Breakeven complexity: A new perspective on neural partial differential equation solvers

Yijing Zhang
University of Wisconsin–Madison
yzhang2637@wisc.edu

Nicholas Roberts*
Princeton University
nick11roberts@princeton.edu

Tanya Marwah†
Google DeepMind
tmarwah@google.com

Mikhail Khodak
University of Wisconsin–Madison
khodak@wisc.edu

Abstract

Neural surrogate solvers of partial differential equations (PDEs) promise dramatic speedups over numerical methods, especially in scenarios requiring many solves. However, current accuracy-based evaluations do not fully consider two central issues: (1) neural solvers incur substantial up-front costs for data generation, training, and tuning; and (2) classical solvers can also generate low-fidelity solutions at a sufficiently low simulation cost. To explicitly account for these realities and fully incorporate end-to-end costs, we propose an evaluation framework centered on **breakeven complexity**,‡ a metric that counts the forward solves before a learned solver is cost-effective relative to an error-equivalent traditional solver. To evaluate this measure, we apply scaling laws to determine how much training budget to allocate to data generation and discuss how to achieve smooth error-matching in diverse settings. We evaluate the breakeven complexity of multiple neural PDE solvers on three PDEs on 2D periodic domains from APEBench and a novel benchmark of flows past multiple obstacles generated by the GPU-native PyFR code. Among other findings, our results suggest that neural PDE solvers become *more effective* as problems get harder in terms of cost, dimension, rollout, physics regime (e.g. higher Reynolds number), etc.

1 Introduction

PDE simulation is a crucial engineering tool that enables prediction and analysis of physics systems in settings where experiments are expensive or impractical [7, 17]. Recently, advances in machine learning (ML) have driven the proliferation of data-driven surrogate models that can quickly solve transient PDEs using only neural network forward passes [18, 20]. This type of acceleration has significant potential impact across numerous large-scale scientific computing tasks.

However, while neural PDE surrogates often have fast inference when trained, they require substantial up-front cost, including data generation, model training and tuning. Furthermore, neural PDE surrogates are typically low-fidelity approximations, while classical simulators can trade-off accuracy and cost via tunable parameters (grid resolution, number of timesteps, etc.), thus also allowing for low-fidelity simulation at a reduced cost [3]. This creates a decision problem that accuracy alone does not address: when is it worthwhile to invest the up-front training cost of a surrogate, rather than simply running a numerical solver at an appropriately chosen lower fidelity?

*Work completed while at the University of Wisconsin–Madison.
†Work completed while at the Simons Foundation.
‡Code: github.com/yijingz02/breakeven_complexity. Data: huggingface.co/datasets/yijingz/breakeven_complexity.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

In some settings, such as real-time inference, reducing latency is often the main concern, motivating surrogate deployment even with nontrivial training costs so long as the surrogate is faster than a classical solver of similar error [29, 3]. We instead are motivated by cases where repeated forward PDE solves are required to fit latent quantities or optimize objectives, as in PDE-constrained inverse, control, and design problems [4]. For such non-real-time applications, the decision becomes fundamentally an amortization trade-off: will the surrogate solver be used enough times such that the compute spent on generating data and training the model is worth it, i.e. not better-spent on querying a traditional solver of similarly low fidelity? Existing evaluation practice for neural PDE surrogates, which mostly focuses on measuring the predictive error against a fixed high-fidelity ground-truth simulator, typically under-represents end-to-end costs, making it difficult to answer this question [27, 16, 23].

To address this gap, we propose a compute-aware evaluation framework that makes the decision trade-off explicit. Our approach is centered around **breakeven complexity**, which measures the number of forward solves required for a neural PDE surrogate to be cost-effective relative to an error-matched low-fidelity classical simulator. Unlike accuracy-only evaluations, breakeven complexity accounts for data generation costs, training costs, inference costs, and low fidelity in a single number. Reported jointly with error, breakeven complexity enables a more realistic comparison between learned models that better-informs decision-making and accurately conveys their potential relative to traditional solvers.

Our specific contributions are the following:

1. We build an evaluation framework around **breakeven complexity**, a cost-aware metric for learned PDE solvers building on the amortization quantity suggested by McGreivy and Hakim [21, Eq. 1]. We formulate its worst-case and average-case variants and make their computation practical by (a) using scaling laws to allocate compute between optimization and data generation at a fixed budget level and (b) showing how to smoothly vary the error while reducing resolution when finding an error-matched solver.
2. We deploy our methodology on three 2D periodic PDE settings from APEBench, solved via the pseudo-spectral Exponax code [16], and on a novel dataset called **BreakFlow** consisting of multi-obstacle flows simulated with the scale-resolving, GPU-native PyFR code [32]. We release multi-fidelity variants of all four datasets along with protocols for enabling the evaluation of breakeven complexity on these and on future benchmarks.
3. We evaluate the breakeven complexity of eight leading models (four supervised solvers and four foundation models) on the above benchmarks. Our investigation shows that models that do well according to reported error can sometimes underperform under our cost-aware metric, that it can be used to study the robustness of learned solvers, and that on the APEBench (periodic) tasks learned solvers can require hundreds of thousands of inference calls in order to be cost effective.
4. On the other hand, by studying breakeven complexity across multiple spatial dimensions, rollout lengths, and Reynolds numbers, we find that it consistently *decreases* as the task gets harder. This suggests that the cost-effectiveness of neural PDE solvers may *improve* with simulation difficulty.

2 Related Work

There are a wide variety of neural network-based approaches to solving PDEs, including physics-informed neural networks (PINNs) [24] and neural operators [18, 30]. Our work is centered around evaluating methods, such as the latter, which train networks on data generated by classical simulators; this field has witnessed a steady improvement in (error-based) performance [30, 6]. Most recently, foundation models have begun to be used in this field as well, promising to alleviate the upfront cost of data generation via transfer from massive pretraining data [13, 20].

These neural surrogates have been evaluated on a variety of benchmarks, largely via their predictive error relative to some high-fidelity numerical simulator. Benchmarks such as PDEBench [27], PDEArena [11], APEBench [16] and the Well [23] all provide large, ready-to-use datasets across different baseline PDEs with standardized metrics and baseline models. However, focusing on accuracy makes it hard to understand tradeoffs with cost, despite this being a crucial question in a field motivated largely by solver speed. Our work addresses this issue by comparing explicitly to an error-matched traditional solver to derive a single interpretable value.

We are not the first to consider varying-fidelity classical solvers in evaluating learned approaches, which has been done for PINNs [26, 10] and (more relevantly) for neural operators by McGreivy and Hakim [21]. The latter showed the importance of simultaneously accounting for cost and accuracy

and the need to compare to effective, well-suited, and problem-specific solvers; these critiques directly motivate our work to develop a cost-aware basis for evaluating neural PDE solvers. McGreivy and Hakim [21] also recommend the metric that in this work we make practical and call "breakeven complexity"; to our knowledge, the measure was not used by them nor in the subsequent literature.

In addition to the framework and empirical evaluations, we also develop a new flow-past-multiple-obstacles benchmark called **BreakFlow**. While several flow-past-object tasks exist [28, 5], ours crucially is generated via the GPU-native PyFR code [32], enabling a direct and fair comparison to neural network training and inference on the same device type.

3 Breakeven Complexity

We are interested in solving θ-parameterized PDEs of form $\partial_t u(t,x) = \mathcal{L}_\theta[u](t,x)$ with initial condition $u(0,x) = u_\theta(x)$ and boundary terms $\mathcal{B}_\theta[u|\partial\mathcal{X}](t,x) = 0$ over a domain $\mathcal{X} \ni x$, with design variable θ parameterizing the differential operator $\mathcal{L}_\theta$, the initial conditions u_θ, and the boundary conditions $\mathcal{B}_\theta$. The goal of a PDE solver is to output an approximate solution $\hat{u}_\theta$ over $\mathcal{X}$, with the approximation quality assessed either by comparing to a high-fidelity reference or via a PDE residual. Traditionally, transient PDEs have been solved by what we will refer to as "classical" solvers, which typically discretize in space and time and solve a sequence of numerical problems on the resulting grid. We will use such solvers to generate our data and high-fidelity reference solutions for evaluations.

