 10% of the data is from the target distribution. In particular, at

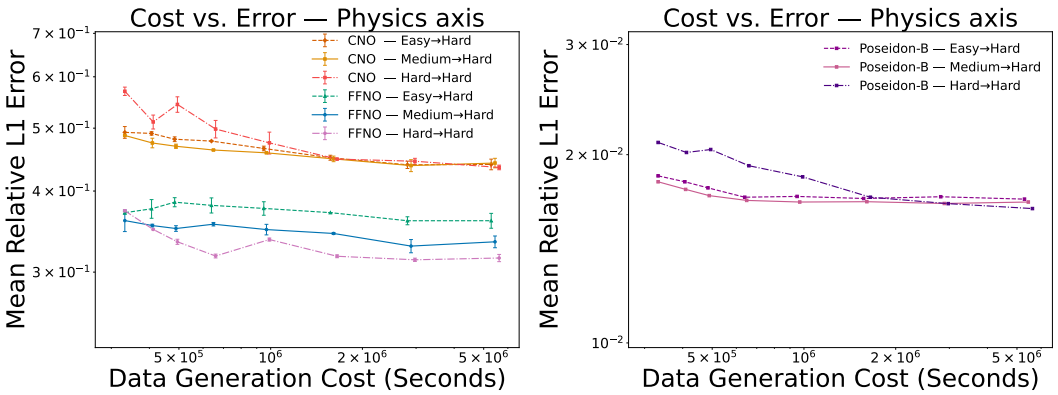


Figure 7: Comparing data generation cost vs. error while augmenting hard (multi-obstacle high Re) FPO examples with easy (multi-obstacle low Re), medium (multi-obstacle medium Re), and hard (multi-obstacle high Re) examples. We fix the number of hard examples to 200 and plot the compute required to generate them and between 1 and 3200 lower and equal difficulty examples. For both supervised models (left) forand Poseidon-B (right), generating medium difficulty data has a generally more favorable tradeoff, achieving the same or lower error at the same budget. At sufficiently large compute budgets, however, training exclusively on hard data (hard to hard) yields the lowest error.

that percentage Poseidon-B improves by roughly 96–97% in terms of error relative to training on all-easy examples. Additional hard data yields only modest improvement.

3. **Combined axis** (Fig. 6): We observe similar behavior when varying along the combined domain geometry and problem physics axis, with most of the benefit of training on the target distribution obtained with 10% examples and improving only modestly afterwards.

In summary, across all three difficulty axes and all model families, we consistently find that a small hard fraction (often around 10%) is enough to obtain most of the performance of hard-only training; results change only marginally beyond ≈25% data from the target distribution.

3.2 Cost-effectiveness of pre-generating fewer medium difficulty examples

Having demonstrated that low-cost simulation data can be added to just a few (harder-to-obtain) examples from the target distribution to recover much of the performance trained solely on the hard examples, we now examine the cost vs. error tradeoffs of using data at different points on the difficulty axis. In particular, we examine whether there are regimes in which it is favorable to generate and train on medium difficulty (e.g. single-obstacle or intermediate Re) examples rather than easy examples. To do so we fix the number of hard examples to 200 and vary the number of lower difficulty examples added to the training mix between 1 and 3200. For completeness, we also evaluate a *hard-on-hard* variant that augments the N_{hard}=200 seed with additional target-distribution (high-Re, multi-obstacle) samples. This setting delivers the lowest error per added sample but at the highest pre-generation cost, so we use it primarily as an upper-bound reference when comparing cost-normalized tradeoffs to medium- and easy-on-hard mixes. Since medium difficulty examples are more costly to generate than easy ones, we study how the error varies as a function of the pre-generation cost. Our main finding is that there indeed are many pre-generation budgets at which the error obtained by training on medium difficulty examples is lower than that obtained by training on (more) easy examples. Below we discuss the extent to which this holds along specific difficulty axes.

1. **Physics axis** (Fig. 7): While the increase in data generation cost is small when going from low to medium Re, for both FFNO and Poseidon the error of the model trained on the latter is lower at all data generation costs evaluated, demonstrating the value of using intermediate rather than easy examples when targeting a hard distribution.
2. **Geometry axis** (Fig. 8): Unlike changing physics, changing the domain geometry by adding obstacles significantly increases computational cost. Nevertheless, for the supervised models (CNO and FFNO) it is usually favorable to train on FPO simulations

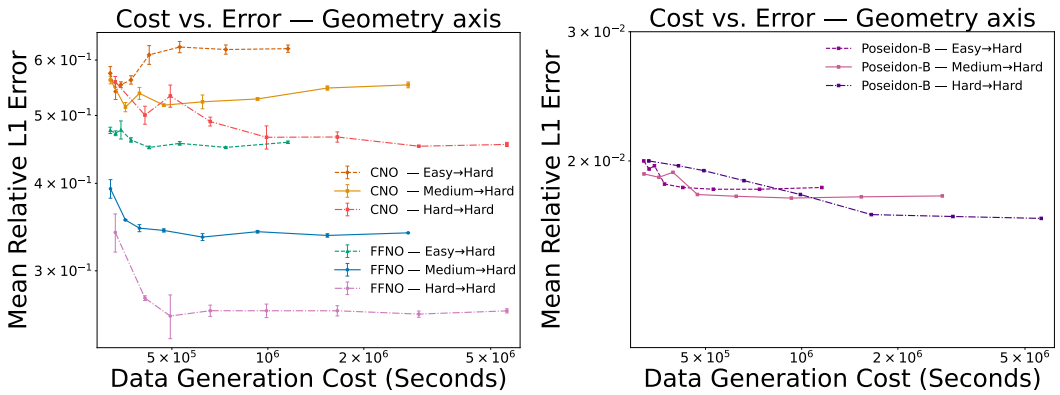


Figure 8: Comparing data generation cost vs. error while augmenting hard (multi-obstacle) FPO with easy (no obstacle), medium (single obstacle), and hard (multi-obstacle) examples. We fix the number of hard examples to 200 and plot the compute required to generate them and between 1 and 3200 lower and equal difficulty examples. For supervised models, generating medium difficulty data has a generally more favorable tradeoff, achieving the same or lower error at the same budget; for Poseidon-B, generating medium data is more cost-effective given 5e5 seconds or more time for pre-generation. At sufficiently large compute budgets, however, training exclusively on hard data (hard to hard) yields the lowest error.

with one (medium) rather than no (easy) obstacles at all computational budgets. For the better-performing Poseidon FM, training on medium difficulty examples is cost-effective at data-generation budget of 5e5 seconds and higher.

3. **Combined axis** (Fig. 9): when the hard examples involve a high Re flow past multiple objects, we find that augmenting with medium difficulty examples performs better than or the same as using (more) easy examples at the same generation budget. However, we also find that for both supervised models and the Poseidon FM that adding increasingly more low difficulty examples starts to increase the error on the target distribution, demonstrating that care needs to be taken when doing this data composition.

In summary, across all difficulty axes we find that it is cost-effective or at least not significantly harmful to train on medium difficulty rather than easy examples, despite the former's greater generation cost. This result demonstrates the importance of considering multiple scales of difficulty when pre-generating a data mixture for a specific high difficulty target distribution.

3.3 Towards foundation datasets

In our last evaluation, we study the implications of multi-difficulty training for large-scale pre-generation of datasets for solving diverse PDE tasks. As an example of the latter, we consider examples from the FlowBench dataset (Tali et al., 2024) of flows past irregular NURBS objects (cf. Section 2.1). Using flows past zero objects and flows past one square object as the easy and medium distributions, respectively, we show in Figure 10 that adding these simpler examples can reduce the error. This is especially pronounced in the case of FFNO when its training data is augmented with single object FPO examples. This suggests the potential utility of pre-generating large medium difficulty datasets and reusing them on multiple other datasets, thus amortizing the pre-generation costs. This pipeline is analogous to that of pre-training a foundation model, the cost of which is amortized as it is applied to multiple downstream tasks.

This idea of a pre-generated "foundation dataset" can also be used to describe much larger-scale efforts like The Well (Ohana et al., 2024). However, what our study demonstrates is that, just like the length and quality of web data used to train large language models matters, so does the difficulty of data pre-generated for training PDEs. In particular, as discussed in the last section, medium (across any axis) data can be much more effective as a mixing dataset than easy (across any axis) data for multi-obstacle performance. The current section further shows this for FlowBench NURBS data. Thus, when pre-generating such large datasets, it will be important to incorporate settings that more closely approach the types of more difficult problems that will be of actual interest to future users of neural PDE solvers.

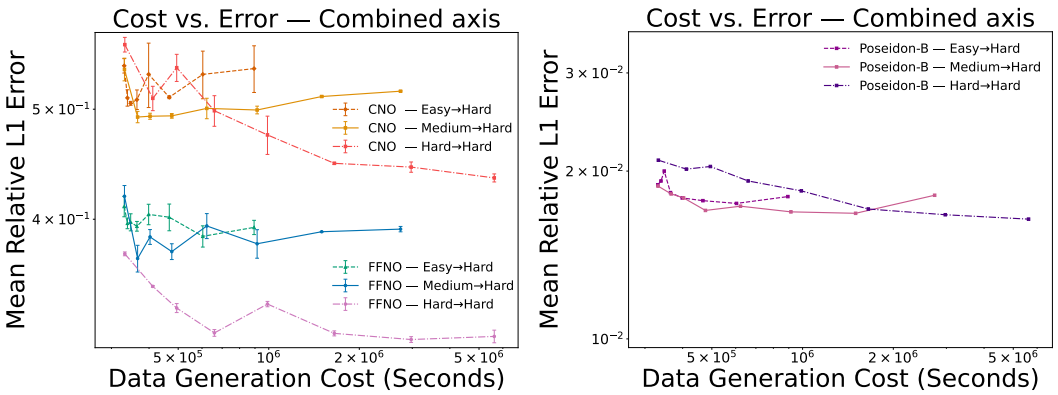


Figure 9: Comparing data generation cost vs. error while augmenting hard (multi-obstacle high Re) FPO examples with easy (no obstacle low Re), medium (single obstacle medium Re), and hard (multi-obstacle high Re) examples. We fix the number of hard examples to 200 and plot the compute required to generate them and between 1 and 3200 lower and equal difficulty examples. For both supervised models and Poseidon-B, generating medium difficulty data has a generally more favorable tradeoff, achieving the same or lower error at the same budget as easy data. However, in all cases, too many lower difficulty examples can reduce performance. At sufficiently large compute budgets, training exclusively on hard data (hard to hard) yields the lowest error.

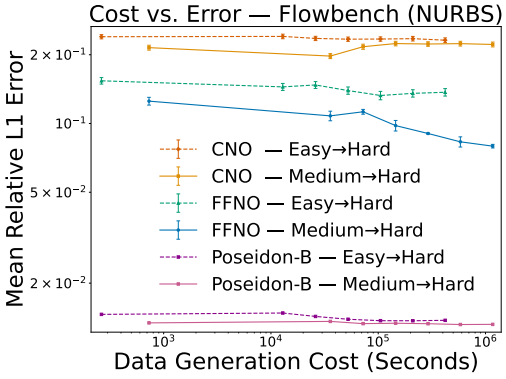


Figure 10: Performance on FlowBench's NURBS FPO simulations when 200 target examples are augmented with 1-3200 zero obstacle FPO (easy) or single square-obstacle FPO (medium) simulations. In multiple cases such, e.g. adding medium examples when training FFNO, doing this data augmentation substantially improves performance on the target FlowBench distribution.

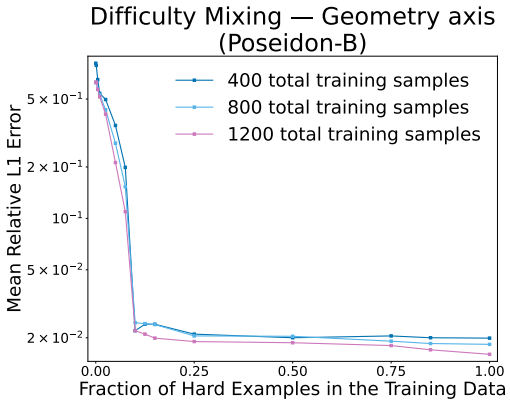


Figure 11: Performance of Poseidon-B as the fraction of target distribution data increases. Each curve fixes the training set size and varies the number of medium (one obstacle) vs. hard (multi-obstacle) examples in it. In all three cases, most of the improvement over training on just single-obstacle examples is obtained by replacing just 10% of the data with target distribution examples.

4 Conclusion

This paper presents a data-centric study on the role of training data composition for neural PDE solvers. Our study considers three difficulty axes comprising the *domain geometry* in the form of the number and shape of flow obstacles, the *problem physics* in the form of the Reynolds number, and the combination of the two. Across all settings we find that examples from lower-difficulty settings can significantly improve the performance on the associated hard test distribution. Furthermore, this result holds for both supervised models like CNO and FFNO as well as the Poseidon family of state-of-the-art multi-physics-pretrained FMs. This suggests that our observed performance gains are not only function of model class or

model capacity but also a function of the quality and difficulty level of the training data distribution, as well as how it relates to the target distribution. In particular, we show that incorporating intermediate-difficulty examples has significant benefits. Therefore, for a fixed computational budget, it may be more cost-effective to generate smaller number of high-quality moderately complex—i.e., intermediate—data, rather than relying on large volumes of simpler data. Hence, our work suggests that future data-generation workflows for neural PDE solvers should take into account tradeoffs between the difficulty of generating low-to-medium-to-high complexity data and the potential benefits of harder-to-simulate data for learning that target distribution.

Reproducibility Statement

We took several steps to make our results easy to reproduce. Problem setups, difficulty axes, dataset sizes/splits, and the evaluation metric are specified in the main text (Secs. 2–3), including the definition of nMAE in Eq. (2) and the exact target-and-mix protocols summarized in Figures 4-9. The full simulation pipeline for generating FPO and LDC datasets—covering domain construction, boundary conditions, Reynolds-number sampling, time scheduling, discretization schemes, solver settings, post-processing to a 128×128 grid, and the saved data format—is documented in the **Supplementary Material, App. B** (FPO in Sec. B.1, LDC in Sec. B.2, scheduling in Sec. B.3, numerics in Sec. B.4, and data format in Sec. B.4.9; Table 1; Figs. 12, 13, 14). Model architectures, training/fine-tuning procedures, hyperparameters, and the compute environment are detailed in **Supplementary, App. C** (CNO in Sec. C.1, F-FNO in Sec. C.2, Poseidon variants in Secs. C.3–C.5), with training-time summaries in **App. C.6** (Table 2). External corpora and out-of-distribution geometry experiments using FlowBench, and how they are combined with our pre-generated data, is described in Section. 3.3. We will release all pre-generated datasets and the code used to produce our figures and results to enable exact replication.

References

Cem Anil, Yuhuai Wu, Anders Andreassen, Aitor Lewkowycz, Vedant Misra, Vinay Ramasesh, Ambrose Slone, Guy Gur-Ari, Ethan Dyer, and Behnam Neyshabur. Exploring length generalization in large language models. *Advances in Neural Information Processing Systems*, 35:38546–38556, 2022.

Johannes Brandstetter, Max Welling, and Daniel E. Worrall. Message passing neural pde solvers. In *International Conference on Learning Representations (ICLR)*, 2022.

Johannes Brandstetter, Daniel Worrall, and Max Welling. Message passing neural pde solvers, 2023. URL https://arxiv.org/abs/2202.03376.

Joan Bruna, Benjamin Peherstorfer, and Eric Vanden-Eijnden. Neural galerkin schemes with active learning for high-dimensional evolution equations. *Journal of Computational Physics*, 496:112588, 2024.

Steven L. Brunton and J. Nathan Kutz. Machine learning for fluid mechanics. *Annual Review of Fluid Mechanics*, 52:477–508, 2020.

Hanseul Cho, Jaeyoung Cha, Pranjal Awasthi, Srinadh Bhojanapalli, Anupam Gupta, and Chulhee Yun. Position coupling: Improving length generalization of arithmetic transformers using task structure. *arXiv preprint arXiv:2405.20671*, 2024.

Maximilian Dauner et al. Residual factorized fourier neural operator for simulation of three-dimensional turbulent flows. In *OpenReview preprint (ICLR submission) id: yGdoTL9g18*, 2024. URL https://openreview.net/forum?id=yGdoTL9g18.

John Guibas, Morteza Mardani, Zongyi Li, Andrew Tao, Anima Anandkumar, and Bryan Catanzaro. Adaptive fourier neural operators: Efficient token mixers for transformers, 2022. URL https://arxiv.org/abs/2111.13587.