3.1 Neural PDE Solvers

A neural PDE solver is one where the values of the function $\hat{u}_\theta(t,x)$ at specific space–time points are computed using a neural network f_w with weights $w \in \mathbb{R}^n$, i.e. $\hat{u}_\theta(t,x) = f_w(\theta,t,x)$. They are often trained by sampling a set of N_{data} points $\theta_i, i = 1,\ldots,N_{\text{data}}$, from some distribution $\mathcal{D}$ over Θ, classically computing corresponding "ground truth" solutions u_i for each of them, and training f_w to approximate these solutions. We will evaluate the performance of learned (and other non-reference) solvers via the **normalized RMSE (nRMSE)** $\|f_w(\theta,\cdot,\cdot) - u_\theta\|_2 / \|u_\theta\|_2$ relative to a ground truth reference solution, averaged over output channels and over a held-out test set of points $\theta \sim \mathcal{D}$. This serves as an imperfect indicator for the performance of the method on solving downstream tasks, e.g. inverse problems over the domain Θ.

3.2 Motivation: Where are neural solvers used?

The scientific workloads where neural PDE solvers have the most promise involve *repeated* forward solves of related PDEs, often to optimize some task metric. For example, solving a design optimization or inverse problem task with an objective like $\theta^\star \in \arg\max_{\theta\in\Theta}\ J(\theta;u_\theta)$. typically requires many forward solves at different values of θ, which is where acceleration via learned solvers may be useful. To quantify this, suppose N forward PDE solves with accuracy ε suffices to optimize this objective to an acceptable value. Then the question we seek to answer next is whether it is better to use a classical or neural solver for this purpose.

3.3 Breakeven Complexity

A key feature of this setting is the up-front cost of a neural PDE solver f_w, which is the sum of two quantities: (i) the cost of generating the data and (ii) the cost of training the model on that data. Since data generation cost can be reasonably assumed to scale linearly with the number N_{data} of training points, we write the data generation cost as $C_{\text{gen}}N_{\text{data}}$, where C_{gen} is the average cost of generating one training trajectory via a classical solver. We then define the total up-front training cost as $B := C_{\text{gen}}N_{\text{data}} + C_{\text{train}}$, where C_{train} is the total cost of all training steps. B can be interpreted as the budget we spend on producing a neural solver before it can be deployed. Let ε_B denote the (worst-case or average) error achieved by the resulting model.

Now, if our downstream task needs N trajectories with error ε_B to solve, then the total cost spent on the task will be $B_{\text{total}} := B + C_{\text{inf}}N$, i.e. the sum of the up-front training cost B and total inference cost $C_{\text{inf}}N$, where C_{inf} is cost per inference using f_w.

To fairly compare a neural PDE solver against a classical solver [21], we compare B_{total} to the cost that a similarly accurate classical solver would incur in order to compute N trajectories. Specifically, let δ_B denote the fidelity—e.g. the resolution—of an **error-matched** classical solver, i.e. one that achieves the same (worst-case or average) error ε_B as the model, and let C_{δ_B} denote its per-trajectory cost. Then, the neural PDE solver is cheaper at the attained error if and only if, for N required

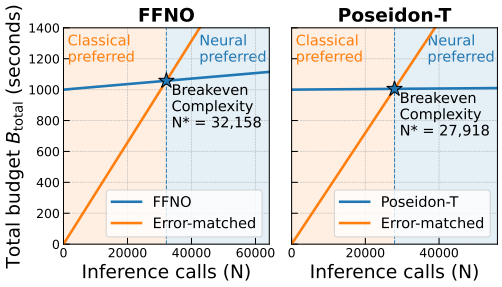

Figure 1: Illustration of the breakeven complexity definition, using FFNO and Poseidon-T on Navier-Stokes as examples. For a fixed training budget $B = 1000$, a learned solver's total costs for simulating N trajectories is $B_{\text{total}} = B + C_{\text{inf}}N$, while the cheapest classical configuration whose error matches the learned accuracy ε_B has cost $C_{\delta_B}N$. The intersection of these two lines, if ever, will occur at $N^\star(B) = \frac{B}{C_{\delta_B} - C_{\text{inf}}}$ (Eq. 1), which we define as the breakeven complexity. For any problem that requires fewer inference calls than this, we prefer to use the classical solver.

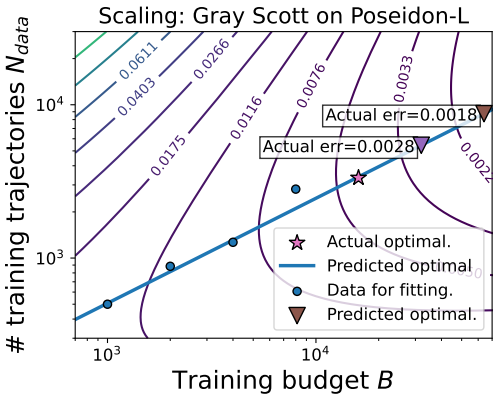

Figure 2: Example of a scaling landscape for the budget-optimal error, using Poseidon-L trained on Gray-Scott and varying the allocation between training trajectories (data-generation) and optimization steps. We fit a model to the small-budget optima and extrapolate to larger budgets to estimate the best-achievable error at each B. A detailed fitting procedure is explained in Appendix C.

trajectories, $B + C_{\text{inf}}N < C_{\delta_B}N$. Rearranging the equation yields an amortization threshold, which we define as the **breakeven complexity**:

$$N^\star := \frac{B}{\max\{C_{\delta_B} - C_{\text{inf}}, 0\}}. \tag{1}$$

As illustrated in Fig. 1, breakeven complexity is the point when a neural PDE solver and its error-matched classical solver require the same cost of solving the downstream task. It thus isolates the amortization regime, being small when (i) the up-front cost B is modest, (ii) inference is substantially cheaper than that of the error-matched classical simulation ($C_{\delta_B} \gg C_{\text{inf}}$), or both. Conversely, breakeven complexity is infinite when classical low-fidelity simulation is already cheaper than inference at matched accuracy.[§] One limitation here is the assumption that simulation costs are roughly the same for all parameters θ, which could fail in certain settings where θ influences the difficulty of the problem; however, taking the average cost over a distribution is a reasonable and meaningful approximation.

4 Evaluation Methodology

In this section, we describe the experimental evaluation protocol we use to compute breakeven complexity. The pipeline involves (i) determining the cost measure, (ii) estimating the best achievable neural solver error for a fixed training budget, (iii) matching the latter's accuracy to a classical baseline's, and (iv) computing the breakeven complexity via the quotient of the training budget and the difference between neural and traditional generation cost.

4.1 Measuring compute cost

We use the wallclock time (seconds) to measure cost. Although FLOPs are commonly used for scaling laws, they are less useful for comparing neural and classical PDE solvers because (i) throughput (FLOPs/s) depends strongly on kernel structure/hardware utilization and (ii) classical solver costs are dominated by stencil/flux evaluations, memory effects, and communications that are not well-represented via FLOPs. Wallclock therefore provides a direct end-to-end cost currency shared by both learned and classical methods, one which is also used in related work [3]. We exclude training

[§]One could choose to also set $N^\star = \infty$ if the nRMSE is greater than one, since that is attained by an all-zero predictor. While we choose to always error-match with a downscaled solver, this observation highlights that $N^\star$ should be assessed together with the error metric it is based on, since that ultimately determines whether a model is suitable for the task.

initialization overheads (e.g. compilation and warmup) as they are not fundamental to the underlying modeling or physics problems. Lastly, in this paper we run both neural and traditional solvers on the same hardware and ensure high GPU-utilization in both cases; in future settings where classical simulation must be done on CPU, a monetary or power usage comparison could be used.

4.2 Error-optimal data generation

A learned solver incurs up-front cost $B = C_{\text{gen}} N_{\text{data}} + C_{\text{train}}$ from data generation and training, where C_{gen} is per-trajectory generation time, N_{data} is the number of generated trajectories, and C_{train} is total training step time. For a given budget B, neural solvers admit many feasible allocations (more data vs. more optimization, different model sizes/hyperparameters) that can yield substantially different errors and inference costs. Since breakeven complexity compares methods at a *fixed up-front budget*, using a non-optimal configuration would conflate the metric with arbitrary training choices. We therefore estimate the *best achievable* error under each budget and use this budget-optimal model as the representative learned solver at budget B.

Empirically, this can be done by sweeping feasible model configurations (i.e. data generation and optimization allocation) for some fixed budget. However, since sweeps at large budgets can be costly, we instead take the following scaling laws-inspired approach. First, we run hyperparameter sweeps on a set of small budgets B while also trying different generation vs. optimization allocations (varying N_{data} vs. C_{train} while keeping $C_{\text{gen}} N_{\text{data}} + C_{\text{train}} \leq B$). We set ε_B to be the best (worst-case or average) error achieved at small budget B and define $N_{\text{data}}(B)$ to be the associated optimal number of training trajectories. Then we fit a scaling model to the labeled pairs $(B, N_{\text{data}}(B))$ in order to predict the error-optimal amount of data to generate at larger budgets B [14]. An example of the fitted scaling landscape graph is show in Fig. 2.