Zhongkai Hao, Chang Su, Songming Liu, Julius Berner, Chengyang Ying, Hang Su, Anima Anandkumar, Jian Song, and Jun Zhu. Dpot: Auto-regressive denoising operator transformer for large-scale pde pre-training. *arXiv preprint arXiv:2403.03542*, 2024.

Peter Hase, Mohit Bansal, Peter Clark, and Sarah Wiegreffe. The unreasonable effectiveness of easy training data for hard tasks. *arXiv preprint arXiv:2401.06751*, 2024.

Maximilian Herde, Bogdan Raonic, Tobias Rohner, Roger Käppeli, Roberto Molinaro, Emmanuel de Bézenac, and Siddhartha Mishra. Poseidon: Efficient foundation models for pdes. *Advances in Neural Information Processing Systems*, 37:72525–72624, 2024.

Jordan Hoffmann, Yohai Bar-Sinai, Lisa M. Lee, Jovana Andrejevic, Shruti Mishra, Shmuel M. Rubinstein, and Chris H. Rycroft. Machine learning in a data-limited regime: Augmenting experiments with synthetic data uncovers order in crumpled sheets. *Science Advances*, 5 (4):eaau6792, 2019. doi: 10.1126/sciadv.aau6792. URL https://www.science.org/doi/abs/10.1126/sciadv.aau6792.

Hrvoje Jasak. *Error analysis and estimation for the finite volume method with applications to fluid flows*. PhD thesis, Imperial College London, University of London, 1996.

Hrvoje Jasak, Aleksandar Jemcov, Zeljko Tukovic, et al. Openfoam: A c++ library for complex physics simulations. In *International workshop on coupled methods in numerical dynamics*, volume 1000, pages 1–20. Dubrovnik, Croatia), 2007.

George Em Karniadakis, Ioannis G. Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.

Frederic Koehler et al. A benchmark for autoregressive neural emulators of pdes. In *NeurIPS 2024 Datasets and Benchmarks Track*, 2024. URL https://arxiv.org/abs/2411.00180.

Nikola B. Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew M. Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces. *Journal of Machine Learning Research*, 24:1–79, 2023.

Aditi S. Krishnapriyan, Amir Gholami, Shandian Zhe, Robert M. Kirby, and Michael W. Mahoney. Characterizing possible failure modes in physics-informed neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/df438e5206f31600e6ae4af72f2725f1-Abstract.html.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations, 2021a. URL https://arxiv.org/abs/2010.08895.

Zongyi Li, Nikola B. Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations (ICLR)*, 2021b.

Zongyi Li, Kamyar Gupta, Nikola B. Kovachki, Kamyar Azizzadenesheli, and Anima Anandkumar. Fourier neural operator with learned deformations for pdes on general geometries. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, March 2021a. ISSN 2522-5839. doi: 10.1038/s42256-021-00302-5. URL http://dx.doi.org/10.1038/s42256-021-00302-5.

Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, 2021b.

Michael McCabe, Bruno Régaldo-Saint Blancard, Liam Holden Parker, Ruben Ohana, Miles Cranmer, Alberto Bietti, Michael Eickenberg, Siavash Golkar, Geraud Krawezik, Francois Lanusse, et al. Multiple physics pretraining for physical surrogate models. *arXiv preprint arXiv:2310.02994*, 2023.

Daniel Musekamp, Marimuthu Kalimuthu, David Holzmüller, Makoto Takamoto, and Mathias Niepert. Active learning for neural pde solvers. *arXiv preprint arXiv:2408.01536*, 2024.

Ajay Nadig, Akshaya Thoutam, Madeline Hughes, Anay Gupta, Andrew W Navia, Nicolo Fusi, Srivatsan Raghavan, Peter S Winter, Ava P Amini, and Lorin Crawford. Consequences of training data composition for deep learning models in single-cell biology. *bioRxiv*, pages 2025–02, 2025.

Ruben Ohana, Michael McCabe, Lucas Meyer, Rudy Morel, Fruzsina Agocs, Miguel Beneitez, Marsha Berger, Blakesly Burkhart, Stuart Dalziel, Drummond Fielding, et al. The well: a large-scale collection of diverse physics simulations for machine learning. *Advances in Neural Information Processing Systems*, 37:44989–45037, 2024.

Jaideep Pathak, Shashank Subramanian, Peter Harrington, Sanjeev Raja, Ashesh Chattopadhyay, Morteza Mardani, Thorsten Kurth, David Hall, Zongyi Li, Kamyar Azizzadenesheli, Pedram Hassanzadeh, Karthik Kashinath, and Animashree Anandkumar. Fourcastnet: A global data-driven high-resolution weather model using adaptive fourier neural operators, 2022. URL https://arxiv.org/abs/2202.11214.

Maziar Raissi, Paris Perdikaris, and George E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

Bogdan Raonic, Roberto Molinaro, Tim De Ryck, Tobias Rohner, Francesca Bartolucci, Rima Alaifari, Siddhartha Mishra, and Emmanuel de Bézenac. Convolutional neural operators for robust and accurate learning of pdes. *Advances in Neural Information Processing Systems*, 36:77187–77200, 2023.

Michael Rotman, Amit Dekel, Ran Ilan Ber, Lior Wolf, and Yaron Oz. Semi-supervised learning of partial differential operators and dynamical flows. In *Proceedings of the Thirty-Ninth Conference on Uncertainty in Artificial Intelligence*, 2023.

Ricardo Buitrago Ruiz, Tanya Marwah, Albert Gu, and Andrej Risteski. On the benefits of memory for modeling time-dependent pdes. *arXiv preprint arXiv:2409.02313*, 2024.

Junhong Shen, Tanya Marwah, and Ameet Talwalkar. Ups: Towards foundation models for pde solving via cross-modal adaptation. *arXiv e-prints*, pages arXiv–2403, 2024.

Zhiqing Sun, Longhui Yu, Yikang Shen, Weiyang Liu, Yiming Yang, Sean Welleck, and Chuang Gan. Easy-to-hard generalization: Scalable alignment beyond human supervision. *arXiv preprint arXiv:2403.09472*, 2024.

Makoto Takamoto, Timothy Praditia, Raphael Leiteritz, and Others. Pdebench: An extensive benchmark for scientific machine learning. *NeurIPS Datasets and Benchmarks*, 2022.

Ronak Tali, Ali Rabeh, Cheng-Hau Yang, Mehdi Shadkhah, Samundra Karki, Abhisek Upadhyaya, Suriya Dhakshinamoorthy, Marjan Saadati, Soumik Sarkar, Adarsh Krishnamurthy, Chinmay Hegde, Aditya Balu, and Baskar Ganapathysubramanian. Flowbench: A large scale benchmark for flow simulation over complex geometries, 2024. URL https://arxiv.org/abs/2409.18032.

Anh Tran, Alexander Mathews, Lexing Xie, and Cheng Soon Ong. Factorized fourier neural operators. *arXiv preprint arXiv:2111.13802*, 2021. URL https://arxiv.org/abs/2111.13802.

Henry G Weller, Gavin Tabor, Hrvoje Jasak, and Christer Fureby. A tensorial approach to computational continuum mechanics using object-oriented techniques. *Computers in physics*, 12(6):620–631, 1998.

A Related work

Two leading paradigms for learning PDE solutions are physics-informed neural networks (PINNs) and neural operators. PINNs embed PDE residuals and boundary conditions into the loss, enabling mesh-free training, strong use of physics priors, and efficacy in small-data forward/inverse settings (Raissi et al., 2019; Karniadakis et al., 2021). At the same time, training can involve challenging multi-term loss balancing and optimization stiffness (Krishnapriyan et al., 2021), with slower convergence on multi-scale or chaotic regimes (e.g., high Re turbulence) and sensitivity to complex geometries or boundary conditions (Karniadakis et al., 2021). Neural operators (e.g., DeepONet, FNO) learn mappings between function spaces, providing amortized inference, cross-discretization/geometry generalization, and scalability via pretraining on synthetic data (Lu et al., 2021b; Li et al., 2021b; Kovachki et al., 2023). Their performance, however, typically depends on substantial supervised datasets; robustness may be reduced under distribution shift across physics/geometry, and accuracy near shocks/discontinuities or conservation/stability guarantees may require additional structure and memory (Kovachki et al., 2023; Brandstetter et al., 2022; Ruiz et al., 2024). Geometry-aware operator variants (e.g., GeoFNO) enhance robustness on irregular domains yet still rely on curated simulation corpora (Li et al., 2023). Recent surveys synthesize these properties across PDE tasks, including turbulent flows (Karniadakis et al., 2021; Brunton and Kutz, 2020).