4.3 Error-matched classical solvers

While high-fidelity classical solvers are typically much more expensive than neural PDE solvers, the latter often have much higher error; thus a fair comparison is to compare to low-fidelity classical solver. Thus, for each training budget B, we find the cheapest configuration of the classical solver whose error matches that of the learned solver's accuracy ε_B and use that as our baseline. This error-matching is essential: classical solvers can often be run at reduced fidelity (and thus much lower cost), so comparing surrogate inference cost to an unnecessarily high-fidelity baseline can overstate the usefulness of neural PDE solvers [21].

More formally, let $\{\mathsf{S}_\delta\}$ be a family of classical solver configurations indexed by fidelity δ, with per-trajectory runtime C_δ. Let $\varepsilon(\delta)$ denote its error, computed in the same way as the neural PDE solver error by comparing to a high-fidelity reference on a set of test trajectories. Then we define the **error-matched solver** to be the cheapest solver that is at least ε_B-accurate:

$$\delta_B \in \arg\min_{\delta} \Big\{ C_\delta : \ \varepsilon(\delta) \leq \varepsilon_B \Big\} \tag{2}$$

In practice, we can construct the family $\{\mathsf{S}_\delta\}$ of low-fidelity solvers by coarsening the resolution and timesteps. Starting from a high-fidelity solver, we decrease the resolution by e.g. setting grid spacing $\Delta x \to r\Delta x$ for $r \in \{2, 4, \dots\}$ in the case of regular grids or analogously reducing the number of nodes in the case of unstructured meshes. For each spatial coarsening level, we increase the time step Δt to reduce runtime, while maintaining stability by enforcing a CFL (Courant–Friedrichs–Lewy) condition constraint. Concretely, we choose Δt so that the nondimensional Courant number remains approximately constant across fidelities, i.e. $\mathrm{CFL} \ \propto \ \frac{u_{\max} \Delta t}{\Delta x} \approx \mathrm{const}$, where $u_{\max}$ is a characteristic velocity scale for the benchmark equation. When the solver has additional stability constraints (e.g., diffusion-like terms), we additionally cap Δt by the corresponding bound. Note that, for some numerical methods (e.g. ETDRK time-steppers [8, 33]), the stability and CFL condition does not depend significantly on Δt. In this case, we would fix Δt and only coarsen the grid resolution. This procedure yields a principled grid-timestep schedule that produces a monotone cost-accuracy tradeoff and avoids unstable configurations.

4.4 Computing breakeven complexity

After determining the error-matched solver by defining a family of low-fidelity solvers and using Eq. 2, we are finally able to compute the breakeven complexity (1) $N^\star(B) = B / \max\{C_{\delta_B} - C_{\text{inf}}, 0\}$, with $N^\star(B) = \infty$ when the denominator is 0, i.e. when inference cost is no cheaper than classical

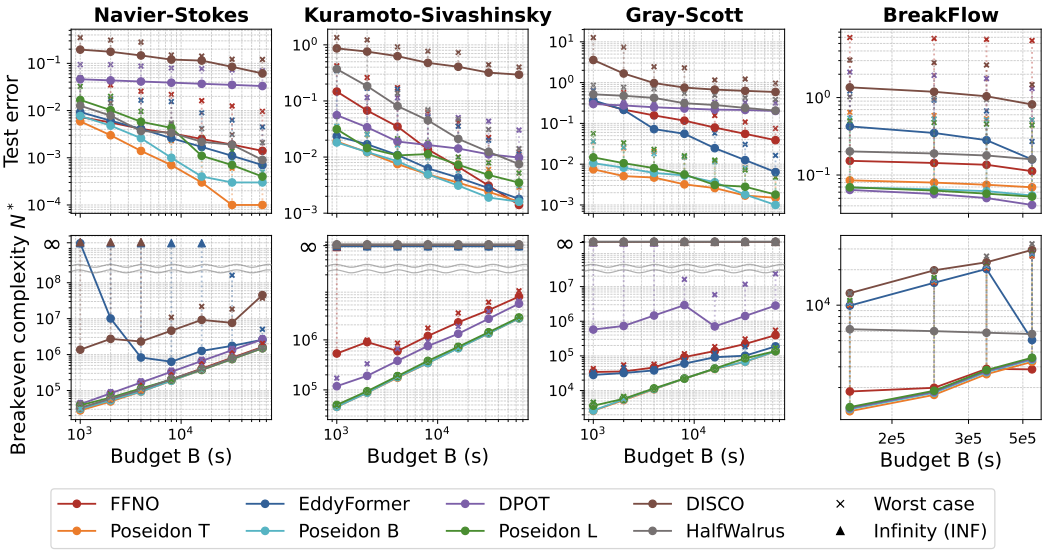

Figure 3: Plots of the test nRMSE (top) and breakeven complexity (bottom), both as functions of the training budget B. "$\times$" and "▲" represent worst-case (error or complexity) and infinite (complexity), respectively. While error decreases monotonically almost everywhere, breakeven complexity is nonlinear because both its numerator and denominator depend on the budget B.

simulation. Note that, given a static set of test trajectories, we can define the error ε_B of the neural PDE-solver and the error $\varepsilon(\delta_B)$ of its matched classical solver in (at least) two ways: via the **average nRMSE** over the test set or the **worst nRMSE** of any trajectory in the test set. We use these to define the **average-case breakeven complexity** and the **worst-case breakeven complexity**, respectively. Average-case breakeven complexity indicates the workload size required for a learned solver to amortize training cost to be more useful than classical simulators when targeting typical accuracy, while worst-case breakeven complexity indicates the workload size required to amortize training cost while meeting robust worst-trajectory accuracy guarantees. Note that since we consider the worst-case for the both the neural solver and the error-matched classical solver, the worst-case breakeven complexity is not necessarily worse than the average-case.

5 Benchmarks and datasets

We evaluate breakeven complexity on six PDE families: five extant settings from APEBench [16] and a new benchmark, BreakFlow, which uses PyFR [32] to simulate flows in complex geometries. The suite covers smooth, dissipative transient dynamics, with simulations run until the evolving quantities converge to a steady state, enter limit cycles, or reach statistically stationary behavior. Details of the benchmarks and datasets are presented in Appendix A.

The **periodic benchmarks** we consider involve three canonical equations: Navier-Stokes (N-S), Kuramoto-Sivashinsky (K-S), and Gray-Scott (G-S). They are chosen due to their nonlinearity and distinct physical characteristics (advective turbulence, chaotic instability-driven dynamics, and reaction–diffusion pattern formation, respectively). While our primary evaluations focus on 2D domains, we also evaluate the K-S equation in 1D and 3D to analyze difficulty scaling across spatial dimensions. The data trajectories are simulated via the Exponax solver from APEBench [16], a GPU-based pseudo-spectral solver with ETDRK time-stepping. Its Fourier spectral discretization is well-suited to smooth periodic dynamics and provides an efficient, well-controlled classical baseline for breakeven analysis.

The **BreakFlow benchmark** is a new setting we introduce consisting of flows past multiple objects simulated in PyFR [32], a flux-reconstruction code that uses artificial viscosity to enable highly parallel GPU operations for incompressible flows in complex geometries; this makes it particularly well-suited for breakeven analysis. The specific PDE instances in BreakFlow involve rightward flows past rectangles, with sizes, positions, and rotations sampled from a distribution ensuring sufficient separation between obstacles and symmetry about the horizontal axis, as detailed in Appendix A. We generate simulations over three different ranges of Reynolds numbers, $(10, 40]$, $(40, 90]$ and $(90, 160]$.