Unlike these efforts, we focus on the data itself, specifically on how generating better quality data may improve performance. There exists some work in the sciences on augmenting scarce experimental datasets with abundant simulated data from simplified systems, e.g. Hoffmann et al. (2019) demonstrated that combining simulated flat-folding patterns with limited experimental data enabled machine learning models to recover structure in crumpled sheets. Similar studies are being conducted in biology, where recent work investigates the effects of training data composition on the performance of foundation models for single-cell genomics (Nadig et al., 2025). We view our contribution as a more systematic study of how to generate and make use of data of different qualities.

Outside of PDEs, data difficulty has also been explored in other areas such as language modeling. For example, a major difficulty axis in natural language processing is context length, with several explorations of how to train models capable of solving long-context tasks without resorting to purely long-context training (Anil et al., 2022; Cho et al., 2024). Separately, our work is also related to work on *easy-to-hard generalization* in arithmetic reasoning tasks (Sun et al., 2024; Hase et al., 2024) Here, the past work has found that appropriately training the models on simpler tasks—for example simpler math problems—leads to better performance on harder tasks. Here a task's hardness is determined according to some human hardness measures, e.g. grade-level for STEM problems.

B Dataset Generation and Simulation Setup

We generate two major datasets—Flow Past Object (FPO) and Lid-Driven Cavity (LDC)—to investigate the impact of domain complexity on the performance and generalization of neural PDE solvers. Each dataset contains three levels of difficulty: *easy* (no obstacles), *medium* (a single obstacle), and *hard* (2–10 randomly placed obstacles). All simulations are run using `OpenFOAM`, a finite-volume CFD solver.

B.1 FPO Domain

In the FPO setting, we simulate flow around one or more square obstacles within a 2×2 m rectangular domain using the `icoFoam` solver. The left boundary is treated as a velocity inlet, where we impose a parabolic inflow profile representative of fully developed laminar channel flow. The right boundary is set as a pressure outlet with fixed value, and the top and bottom boundaries are treated as no-slip walls.

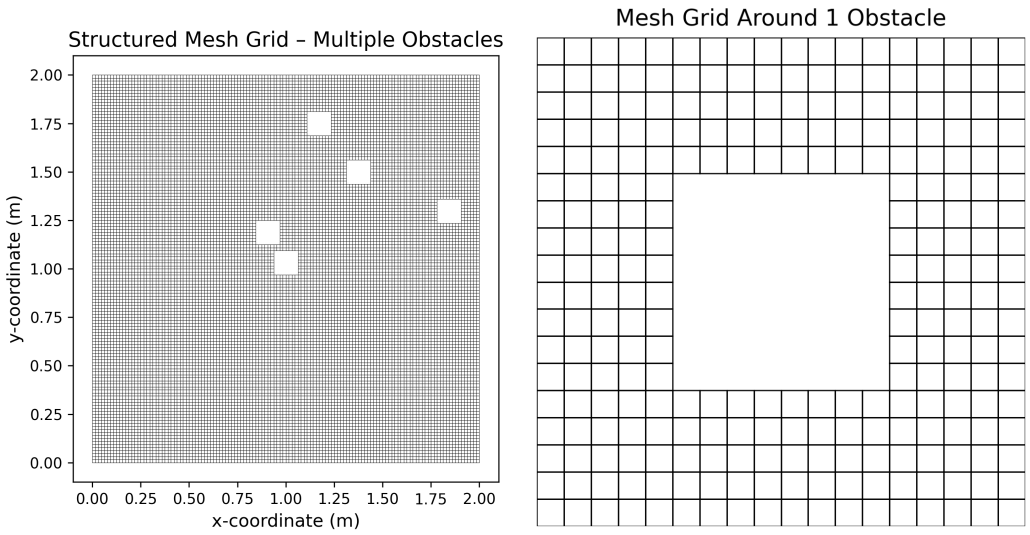


Figure 12: Structured mesh used in our multi obstacle setup. **Left**: full 2×2 m mesh domain. **Right**: zoom-in around one obstacle.

To define the parabolic inlet condition, we prescribe the horizontal velocity component $u(y)$ across the height $H = 2$ m of the domain using the analytical profile for plane Poiseuille flow:

$$u(y) = 4U_{\max} \cdot \frac{y(H-y)}{H^2}, \quad y \in [0, H],$$

where $U_{\max}$ is the peak velocity occurring at the vertical midline ($y = H/2$). This ensures zero velocity at the top and bottom walls ($y = 0, H$) and a smooth parabolic profile across the inlet face.

Reynolds numbers are sampled from a truncated normal distribution $\mathcal{N}(5000, 2000^2)$ with support in $[100, 10000]$, and the corresponding $U_{\max}$ is scaled to satisfy:

$$\mathrm{Re} = \frac{U_{\text{avg}} \cdot L}{\nu}, \quad \text{with } U_{\text{avg}} = \frac{2}{3}U_{\max}, \quad L = 2 \text{ m}, \quad \nu = 1.5 \times 10^{-5} \text{ m}^2/\text{s},$$

where U_{avg} is the mean velocity of the parabolic profile. Solving for $U_{\max}$ ensures consistency between the desired Reynolds number and the imposed inlet condition.

Obstacle configurations are generated by randomly placing between 2 and 10 square holes in the domain, using a rejection sampling algorithm to prevent overlap or boundary collision. For each simulation:

- The geometry is procedurally constructed by modifying `blockMeshDict`, and mesh generation is handled via OpenFOAM's native utilities.
- The simulation duration is dynamically adjusted based on the sampled Reynolds number using a characteristic time scale, and outputs are recorded at 20 evenly spaced intervals.
- Velocity and pressure fields are post-processed using OpenFOAM utilities and interpolated onto a 128×128 uniform grid via barycentric interpolation.

B.2 LDC Domain

In the LDC setting, fluid flows in a closed 2×2 m cavity with a moving top wall. We again use the `icoFoam` solver with zero velocity on side and bottom walls and a parabolic profile imposed on the top wall. The top-wall velocity is scaled to match a target Reynolds number:

$$U_{\max} = \frac{\mathrm{Re} \cdot \nu}{L}, \quad \text{with } \nu = 1.5 \times 10^{-5} \text{ m}^2/\text{s}, L = 2 \text{ m}.$$