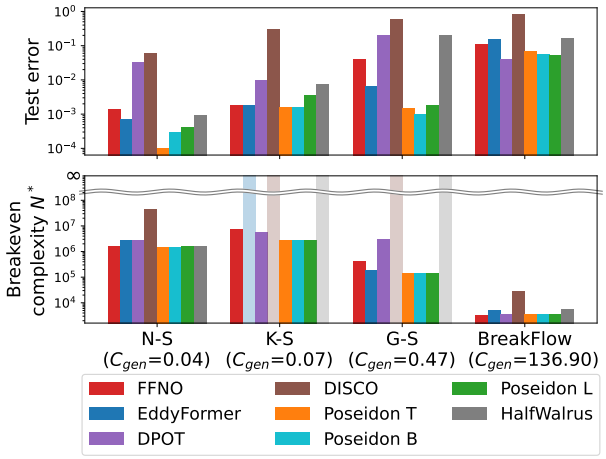

Figure 4: nRMSE (top) and breakeven complexity (bottom) for different models across the four tasks at the largest budget. While nRMSE suggests neural surrogates will struggle on (classically) demanding tasks given the same (training) budget, our compute-aware metric suggests the opposite, being lower for PDEs with longer simulation time (C_{gen}).

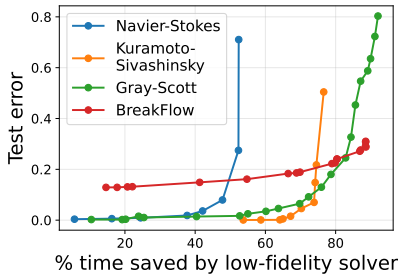

Figure 5: Comparison of how decreasing resolution increases simulation speed across our four PDE settings; the time saved is relative to the high-fidelity solver, with the sequence of low-fidelity solvers determined as in Section 4.3. Unlike the APEBench simulations, BreakFlow is extremely sensitive to changes in resolution, with error increasing rapidly after a slight decrease in solve cost.

6 Experimental Results

We use the evaluation pipeline in Sec. 4 to compute breakeven complexity across the four PDE setups in Sec. 5 for multiple model classes. Specifically, we evaluate two categories of neural PDE solvers: *supervised* models trained from scratch on each task and *foundation-model* (FM) approaches adapted to each task via fine-tuning. For the former we consider four representative architectures: (i) *FFNO* [30], an efficient and tuned variant of the Fourier Neural Operator [18], (ii) *EddyFormer*, a Transformer-based model [6], (iii) *DISCO*, which uses a hypernetwork to dynamically generate an autoregressive operator [22], and (iv) *DPOT*, which uses a Fourier-based attention mechanism [12]. For the latter, we consider two representative FM architectures: (i) the *Poseidon* family (Tiny, Base, and Large) pre-trained on diverse fluid data [13], and (ii) *HalfWalrus*, a Transformer-based cross-domain FM pre-trained on a broad set of continuum dynamics from many physical scenarios [20]. When computing breakeven complexity, we exclude pretraining cost, treating it as a shared, reusable expense amortized across many downstream tasks and thus not attributable to any single deployment.

A complete report of our breakeven complexities at different budgets can be found in Appendix F, with model training details provided in Appendix B. Note that we compute both *average-case* and *worst-case* variants (Sec. 4.4), where worst-case is taken over test trajectories. In this section, we organize our experimental analysis around the following three claims:

1. By forcing the consideration of compute cost, breakeven complexity reveals relative model strengths and weaknesses overlooked in previous accuracy-driven evaluations.
2. We can compare worst-case and average-case variants of breakeven complexity to gain an understanding of the robustness of a learned model relative to a low-fidelity classical solver.
3. Breakeven complexity evaluations reveal that neural solvers need to be called hundreds of thousands of times to be effective in the basic toy PDEs that are typically studied in scientific ML, but on the other hand neural solvers become *more* cost-effective on *harder* PDE problems.

6.1 Claim 1: The value of cost-awareness

In Fig. 3 we plot the budget-optimal test error ε_B and the corresponding breakeven complexity $N^\star(B)$ as functions of the training budget B, doing so across all models and settings under consideration. While the error curves are almost uniformly decreasing across the models and settings, the breakeven complexity can be non-monotone; notably, EddyFormer struggles on low budgets for both fluid settings (Navier-Stokes and BreakFlow) but becomes much stronger at higher budgets, suggesting that to be effective it must be trained sufficiently long and on sufficiently many samples. On the other hand, EddyFormer has $N^\star = \infty$ on Kuramoto–Sivashinsky because its inference cost

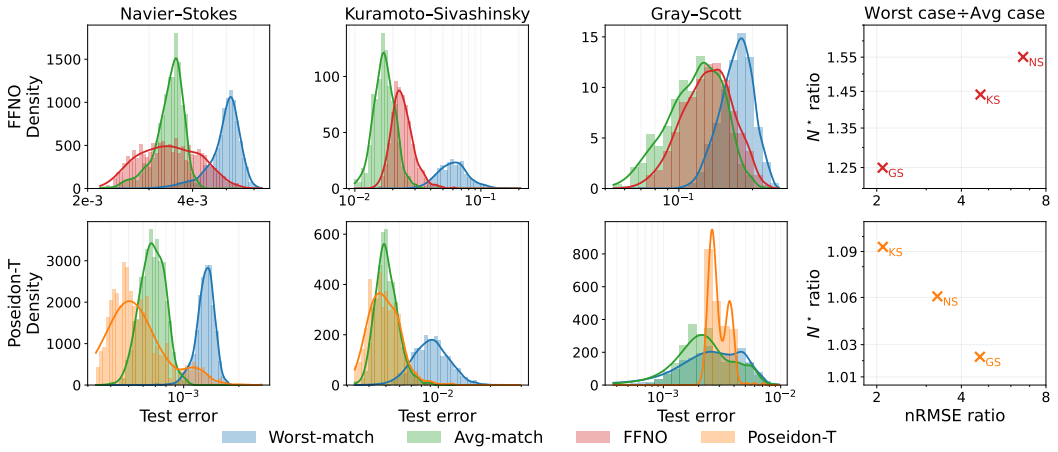

Figure 6: Comparison of the error distributions of FFNO and Poseidon-T (training budget 8K) along with the distributions of the classical solvers that match their worst-case and average-case performance. On the right we compare the worst-case to average-case ratios for $N^\star$ and nRMSE. For both models, their error distribution on Gray-Scott is much more robust, i.e. more concentrated, than on the other PDEs, making the distribution of the worst-case match solver fairly close to that of the average-case; this in turn means the worse solver is not much faster and so the worst-case breakeven complexity does not increase so much over its average-case. This is evidenced in the plots on the right, where the ratio of worst-to-average-case complexity is smaller for Gray-Scott. Notably, the ratio of the worst-case to the average-case nRMSE does not correlate to robustness in this way.

is greater than the simulation cost of an error-matched classical solver, which permits substantial coarsening before its error rises sharply (Fig. 5). The same holds for two of the APEBench settings for HalfWalrus, owing to its high inference cost (Appendix B). A surprising finding is that while the Poseidon models and DPOT dominate BreakFlow according to test error, their breakeven complexity is not much better than that of HalfWalrus and FFNO, with the latter being the most cost-effective at the highest budget we evaluate despite being a supervised model.

Beyond this, breakeven complexity's forced consideration of cost tradeoffs reveals performance shifts that standard metrics might overlook. Most notably, when training cost is equalized we find that Poseidon-L is the worst FM in its family on all tasks except BreakFlow, according to both breakeven complexity (albeit barely) and test nRMSE (more significantly). In contrast, it was reported to be far better than Poseidon-T and Poseidon-B in aggregate on similar tasks [13, Table 9]. This demonstrates the usefulness of our new measure in forcing an accounting of compute costs when doing evaluation.

6.2 Claim 2: Measuring robustness

As discussed in Section 3, we can measure breakeven complexity by matching learned and traditional solvers via their *average* test error or their *worst-case* test error. As shown in Figure 6, it is instructive to look at their ratio: both FFNO and Poseidon-T have relatively close average and worst-case complexities on Gray-Scott, but on the other PDEs the worst-case tends to be much higher. Looking at the distribution of errors across the test trajectories of each task in Fig. 6, we see that Gray-Scott is also the setting where the surrogates' errors are more tightly concentrated than those of their matched solvers. This suggests a smaller gap between worst-case and average-case breakeven complexity reflects a more robust learned solver, i.e. one that is less likely to have instances where it performs much worse than the average instance. Mathematically, this will be caused by the worst-case error-matched solver having an error distribution not too far from the average-case, thus also not being much cheaper and therefore not significantly increasing the complexity. Since optimization problems naturally require exploring unseen parts of the parameter space, the fact that breakeven complexity, unlike nRMSE, can be indicative of robustness is practically meaningful.