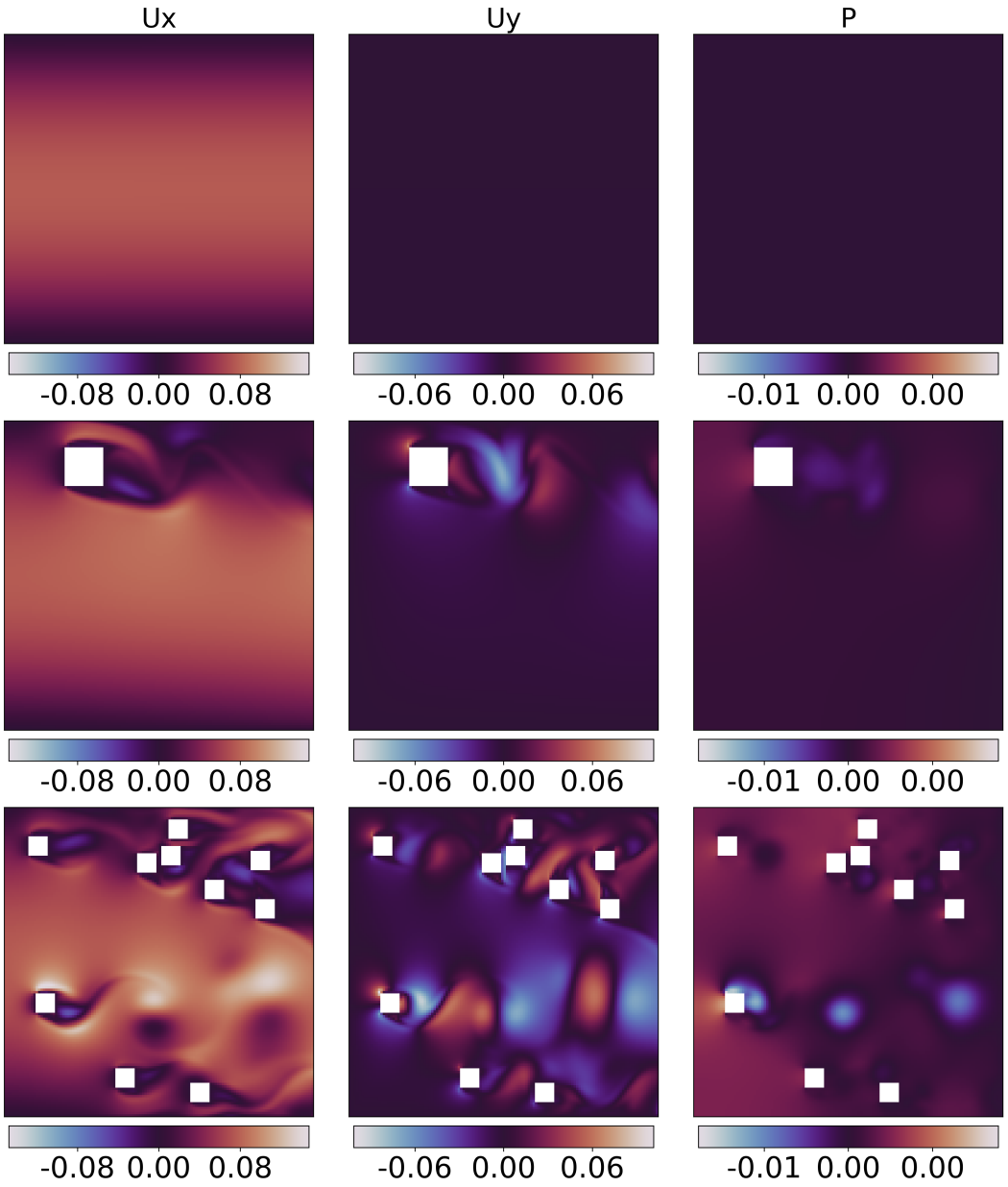


Figure 13: FPO Flow Fields (Velocity). Top row shows velocity and pressure fields for easy, medium, and complex setups; White areas represent the square obstacles (holes) in the domain.

B.3 Flow Development Scheduling.

To ensure that each simulation reaches a fully developed state before data is recorded, we adaptively determine the simulation end time based on the sampled Reynolds number Re. This is critical in both the FPO and LDC domains, where flow transients can vary significantly with Re, and premature truncation would lead to incomplete or biased solution fields.

We define a piecewise scheduling rule that maps the Reynolds number to a simulation end time T_{end} via either a linear scaling or a constant duration, depending on the flow regime.

For moderate to high Re values, we employ the characteristic viscous diffusion time scale:

$$t_{\text{nd}} = \frac{L^2}{\nu \cdot \text{Re}},$$

where $L = 2$ m is the characteristic length of the domain, and $\nu = 1.5 \times 10^{-5}$ m^2/s is the kinematic viscosity of the fluid. The total simulation time is then computed as:

$$T_{\text{end}} = \gamma \cdot t_{\text{nd}},$$

where γ is a multiplicative factor that increases with Re to accommodate longer transients associated with inertial effects. For very low Reynolds numbers (Re < 100), where steady-state is approached slowly and viscous effects dominate, we assign a fixed total duration of 2700 seconds.

The multiplicative factors γ are manually calibrated for different Re ranges as shown below:

Re Range	γ	T_{end} Formula
5000–10000	40	$T_{\text{end}} = 40 \cdot t_{\text{nd}}$
4000–5000	30	$T_{\text{end}} = 30 \cdot t_{\text{nd}}$
2500–4000	20	$T_{\text{end}} = 20 \cdot t_{\text{nd}}$
1000–2500	10	$T_{\text{end}} = 10 \cdot t_{\text{nd}}$
500–1000	5	$T_{\text{end}} = 5 \cdot t_{\text{nd}}$
400–500	4	$T_{\text{end}} = 4 \cdot t_{\text{nd}}$
300–400	3	$T_{\text{end}} = 3 \cdot t_{\text{nd}}$
200–300	2	$T_{\text{end}} = 2 \cdot t_{\text{nd}}$
100–200	1	$T_{\text{end}} = 1 \cdot t_{\text{nd}}$
10–100	–	$T_{\text{end}} = 2700$ s (fixed)

Table 1: Reynolds-number-dependent scheduling of simulation end time.

The computed T_{end} is rounded up to the nearest hundred and used to configure the `controlDict` file for each simulation. The write interval is also dynamically selected to yield 20 evenly spaced output frames, ensuring consistent temporal sampling across all Reynolds number regimes. This scheduling mechanism guarantees physically meaningful and temporally aligned datasets, while avoiding wasted computation for low-Re flows or premature termination for higher-Re flows.

B.4 Numerical Discretization and Solver Configuration

All simulations are performed using OpenFOAM v2406 to numerically solve the incompressible Navier–Stokes equations. To ensure stable and accurate data generation across diverse Reynolds numbers and geometric complexities, we adopt a consistent finite-volume setup for time integration, spatial discretization, and linear solver configurations.

Governing Equations. We solve the incompressible Navier–Stokes system in the exact form used in the main paper:

$$\partial_t \mathbf{u} + (\mathbf{u} \cdot \nabla)\mathbf{u} + \nabla p = \nu \Delta \mathbf{u} \qquad \text{and} \qquad \nabla \cdot \mathbf{u} = 0 \tag{3}$$

where $\mathbf{u}(\mathbf{x}, t)$ is the velocity field, p is the (kinematic) pressure, and ν is the kinematic viscosity.

B.4.1 Time Discretization

To maintain numerical robustness at higher Reynolds numbers and small time steps, we discretize the temporal derivative using a first-order implicit backward-Euler scheme:

$$\frac{\partial \mathbf{U}}{\partial t} \approx \frac{\mathbf{U}^{n+1} - \mathbf{U}^n}{\Delta t}.$$

This choice offers unconditional stability and aligns with OpenFOAM's standard transient solvers.

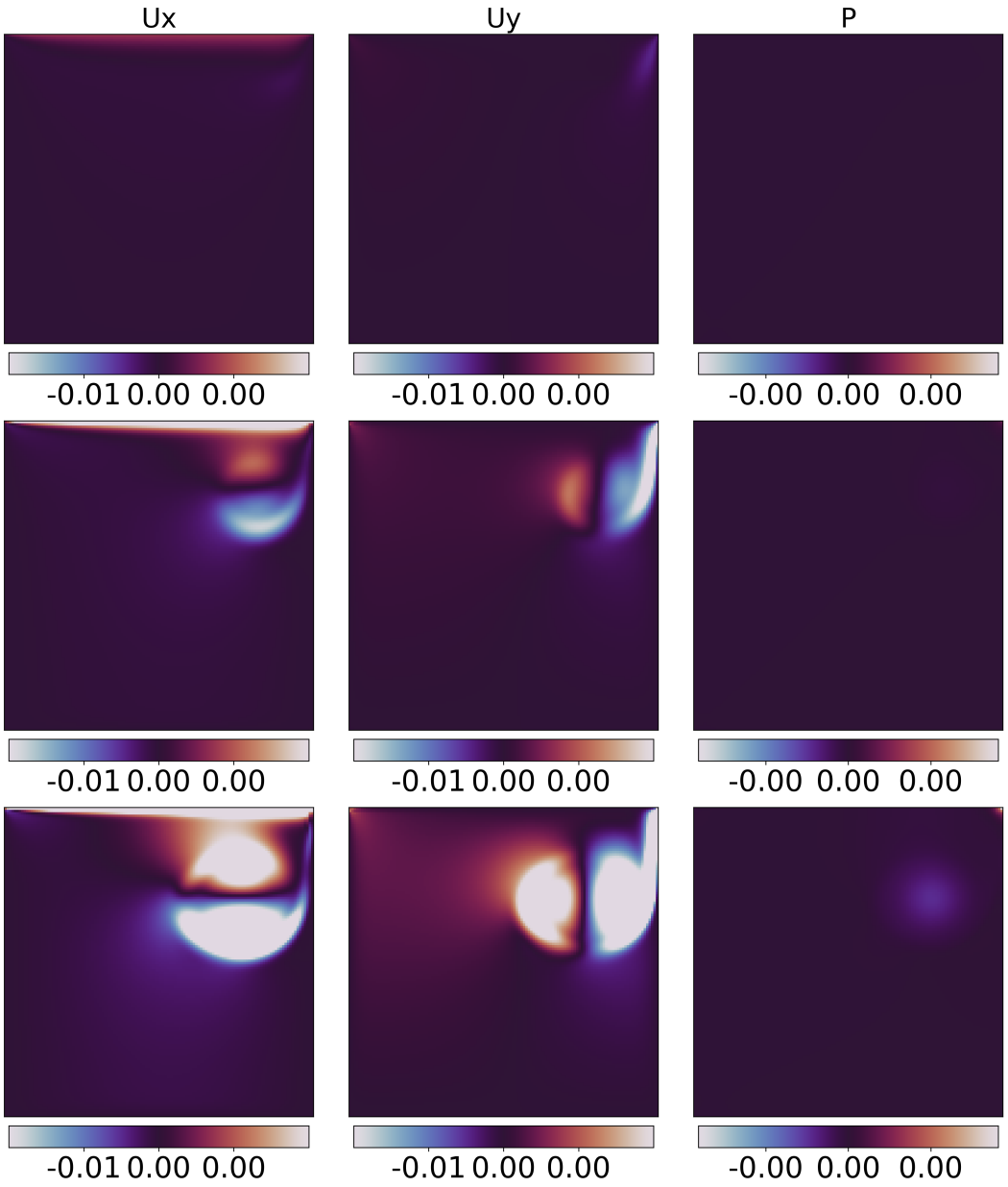


Figure 14: LDC Flow Fields (Velocity). Top row: velocity and pressure fields for easy, medium, and complex setups.

B.4.2 Spatial Derivatives

All spatial derivatives are evaluated using the finite-volume method with second-order accurate schemes:

- **Gradient terms** such as ∇p and $\nabla \mathbf{U}$ use central differencing:

$$\nabla \phi \approx \texttt{Gauss linear},$$

  which preserves smooth fields with low numerical diffusion.
- **Convective fluxes**, dominant at higher Reynolds numbers, use an upwind-biased linear scheme with gradient reconstruction:

$$\nabla \cdot (\phi \mathbf{U}) \approx \texttt{Gauss linearUpwind grad(U)},$$

balancing stability with second-order accuracy, especially near obstacles where steep gradients occur.

- **Diffusive terms** (Laplacians) use:

$$\nabla^2\phi \approx \texttt{Gauss linear orthogonal},$$

appropriate for our structured Cartesian grids.[1]

B.4.3 Interpolation and Surface Gradients

Cell-face values are interpolated linearly:

$$\phi_f \approx \texttt{linear}(\phi),$$

and surface-normal gradients use the `orthogonal` scheme, leveraging the grid's structured nature.

B.4.4 Linear Solvers

The momentum and pressure equations are solved using efficient iterative solvers:

- **Pressure** (p): PCG (Preconditioned Conjugate Gradient) with DIC (Diagonal-based Incomplete Cholesky) preconditioning.
- **Velocity** ($\mathbf{U}$): `smoothSolver` with symmetric Gauss–Seidel smoothing.

Per-equation tolerances are:

$$\begin{aligned} \text{Pressure:} \quad & \texttt{tolerance} = 10^{-6}, \quad \texttt{relTol} = 0.05 \text{ (final: 0)} \\ \text{Velocity:} \quad & \texttt{tolerance} = 10^{-5}, \quad \texttt{relTol} = 0 \text{ (final)} \end{aligned}$$

B.4.5 Design Motivation

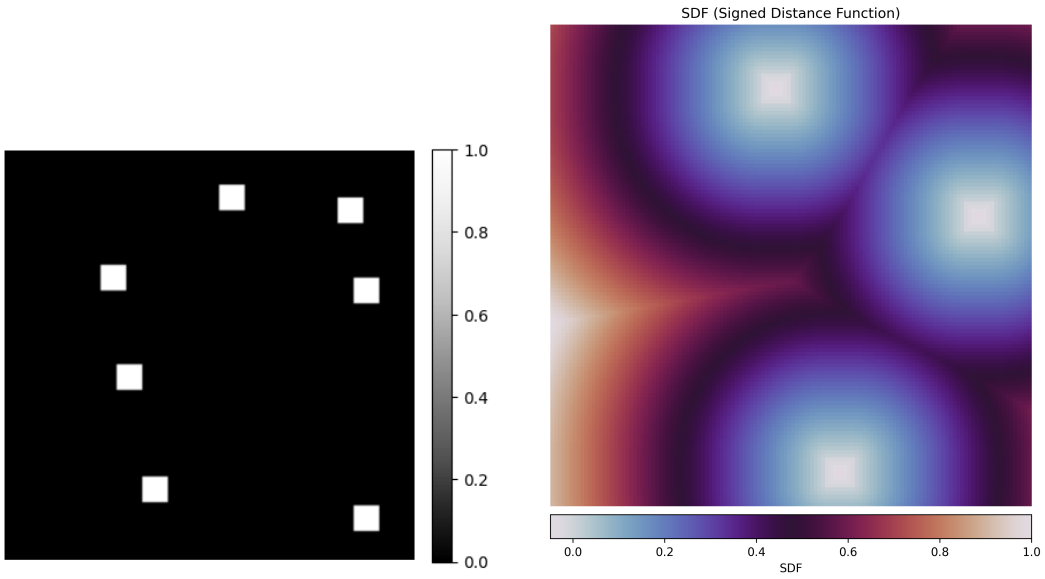
This configuration follows established best practices in the OpenFOAM ecosystem and prior simulation-driven ML benchmarks, ensuring numerical stability and physical realism across a wide range of Reynolds numbers. We adopt `Gauss linear` for gradients and diffusive terms to preserve smoothness on structured grids Weller et al. (1998), and `linearUpwind grad(U)` to balance accuracy and robustness in the presence of sharp gradients and internal obstacles Jasak (1996). The backward-Euler time integration and implicit solvers align with standard OpenFOAM settings for incompressible flows and are widely used in both industrial and academic studies Jasak et al. (2007).

B.4.6 Simulation Pipeline (Transient)

We automate data generation via modular Python scripts for both FPO and LDC:

1. **Domain Construction:** Randomized obstacle positions are sampled; a mesh is constructed via a modified `blockMeshDict`.
2. **Velocity and Controls:** Boundary velocity profiles and run duration are computed from the sampled Reynolds number.
3. **Simulation Execution:** The case is solved using `icoFoam`; fields are written at fixed intervals to yield 20 timesteps.
4. **Postprocessing:** Velocity and pressure fields are parsed and projected onto a 128×128 regular grid.
5. **Geometry Encoding:** Each grid cell includes a binary mask (fluid vs. obstacle) and a signed distance field (SDF) via an Euclidean distance transform.

[1] If mild non-orthogonality appears, `Gauss linear corrected` is a safe alternative.



(a) Binary mask: white = hole, black = fluid.

(b) Signed distance field (SDF) corresponding to the mask.

Figure 15: Geometry encodings. Visualization of the binary mask and corresponding SDF used to encode obstacle geometry.

B.4.7 Randomized Obstacle Generation

We construct domains with multiple internal holes via `blockMeshDict`:

- *Random Hole Placement:* Sample $n \in [2, 10]$ axis-aligned rectangles $\{x, y, w, h\}$ strictly within $[0, 2] \times [0, 2]$ (optionally enforcing non-overlap).
- *Block Decomposition:* Subdivide a structured Cartesian grid; cells lying entirely inside holes are removed. Faces adjoining missing cells become boundary patches `hole1`, `hole2`, ...
- *Boundary Patches:* The outer walls (including the moving lid for LDC) and the hole patches are set as no-slip walls ($\mathbf{U} = \mathbf{0}$) unless the experiment specifies inlets/outlets (FPO).

B.4.8 Boundary Conditions and Reynolds Number

For LDC, the top-wall velocity U_{lid} is set to match a target Reynolds number:

$$\text{Re} = \frac{U_{\text{lid}}\, L}{\nu} \quad \Longrightarrow \quad U_{\text{lid}} = \frac{\text{Re}\, \nu}{L}.$$

For FPO, inlet speed is set analogously; outlets use zero-gradient pressure and velocity conditions consistent with standard setups.

B.4.9 Data Format

Each trajectory is stored as a NumPy array with shape `(20, 128, 128, 6)`, containing six channels: horizontal velocity u, vertical velocity v, pressure p, normalized Reynolds number $\widehat{\text{Re}}$, binary mask, and SDF. A representative visualization is shown in Figure 15.