6.3 Claim 3: Optimism in the face of harder PDE problems

At first glance, our evaluations in toy settings suggest that hundreds of thousands of calls are necessary to reach cost-effectiveness against error-matched traditional solvers (Fig. 4). However,

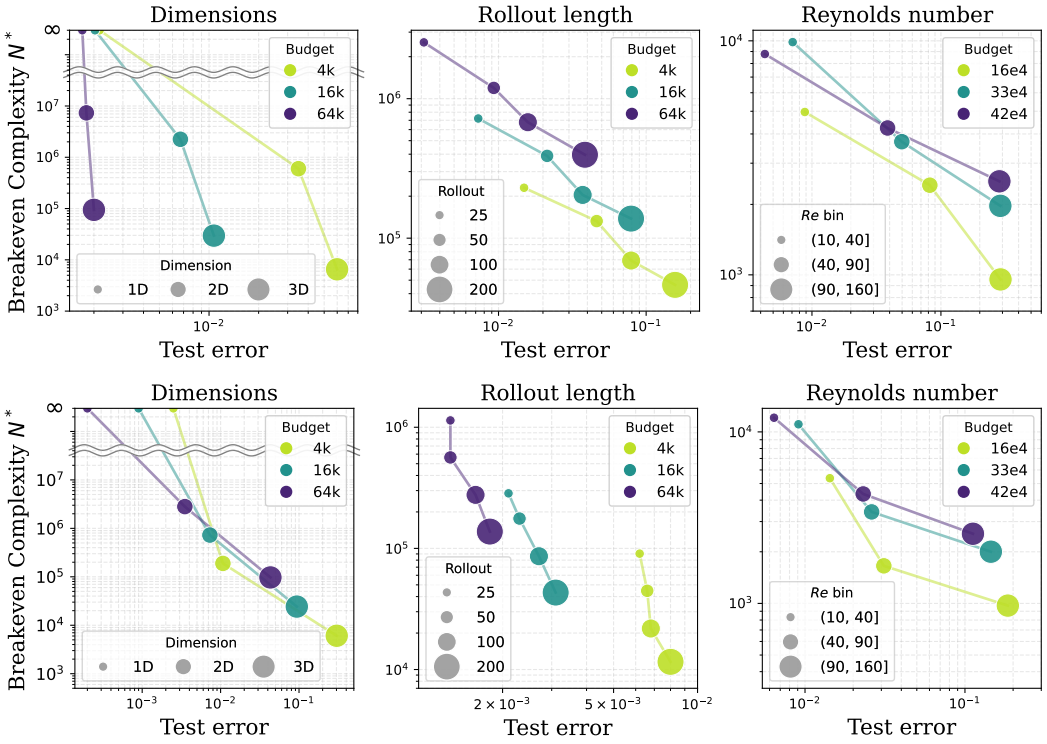

Figure 7: Plot showing the inverse relationship between test error and breakeven complexity of FFNO (top) and Poseidon-L (bottom) as problem difficulty increases on Kuramoto-Sivashinsky (left, varying spatial dimension), Gray-Scott (middle, varying rollout length), and BreakFlow (right, varying Reynolds number). Across all three problem settings, increasing problem difficulty sharply reduces breakeven complexity at the same budget, indicating a growing computational advantage for neural surrogates over classical solvers.

the same plot demonstrates that breakeven complexity scales inversely with the computational cost of classically solving the PDEs, a basic notion of problem difficulty. In particular, neural surrogates typically require only a few thousand inference calls to break even on BreakFlow, an order of magnitude fewer than on Gray-Scott (for most models) and two orders of magnitude fewer than on Navier-Stokes and Kuramoto-Sivashinsky. On the other hand, simulating BreakFlow is two orders of magnitude more expensive than G-S and three orders more than N-S and K-S. This trend suggests that the utility of neural PDE solvers may lie largely in high-fidelity regimes.

However, scaling with simulation cost does not on its own explain this trend, as there could be other reasons why learned solvers may have such a dramatically smaller breakeven complexity on BreakFlow. For example, Figure 5 shows that on the APEBench settings we can easily decrease the solver cost without significantly increasing error, while on BreakFlow the solver is more sensitive, with error rising dramatically even when the low-fidelity solver is only 10% faster than the high-fidelity one. In such cases, the simulation cost of the error-matched solver will remain large and thus will dominate the denominator of Eq. 1, making breakeven complexity small.

We thus conduct further analysis along three other, better-defined difficulty axes: spatial dimensionality (on K–S), temporal rollout length (on G–S), and flow complexity in terms of Reynolds number Re (on BreakFlow). As shown in Figure 7, for both FFNO and Poseidon-L models, all three settings exhibit the same pattern observed above: as problem difficulty increases while the budget is kept fixed, the breakeven complexity *decreases* while the test error *increases*. In more detail, moving from 1D to 3D on Kuramoto–Sivashinsky substantially increases classical solver cost, lowering the breakeven threshold despite the increase in surrogate error. For Gray–Scott, longer rollout horizons make coarse classical solvers less viable because temporal errors accumulate, requiring finer resolutions and again reducing breakeven complexity. Lastly, BreakFlow simulations with well-formed vortex streets

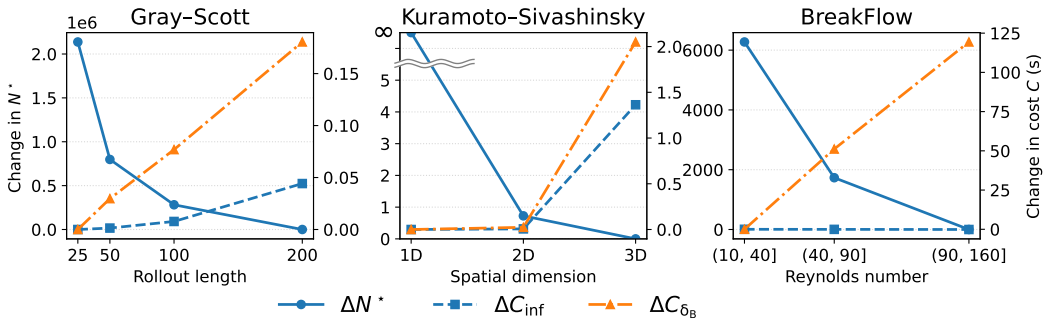

Figure 8: Decomposition of breakeven complexity across difficulty levels for FFNO at the largest evaluated budget (64K seconds for the periodic settings and 420K seconds for BreakFlow). We report changes in $N^\star$ (left axis), neural inference cost C_{inf} (right axis, blue), and error-matched classical solver cost C_{δ_B} (right axis, orange) across three difficulty axes (rollout length, spatial dimensionality, and physical complexity measured by Reynolds number). We see that, across all three settings, the classical solver cost increases substantially faster than the neural inference cost.

($Re \in (90, 160]$) require finer grids and more restrictive time steps than steady flows ($Re \in (10, 40]$), yielding the same trend along the flow-complexity axis [31, 25].

To better understand this trend, we further decompose the breakeven complexity at the largest evaluated budget over the FFNO difficulty ablations. Since B is fixed within each difficulty sweep, a decrease in $N^\star$ is driven by an increase in the per-trajectory saving $C_{\delta_B} - C_{\text{inf}}$. Fig. 8 shows that across all three difficulty axes, C_{δ_B} grows substantially more rapidly with difficulty than C_{inf}. Thus, on the harder problems we tested, the learned solvers save more time per trajectory relative to the error-matched classical solver and break even in fewer trajectories, despite their higher test errors.

These results suggest that neural surrogates can be most computationally advantageous in regimes where classical solver costs grow rapidly, i.e., on more challenging, large-scale problems. As these are precisely the regimes of interest in practical scientific computing, breakeven complexity provides a more optimistic view of the potential of ML for scientific computing than that suggested by McGreivy and Hakim [21], despite their initial introduction of the metric.

7 Conclusion

We formalize *breakeven complexity*, a framework for evaluating neural PDE solvers that complements standard error-based evaluation by quantifying the threshold at which a learned PDE solver becomes cost-effective relative to an error-matched classical baseline; it thus integrates end-to-end costs (data generation, training, inference) while accounting for low-fidelity solvers. Across three 2D periodic PDEs and the BreakFlow benchmark, the metric reveals hidden weaknesses in past evaluations, naturally quantifies robustness, and suggests that the greatest potential of neural PDE solvers will be realized on more challenging benchmarks. As a result, we believe the metric should be considered by any rigorous evaluation of neural PDE methods and should also inform future benchmark construction. The metric can temper overoptimistic speedup claims and thus may help researchers with limited compute choose affordable, reliable solvers.