C Model Architectures

In this section, we provide implementation and training details for the neural operator models evaluated in our study. We focus on two classes of models: (1) the Convolutional Neural

Operator (CNO) (Raonic et al., 2023), which is trained from scratch, and (2) Poseidon-T (Herde et al., 2024), a pretrained transformer-based model fine-tuned on our downstream task. These models differ significantly in their architectural design, parameterization, and training strategy.

All models are trained to predict velocity and pressure fields for 2D incompressible Navier-Stokes simulations. Given an input-output pair (u_t, u_{t+1}), where $u_t \in \mathbb{R}^{C\times H\times W}$ denotes the flow variables at timestep t and C is the number of channels, models are trained to minimize the relative ℓ_1 loss:

$$\mathcal{L}(u_t, u_{t+1}) = \frac{\|u_{t+1} - \hat{u}_{t+1}\|_1}{\|u_{t+1}\|_1 + \epsilon}, \tag{4}$$

where $\hat{u}_{t+1}$ is the model prediction and $\epsilon = 10^{-10}$ ensures numerical stability.

C.1 Convolutional Neural Operator (CNO)

The Convolutional Neural Operator (CNO) used in our experiments is based on the architecture proposed by Raonic et al. (2023), with several adjustments tailored to our time-dependent Navier-Stokes flows. The model is implemented in PyTorch Lightning and trained in an autoregressive supervised setting.

The model is structured as an encoder-decoder network with residual blocks, optional attention in the bottleneck, and optional time-conditioning via instance normalization. Key components of the architecture include:

- **Encoder and Decoder:** The encoder has $N_{\text{layers}} = 4$ downsampling stages with a channel multiplier of 32. The decoder mirrors this with corresponding upsampling layers. Each stage consists of a combination of convolutional and spectral convolution blocks.
- **Residual Blocks:** The encoder and bottleneck include $N_{\text{res}} = 8$ residual blocks per level and $N_{\text{res\_neck}} = 8$ blocks in the bottleneck.
- **Normalization:** We use instance normalization (specified via `nl_dim` $= [2, 3]$), conditional on the input timestep.
- **Time Conditioning:** The model is trained with `is_time` $= 1$ to incorporate the current timestep as an additional input feature.
- **Grid Resolution:** All experiments use a spatial resolution of 128×128.
- **Loss Function:** We use a normalized relative ℓ_1 loss, computed per-sample and averaged over the batch.
- **Autoregressive Training:** The CNO is trained in an autoregressive supervised manner using transitions of the form $t \to t{+}1$ only. We set `allowed` $=$ `'one'` in the training configuration, restricting the training to adjacent timestep pairs.
- **Other Details:** Training used a batch size of 32, learning rate of 7.5×10^{-4}, step learning-rate schedule over **400 epochs**, and weight decay of 1×10^{-6}.

0.00075 The model contains approximately 18 million parameters and does not utilize attention in our setup.

C.2 Factorized Fourier Neural Operator (F-FNO)

The Factorized Fourier Neural Operator (F-FNO) used in our experiments is based on the architecture proposed by Tran et al. (2021), designed for efficient learned simulation of PDEs. The model is implemented in PyTorch Lightning and trained in an autoregressive one-step prediction setting. It consists of a deep sequence of Fourier operator layers with factorized spectral convolutions and improved residual connections, which allow stable training even at greater network depths than the original FNO . Key components of the architecture include:

- **Network Depth and Parameters:** We deploy a 5-layer F-FNO, following the high-capacity configuration from the original paper.

. This is an order of magnitude fewer parameters than a comparable geo-FNO model, despite the increased depth, due to the factorized kernel representation.

- **Spectral Layers:** Each layer applies a separable Fourier convolution on the input features, factorizing the transform over each spatial dimension . In practice, we drop half of the higher-frequency Fourier modes in each layer to reduce computational cost (e.g., on a 64×64 grid we keep only the top 16 modes per dimension) . The retained frequency components serve as learned global convolution kernels applied via inverse FFT.
- **Feedforward Block:** After the spectral convolution, each layer includes a two-layer feed-forward network (pointwise MLP) operating in the physical space . We use ReLU activations in these feed-forward layers . This pointwise MLP mixes features per grid location and is analogous to the transformer's MLP block, providing non-linear coupling of the channels.
- **Residual Connections:** A skip connection adds each layer's input to its output after the non-linear feed-forward stage . This post-activation residual design preserves more of the original signal and stabilizes gradient flow in deep stacks , enabling the F-FNO to converge even with dozens of layers (where the original FNO would diverge at ≥ 12 layers).
- **Coordinate Encoding:** Following Tran et al. (2021), we augment the input with explicit spatial coordinate channels. Each input field is concatenated with its normalized x and y coordinates (as two additional channels), providing a positional encoding that consistently improves accuracy . The Fourier layers inherently utilize absolute positions (through the grid indices in the transform), while the feed-forward layers benefit from the coordinate features to capture location-dependent effects .
- **Autoregressive Training:** We train the F-FNO in a one-step-ahead prediction manner. The model uses only the current state as input to predict the next state, enforcing a first-order Markov assumption (no multi-step history) . We employ teacher forcing during training, i.e. at each training step the ground-truth state at time t is provided as input to predict $t+1$. This approach avoids the need to unroll long sequences during training and was found to improve stability and accuracy.
- **Input Normalization and Noise:** We apply per-variable normalization to input fields and add a small Gaussian noise perturbation during training . These techniques, recommended by **?**, act as regularization and help prevent training instabilities (we observed that without the added noise, the model's validation loss could sometimes spike early in training).
- **Training Setup:** The F-FNO is trained with a batch size of 16, using a learning rate of 5×10^{-5} and a cosine annealing schedule (no restarts) over **400 epochs**, along with a weight decay of 1×10^{-6}. These hyperparameters match those used for our other models to ensure a fair comparison. We did not employ the optional weight-sharing of Fourier weights across layers in our configuration, as it has minimal impact on performance at this depth .

C.3 Poseidon-T

We evaluate Poseidon-T using the pretrained model checkpoints provided by Herde et al. (2024), available on Hugging Face.[2] We perform fine-tuning on our custom datasets without any further pretraining.

The architecture follows a SwinUNet-based transformer backbone with hierarchical attention and patch embeddings. We retain the pretrained model structure and only update weights via supervised autoregressive finetuning. Key configuration details include:

- **Backbone:** SwinUNet with hierarchical attention and window-based self-attention.
- **Variant:** We use Poseidon-**T**, which has a base embedding dimension of 48, depths $[4, 4, 4, 4]$, and patch size 4.

[2] https://huggingface.co/camlab-ethz/Poseidon-T

- **Resolution:** All inputs are processed at 128×128 resolution.
- **Training Setup:** Fine-tuning is performed for 100 epochs with batch size 16, weight decay of 1×10^{-6}, and cosine learning rate schedule starting from 5×10^{-5}.
- **Loss Function:** We use a normalized relative ℓ_1 loss, computed per-sample and averaged over the batch.

Only the decoder and time-conditioning layers are updated during fine-tuning. The rest of the model remains unchanged from the pretrained checkpoint.

C.4 Poseidon-B

We evaluate Poseidon-B using the pretrained model checkpoint provided by Herde et al. (2024), available on Hugging Face.[3] We perform fine-tuning on our custom datasets without any further pretraining. The architecture mirrors Poseidon-T's setup, following a SwinUNet-based transformer backbone with hierarchical (U-Net style) multiscale attention and window-based self-attention. We retain the pretrained model structure and update weights via supervised autoregressive fine-tuning. Key configuration details include:

- **Backbone:** SwinUNet with hierarchical attention (patch merging/expansion) and windowed self-attention.
- **Variant:** Poseidon-**B**, base embedding dimension 96, depths [8, 8, 8, 8] (eight SwinV2 transformer blocks per level), patch size 4.
- **Resolution:** All inputs are processed at 128×128 resolution (matching the pretraining grid size).[4]
- **Training setup:** 100 epochs, batch size 16, weight decay 1×10^{-6}, cosine learning-rate schedule starting from 5×10^{-5}.
- **Loss:** Normalized relative ℓ_1 loss, computed per-sample and averaged over the batch.
- **Fine-tuned parameters:** Only the decoder and time-conditioning layers are updated; all other weights remain frozen from the pretrained checkpoint.