Limitations of breakeven complexity include its dependence on the hardware, the quality of the classical baseline implementation, and the modeling of inference and training costs; this makes it more challenging to work out than simple error statistics. However, such analysis is exactly what enables a practitioner to assess its value for their use case. Our evaluation is also limited in coverage, with no (e.g.) out-of-distribution settings, shock-dominated dynamics, or stationary problems.

Future work may thus extend the framework or its evaluation to such regimes, or incorporate heterogeneous hardware, adaptive classical solvers, or uncertainty in workload forecasts. Other directions include combining the metric with concrete inverse problems and defining it for the case of hybrid solvers that use learning to correct low-fidelity traditional solvers [15, 6].

References

[1] Abnar, S., H. Shah, D. Busbridge, A. El-Nouby, J. M. Susskind, and V. Thilak (2025). Parameters vs FLOPs: Scaling laws for optimal sparsity for mixture-of-experts language models. In *Forty-second International Conference on Machine Learning*.

[2] Ahrens, J. P., B. Geveci, and C. C. Law (2005). ParaView: An end-user tool for large-data visualization. In *The Visualization Handbook*.

[3] Ashton, N., J. Brandstetter, and S. Mishra (2025). Fluid intelligence: A forward look on AI foundation models in computational fluid dynamics. *arXiv preprint arXiv:2511.20455*.

[4] Bochev, P. B. and D. Ridzal (2009). An optimization-based approach for the design of PDE solution algorithms. *SIAM journal on numerical analysis 47*(5), 3938–3955.

[5] Choudhary, N., V. Singh, A. Talwalkar, N. M. Boffi, M. Khodak, and T. Marwah (2025). Pre-generating multi-difficulty PDE data for few-shot neural PDE solvers. *arXiv preprint arXiv:2512.00564*.

[6] Du, Y. and A. S. Krishnapriyan (2025). EddyFormer: Accelerated neural simulations of three-dimensional turbulence at scale. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

[7] Evans, L. C. (2022). *Partial differential equations*, Volume 19. American mathematical society.

[8] Fu, Z., J. Shen, and J. Yang (2025). Higher-order energy-decreasing exponential time differencing Runge-Kutta methods for gradient flows. *Science China Mathematics 68*(7), 1727–1746.

[9] Geuzaine, C. and J.-F. Remacle (2009). Gmsh: A three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering 79*, 1309–1331.

[10] Grossmann, T. G., U. J. Komorowska, J. Latz, and C.-B. Schönlieb (2024, 01). Can physics-informed neural networks beat the finite element method? *IMA Journal of Applied Mathematics 89*(1), 143–174.

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

[12] Hao, Z., C. Su, S. Liu, J. Berner, C. Ying, H. Su, A. Anandkumar, J. Song, and J. Zhu (2024). DPOT: auto-regressive denoising operator transformer for large-scale PDE pre-training. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

[13] Herde, M., B. Raonic, T. Rohner, R. Käppeli, R. Molinaro, E. de Bezenac, and S. Mishra (2024). Poseidon: Efficient foundation models for PDEs. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

[14] Hoffmann, J., S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. de Las Casas, L. A. Hendricks, J. Welbl, A. Clark, T. Hennigan, E. Noland, K. Millican, G. van den Driessche, B. Damoc, A. Guy, S. Osindero, K. Simonyan, E. Elsen, O. Vinyals, J. W. Rae, and L. Sifre (2022). Training compute-optimal large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA. Curran Associates Inc.

[15] Kochkov, D., J. A. Smith, A. Alieva, Q. Wang, M. P. Brenner, and S. Hoyer (2021). Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences 118*(21).

[16] Koehler, F., S. Niedermayr, R. Westermann, and N. Thuerey (2024). APEBench: A benchmark for autoregressive neural emulators of PDEs. *Advances in Neural Information Processing Systems (NeurIPS) 37*.

[17] LeVeque, R. J. (2007). *Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems*. SIAM.

[18] Li, Z., N. B. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar (2021). Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations*.

[19] Lu, Y., H. Chen, J. Lu, L. Ying, and J. Blanchet (2022). Machine learning for elliptic PDEs: Fast rate generalization bound, neural scaling law and minimax optimality. In *International Conference on Learning Representations*.

[20] McCabe, M., P. Mukhopadhyay, T. Marwah, B. R.-S. Blancard, F. Rozet, C. Diaconu, L. Meyer, K. W. Wong, H. Sotoudeh, A. Bietti, et al. (2025). Walrus: A cross-domain foundation model for continuum dynamics. *arXiv preprint arXiv:2511.15684*.

[21] McGreivy, N. and A. Hakim (2024, September). Weak baselines and reporting biases lead to overoptimism in machine learning for fluid-related partial differential equations. *Nature Machine Intelligence 6*(10), 1256–1269.

[22] Morel, R., J. Han, and E. Oyallon (2025). DISCO: learning to DISCover an evolution operator for multi-physics-agnostic prediction. In *Forty-second International Conference on Machine Learning*.

[23] Ohana, R., M. McCabe, L. Meyer, R. Morel, F. Agocs, M. Beneitez, M. Berger, B. Burkhart, S. Dalziel, D. Fielding, et al. (2024). The Well: a large-scale collection of diverse physics simulations for machine learning. *Advances in Neural Information Processing Systems 37*, 44989–45037.

[24] Raissi, M., P. Perdikaris, and G. E. Karniadakis (2017). Physics informed deep learning (Part I): Data-driven solutions of nonlinear partial differential equations. *arXiv preprint arXiv:1711.10561*.

[25] Roshko, A. (1954). On the development of turbulent wakes from vortex streets. Technical report.

[26] Sacchetti, A., B. Bachmann, K. Löffel, U.-M. Künzi, and B. Paoli (2022). Neural networks to solve partial differential equations: a comparison with finite elements. *IEEE Access 10*, 32271–32279.

[27] Takamoto, M., T. Praditia, R. Leiteritz, D. MacKinlay, F. Alesiani, D. Pflüger, and M. Niepert (2022). PDEBench: An extensive benchmark for scientific machine learning. *Advances in Neural Information Processing Systems 35*, 1596–1611.

[28] Tali, R., A. Rabeh, C.-H. Yang, M. Shadkhah, S. Karki, A. Upadhyaya, S. Dhakshinamoorthy, M. Saadati, S. Sarkar, A. Krishnamurthy, C. Hegde, A. Balu, and B. Ganapathysubramanian (2025). FlowBench: A large scale benchmark for flow simulation over complex geometries. *Journal of Data-centric Machine Learning Research*.

[29] Tangsali, K., R. Ranade, M. A. Nabian, A. Kamenev, P. Sharpe, N. Ashton, R. Cherukuri, and S. Choudhry (2025). A benchmarking framework for ai models in automotive aerodynamics. *arXiv preprint arXiv:2507.10747*.

[30] Tran, A., A. Mathews, L. Xie, and C. S. Ong (2023). Factorized Fourier neural operators. In *The Eleventh International Conference on Learning Representations*.

[31] Williamson, C. H. (1989). Oblique and parallel modes of vortex shedding in the wake of a circular cylinder at low Reynolds numbers. *Journal of Fluid Mechanics 206*, 579–627.

[32] Witherden, F., A. Farrington, and P. Vincent (2014, November). PyFR: An open source framework for solving advection–diffusion type problems on streaming architectures using the flux reconstruction approach. *Computer Physics Communications 185*(11), 3028–3040.

[33] Xu, Z., Z. Sun, and Y.-T. Zhang (2025). Stability and time-step constraints of exponential time differencing Runge–Kutta discontinuous Galerkin methods for advection-diffusion equations. *Journal of Scientific Computing 105*(2), 1–31.

A Dataset specifications

All data generation uses FP32 on NVIDIA L40S GPUs with no concurrent jobs on the GPU.

2D incompressible Navier-Stokes We consider this equation in vorticity form. Letting the velocity field be $\mathbf{u}(t,x) = (u_1, u_2)$ with $\nabla \cdot \mathbf{u} = 0$, define the scalar vorticity $\omega(t,x) := \partial_{x_1} u_2 - \partial_{x_2} u_1$. The vorticity equation is

$$\partial_t \omega + \mathbf{u} \cdot \nabla \omega = \nu \Delta \omega + f, \tag{3}$$

where $\nu > 0$ is the kinematic viscosity and f is forcing.

We generate a total of 200K training trajectories plus 1K testing trajectories using Exponax [16]. We generate at a resolution of 256×256 with time step $\Delta t = 1 \times 10^{-3}$ and simulate over $T = 1$ seconds on a 2D square periodic domain of edge-length $L = 2$. We store 10 frames in total and spectrally downsample the saved frames into resolution 64×64. The average per-trajectory simulation time is 0.0435 seconds.