C.5 Poseidon-L

We evaluate Poseidon-L using the pretrained model checkpoint provided by Herde et al. (2024), available on Hugging Face.[5] We fine-tune this model on our custom datasets with no additional pretraining. The architecture is identical to the other Poseidon variants, employing the same SwinUNet-style transformer backbone with hierarchical multiscale attention and window-based (shifted-window) self-attention. We preserve the original model architecture and learn weights via supervised autoregressive fine-tuning. Key configuration details include:

- **Backbone:** SwinUNet with hierarchical attention and window-based self-attention (shifted-window mechanism).
- **Variant:** Poseidon-**L**, base embedding dimension 192, depths [8, 8, 8, 8] (eight SwinV2 transformer blocks at each level), patch size 4.
- **Resolution:** All inputs are handled at 128×128 resolution, consistent with the model's pretraining grid size.[6]
- **Training setup:** 100 epochs, batch size 16, weight decay 1×10^{-6}, cosine learning-rate schedule starting from 5×10^{-5}.
- **Loss:** Normalized relative ℓ_1 loss, computed per-sample and averaged over the batch.

[3] https://huggingface.co/camlab-ethz/Poseidon-B
[4] https://huggingface.co/camlab-ethz
[5] https://huggingface.co/camlab-ethz/Poseidon-L
[6] https://huggingface.co/camlab-ethz

- **Fine-tuned parameters:** Only the decoder and time-conditioning (time-encoded layer normalization) layers are updated during fine-tuning; all other weights are frozen to their pretrained values.

C.6 Computational Resources

We report compute details and training runtimes for the experiments conducted in this study. All models were trained on a single GPU using PyTorch with SLURM-based scheduling. The Convolutional Neural Operator (CNO) models were trained from scratch for 400 epochs, while Poseidon-T/B/L models were fine-tuned for 200 epochs from publicly released checkpoints.

CNO models were trained from scratch while Poseidon models were fine-tuned on a single NVIDIA L40S GPU on the Babel cluster. For autoregressive training with $t \rightarrow t{+}1$ supervision, training time scaled approximately with the number of trajectories. Table 2 summarizes the approximate training time for varying dataset sizes. For completeness, we include FFNO (trained from scratch) and additional Poseidon variants (B and L) alongside Poseidon-T. FFNO and CNO were trained from scratch; all Poseidon variants (T/B/L) were fine-tuned.

Table 2: Approximate training durations with increasing number of training trajectories on a single NVIDIA L40S GPU. FFNO and CNO trained from scratch; Poseidon variants fine-tuned.

Training Trajectories	CNO	FFNO	Poseidon-T	Poseidon-B	Poseidon-L
200	3h 30m	1h	2h	6h 40m	–
400	6h 48m	7h	3h	11h 30m	–
800	7h 55m	1d 13h	9h	19h	1d 3h
1600	1d 4h	1d 21h	18h	1d 16h	–

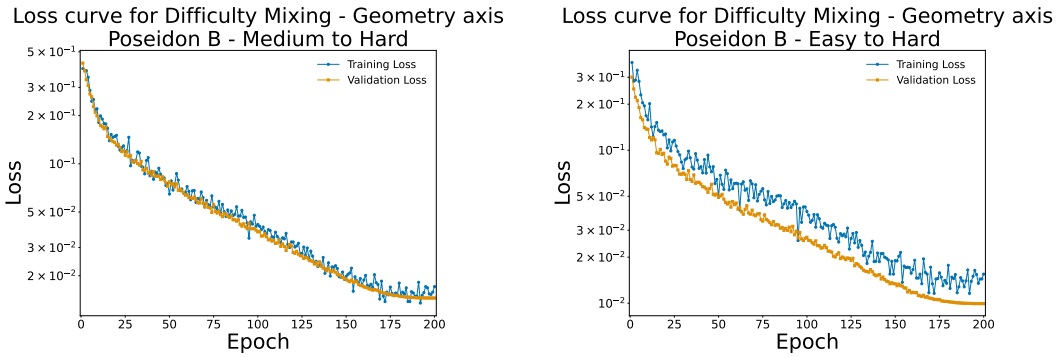
All experiments were run using a single GPU with no mixed precision or distributed training.

C.7 Training Convergence Analysis

To provide additional insight into the training dynamics of our models across different difficulty mixing scenarios, we present the convergence behavior during training and validation in Figure 16. The training loss is computed as the mean L1 loss over the training set, which consists of samples from both easy/medium and hard data according to the difficulty ratio of each experiment. The validation loss is computed as the mean L1 loss over a fixed validation set of 100 samples per experiment: 50 samples drawn from the easy/medium distribution and 50 samples from the hard distribution. Figure 16 shows the training and validation loss curves for Poseidon-B across two difficulty mixing scenarios: medium-to-hard and easy-to-hard geometry composition. Both subplots demonstrate smooth convergence and stable validation loss across different mixing ratios, reflecting the robustness of our training procedure and validating the generalization properties of models trained on lower-difficulty data augmented with target-difficulty examples. It is important to note that the validation loss shown here uses a balanced split (50 easy/medium, 50 hard samples) to monitor training stability across difficulty compositions, whereas the test performance numbers reported in the main paper evaluate on the complete hard dataset to assess generalization to the target distribution.

C.8 Additional Results: FlowBench Harmonics Analysis

To further validate the applicability of our findings beyond the primary FPO and LDC domains, we examine the performance on the FlowBench dataset using Harmonics geometries. Similar to the NURBS results presented in the main paper, we augment the target Harmonics examples with zero-obstacle FPO (easy) and single-obstacle FPO (medium) data. Figure C.8 presents the cost versus error scaling behavior for models trained on 100 target Harmonics examples augmented with varying data generation costs from lower-difficulty FPO simulations. Consistent with our observations on NURBS geometries and our primary difficulty-mixing experiments, adding lower-difficulty examples substantially reduces error across multiple



(a) Medium-to-hard geometry. **(b)** Easy-to-hard geometry.

Figure 16: Training convergence for Poseidon-B across difficulty mixing scenarios. Both subplots show training and validation loss (L1 loss) over epochs for models trained with varying fractions of easy/medium and hard examples. Training loss is computed over the full training set (with mixed difficulty ratio), while validation loss is computed over a fixed set of 100 samples (50 from easy/medium, 50 from hard). Convergence is smooth and stable, demonstrating the robustness of the difficulty mixing strategy.

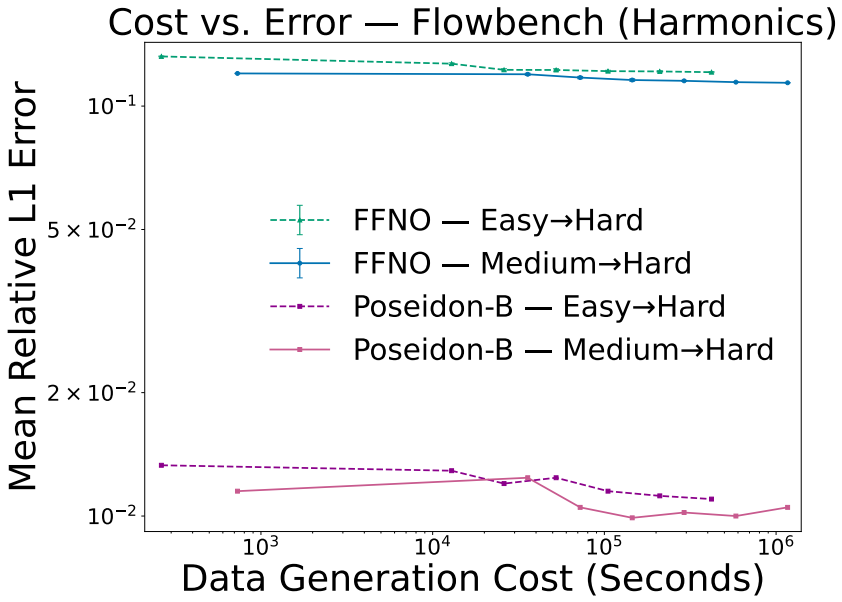


Figure 17: Performance on FlowBench's Harmonics FPO simulations when 100 target examples are augmented with zero-obstacle FPO (easy) or single-obstacle FPO (medium) simulations. Similar to NURBS results, data augmentation with medium-difficulty examples substantially improves performance for most models on the target Harmonics distribution.

model architectures. This result demonstrates that difficulty mixing is an effective and generalizable strategy for improving performance on challenging datasets, extending the applicability of our approach across diverse geometric families.