1D, 2D and 3D Kuramoto-Sivashinsky We consider this equation on a periodic domain $\mathcal{X} = [0, L]^d$, where the spatial dimension is $d \in \{1, 2, 3\}$. While the 2D formulation serves as our standard benchmark, the 1D and 3D variants are included specifically for the difficulty scaling analysis. The governing equation for a scalar field $u(t,x)$ is:

$$\partial_t u \; + \; \Delta u \; + \; \Delta^2 u \; + \; \frac{1}{2} \|\nabla u\|_2^2 \; = \; 0, \tag{4}$$

where ∇ and Δ denote the spatial gradient and Laplacian, respectively, and Δ^2 is the biharmonic operator.

We generate a total of 200K training trajectories plus 1K testing trajectories using Exponax [16]. We generate the data at resolution of 256 with time step $\Delta t = 0.1$ and simulate over $T = 10$ seconds on a 2D square periodic domain of edge-length $L = 50$. We store 50 frames in total and spectrally downsample the saved frames into resolution 64×64. The average per-trajectory simulation time for 1D is 0.0004 seconds. The average per-trajectory simulation time for 2D is 0.0667 seconds. The average per-trajectory simulation time for 3D is 2.7110 seconds.

2D Gray-Scott Let $u(t,x)$ and $v(t,x)$ denote chemical concentrations on a 2D domain $\mathcal{X}$. The Gray–Scott system is formulated by

$$\partial_t u = D_u \Delta u - uv^2 + F(1-u) \tag{5}$$

$$\partial_t v = D_v \Delta v + uv^2 - (F+k)\, v, \tag{6}$$

for diffusivities $D_u, D_v > 0$ and reaction parameters F, k.

We generate a total of 200K training trajectories plus 1K testing trajectories using Exponax [16]. The reaction parameters are $F = 0.029$ and $k = 0.057$ and the diffusivities are set to 2.1×10^{-5} and 1.1×10^{-5}. We generate at a resolution of 256×256 with time step $\Delta t = 0.5$ and simulate over $T = 2000$ seconds on a 2D square periodic domain of edge-length $L = 2$. The end-time is specifically chosen because most Gray-Scott simulations stabilize well before then. We store 200 frames in total and spectrally downsample the saved frames into resolution of 64×64. The average per-trajectory simulation time is 0.4751 seconds.

BreakFlow The BreakFlow dataset is constructed to systematically evaluate the performance, scalability, and physical consistency of neural partial differential equation (PDE) solvers. An example of the resulting vorticity field is given in Fig. 9.

Each simulation consists of four steps:

1. Domain generation: we simulate over the square domain $[0, 50] \times [-25, 25]$ with an outlet boundary on the right and inlet boundaries on left, top, and bottom. We generate rectangular obstacles by looping in a nested fashion over the integers $x = \{11, \ldots, 25\}$ and $y \in \{0, \ldots, 15\}$; at each (x, y) coordinate we place a rectangle with probability 0.5. The rectangle's dimensions are determined by randomly sampling an area a between 4 and 30

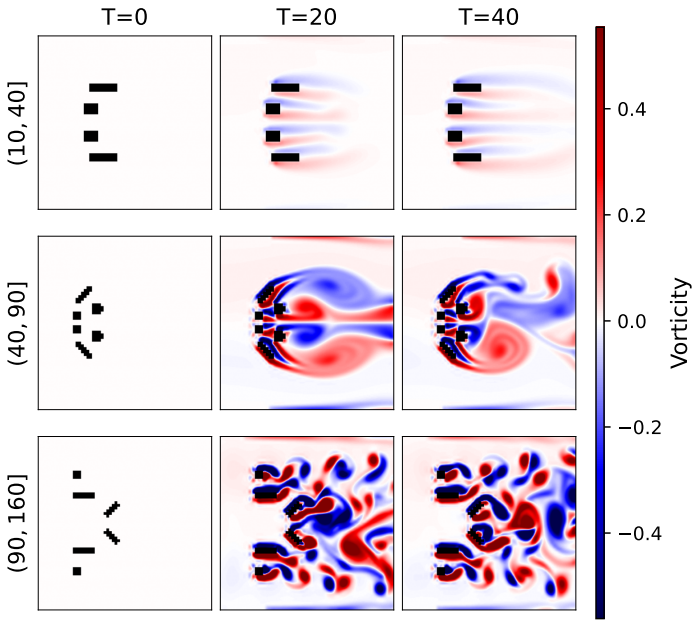

Figure 9: Vorticity snapshots at $t = 0, 200, 400$ of BreakFlow simulations of different Re bins ($Re \in (10, 40], (40, 90]$ and $(90, 160]$) generated via PyFR. There are inlet boundaries on all sides apart from an outlet on the right. Black boxes represent obstacles.

(or 4 and 60 if $y = 0$), then randomly sampling a height $h \in \mathbb{Z}$ and width $w \in \mathbb{Z}$ such that $wh = a$, $w \geq 2$, and $h \geq 2$. The rectangle's rotation is randomly sampled from $\{0, 15, \ldots, 165\}$ degrees, except if $y = 0$ in which case it is either not rotated or sampled from $\{0, 45\}$ if it is a square ($h = w$). If the resulting rectangle overlaps or is closer than distance two to a previously placed rectangle, we do not place it. Lastly, to make the obstacles symmetric about the $y = 0$ axis we take all rectangles with centers (x, y) and place a mirrored rectangle at $(x, -y)$.

2. Mesh generation: we generate a first-order triangular mesh using the Gmsh utility [9]. The mesh is graded to have edge-length $\frac{1}{3}$ near the obstacles, $\frac{2}{3}$ at the edge of a refinement zone $[5, 45] \times [-20, 20]$, and 1 at the edge of the domain, with edge-sizes varying smoothly between them. To decrease the simulation resolution, we simultaneously increase these mesh sizes.

3. Simulation: we run PyFR using the settings of the 2D incompressible cylinder flow example,[¶] but extending the simulation time to 400. To decrease the simulation resolution, we increase the timestepping parameters `dt` and `pseudo-dt`. The average per-trajectory simulation time is 136.9045 seconds.

4. Lastly, we convert the data from an unstructured mesh to a regular 64×64 grid using the ParaView utility [2]. Both the gridded data and unstructured simulation files are available in the dataset repository.

The dataset spans a Reynolds number range of $Re \in (10, 160]$. Data generation is organized into three discrete bins, with Re sampled uniformly at random within each respective range: (1) Bin 1: $Re \in (10, 40]$; (2) Bin 2: $Re \in (40, 90]$ and (3) Bin 3: $Re \in (90, 160]$. Each trajectory is simulated for 400 seconds, yielding 40 saved frames. This duration lets most trajectories reach limit-cycle behavior. For each Reynolds bin, we generate 2,000 training trajectories and 50 test trajectories. We preserve per-simulation wallclock time, velocity x, velocity y, pressure, and vorticity field in the benchmark dataset. All experiments are trained on x-velocity, y-velocity, and pressure fields, and report nRMSE averaged over these fields. The main experiments use shuffled combinations of all three bins for training and testing.

[¶] `https://pyfr.readthedocs.io/en/latest/examples.html#d-incompressible-cylinder-flow`

B Training Details

We present the training settings used to compute breakeven complexity:

- **FFNO** [30]: FFNO is trained with model width of 64, 16 modes and 24 number of layers, learning rate of 1×10^{-3} on periodic equations and 1×10^{-4} on BreakFlow, and weight decay of 1×10^{-4} across all equations. The batch size for training on periodic equations is 100, and is 16 for training on BreakFlow. The FFNO model has a total of 6,698,113 parameters.
- **EddyFormer** [6]: EddyFormer is trained with model width of 32, FFN width of 128, 10 attention blocks, 8 attention heads, learning rate of 1×10^{-3} on periodic equations and 1×10^{-4} on BreakFlow, weight decay of 1×10^{-4} across all equations. The batch size for training on periodic equations is 64, and is 16 for training on BreakFlow. The EddyFormer model has a total of 3,499,171 parameters.
- **DISCO** [22]: DISCO is trained with patch size 16, start channels 8, 4 down/4 up blocks, 4-group GroupNorm, 6 heads, 12 processor blocks, learning rate of 1×10^{-3} on periodic equations and 1×10^{-4} on BreakFlow, and weight decay of 1×10^{-4} across all equations. The batch size for training on periodic equations is 64, and is 16 for training on BreakFlow. The DISCO model has a total of 100,765,696 parameters.
- **DPOT** [12]: DPOT is trained with 8×8 patches, embedding width 512, 4 layers, 4 blocks, MLP ratio 1, output head dimension 32, Adam optimizer, learning rate of 1×10^{-3} on periodic equations and 1×10^{-4} on BreakFlow, and weight decay of 1×10^{-4} across all equations. The batch size for training on periodic equations is 64, and is 16 for training on BreakFlow. The model has a total of 7,010,355 parameters.
- **Poseidon** [13]: Poseidon is tuned on pretrained Poseidon weights [‖], with learning rate of 1×10^{-3} on periodic equations and 1×10^{-4} on BreakFlow, and weight decay of 0.01 across all equations. The batch size for Navier–Stokes is 100, for Kuramoto-Sivashinsky is 300, for Gray-Scott is 200, and for BreakFlow is 16. All Poseidon T, B, and L models share the same set of training settings. Poseidon T has 21M parameters. Poseidon B has 158M parameters. Poseidon L has 629M parameters.
- **HalfWalrus** [20]: HalfWalrus is the 6.4×10^8-parameter variant introduced in McCabe et al. [20, Table 2], with the same model configuration. Its checkpoint was not public at the time of our experiments; we requested and received it from the Walrus team. We selected this variant after preliminary full-Walrus measurements gave $N^\star = \infty$, driven by its high inference cost, on all three periodic benchmarks. We tuned with learning rate of 1×10^{-3}, weight decay of 0.01 across all equations. The batch size for Navier–Stokes is 100, for Kuramoto–Sivashinsky is 300, for Gray-Scott is 200, and for BreakFlow is 16.

Training parameters are chosen to be the optimal values from parameter sweeps. Batch sizes and learning rates tend to be large to encourage faster convergence and lower up-front training cost. The reported budget B includes data generation and the selected training run, excluding the parameter sweeps.

For all training, we warm up for 5 training steps (including the forward pass, backward pass, optimizer step, etc.). We then run another five warm-up training steps, take the average of their step times as the estimate of training-step time, and use this estimate to reset the scheduler. The training budget is decremented using the measured wall-clock time during training.

All training uses the same hardware setup as data generation (Appendix A).

Inference timing. For each model–task pair, we use the largest inference batch that fits on a single NVIDIA L40S. We discard three complete warm-up inference steps, then average the batch runtime divided by batch size across inference batches to obtain C_{inf}. Inference uses FP32 with no concurrent GPU jobs; this measures amortized time per trajectory for many independent solves.

[‖] https://huggingface.co/collections/camlab-ethz/poseidon

C Data scaling fit procedure

We use scaling laws as empirical models of the tradeoff between data generation and optimization, motivated by observed scaling behavior [3, 19]. For a fixed model-task pair, we use a Chinchilla-style additive power-law model [14],

$$\widehat{\mathcal{L}}(N_{\text{data}}, C_{\text{train}}) = L_\infty + aN_{\text{data}}^{-\alpha} + dC_{\text{train}}^{-\beta}, \tag{7}$$

where N_{data} is the number of training trajectories and C_{train} is the wall-clock time allocated to optimization. The coefficients $a, d > 0$ control the magnitudes of the data and optimization terms, while $\alpha, \beta > 0$ are their respective scaling exponents. The fitted L_∞ represents an effective error floor for the fixed model and training regime. For each model–task pair, we fit $(L_\infty, a, \alpha, d, \beta)$ independently to the measured average-case nRMSE of small-budget runs; these parameters are not assumed to be universal.

Predicted allocation frontier. Given a total budget B and per-trajectory data-generation cost C_{gen}, the allocation must satisfy $C_{\text{gen}}N_{\text{data}} + C_{\text{train}} \leq B$. Since the fitted error decreases with optimization time, the predicted allocation uses the remaining budget for training. We therefore define the budget-constrained error surface as

$$\widehat{\mathcal{L}}_B(N_{\text{data}}) := \widehat{\mathcal{L}}\big(N_{\text{data}}, B - C_{\text{gen}}N_{\text{data}}\big), \qquad 0 < N_{\text{data}} < \frac{B}{C_{\text{gen}}}. \tag{8}$$

The predicted allocation frontier is

$$\widehat{N}_{\text{data}}(B) \in \underset{0<N_{\text{data}}<B/C_{\text{gen}}}{\arg\min} \widehat{\mathcal{L}}_B(N_{\text{data}}). \tag{9}$$

Here, $\widehat{N}_{\text{data}}(B)$ denotes the predicted number of training trajectories, distinct from the breakeven complexity $N^\star$. This allocation minimizes the fitted average-case nRMSE; both average-case and worst-case breakeven complexities are subsequently evaluated for the same trained model using its measured errors.

Landscape and large-budget evaluation. For illustration, Fig. 2 shows the fitted budget-constrained surface $\widehat{\mathcal{L}}_B(N_{\text{data}})$ on a dense log-spaced grid with total-budget range $B \in [9 \times 10^2, 7 \times 10^4]$ seconds and training-data range $N_{\text{data}} \in [4 \times 10^2, 3 \times 10^4]$. Feasible allocations satisfy $C_{\text{gen}}N_{\text{data}} < B$. The contour levels span the 3rd to 97th percentiles of the fitted values, with only a subset labeled for readability. The curve shows the predicted allocation frontier $\widehat{N}_{\text{data}}(B)$, with small-budget markers at $B \in \{1000, 2000, 4000, 8000\}$ seconds.

We train at the predicted large-budget allocations and report the measured errors of those models, rather than loss values read from the extrapolated surface. In particular, the 64K-second Poseidon-L allocation in Fig. 2 was predicted using only smaller-budget runs, and its displayed error was measured after training. This tests the achieved performance of the predicted configuration; it does not by itself establish globally optimal allocation or validate extrapolation for other models and tasks.

D Time vs. FLOP choice

Method / Workload	FLOP/sec	Relative to spectral solver
FFNO training	3.542×10^{11}	$90.40\times$
FFNO inference	3.550×10^{11}	$90.61\times$
PyTorch spectral workload	3.918×10^{9}	$1.00\times$

Table 1: Profiler-reported throughput (FLOP/sec) for warmed-up FP32 runs at batch size 100. All workloads use PyTorch and the same profiler; the reference is a representative RK4 pseudo-spectral workload, not the JAX/Exponax implementation used for the periodic benchmarks. Counts depend on profiler operation coverage.

Although FLOPs are a common measure of computational cost in ML frameworks [14, 1], when comparing neural and numerical PDE solvers FLOPs alone are not an adequate fair comparison standard, as the realized throughput (FLOP/sec) can differ substantially between workloads.

We profile FFNO training, FFNO inference, and a representative RK4 pseudo-spectral workload implemented in PyTorch on the same NVIDIA L40S GPU. JAX/XLA supports FLOP estimation, but its operation coverage and counting conventions differed from the PyTorch profiler in our tests. We therefore use a common profiler for this illustrative comparison and do not interpret it as a FLOP measurement of Exponax [16]. Table 1 reports above 3.5×10^{11} FLOP/sec for FFNO and 3.92×10^{9} for the PyTorch spectral workload.

The reported ratios are approximately $90\times$, or nearly two orders of magnitude. They depend on workload structure and profiler coverage: operations delegated to optimized libraries, including FFTs, may be counted differently or incompletely. Low reported FLOP/sec does not by itself demonstrate low hardware utilization, especially for bandwidth-bound kernels. These numbers therefore motivate measuring elapsed cost directly; they do not establish an intrinsic utilization gap. Moreover, many numerical methods depend strongly on CPU performance, memory access patterns, communication, and implementation details, making FLOP-based comparisons less fair across heterogeneous solver families.

For the above reasons, and to be consistent with previous cost-aware research in the neural PDE solver community [3, 21], we use wall-clock as the primary cost measure. Wall-clock can better reflect the actual tradeoff when choosing between a neural surrogate and an error-matched numerical solver.

E Interpreting breakeven complexity

Breakeven complexity ($N^\star$) is not an intrinsic property of a neural architecture or PDE. Rather, it is a deployment-dependent metric dictated by the chosen classical solver, error metric, hardware, and implementation details. Its value serves as a calibrated amortization threshold for a specific setup, not a universal ranking of solvers.

For $C_{\delta_B} > 0$, define $r = C_{\text{inf}}/C_{\delta_B}$. When $r < 1$, Eq. (1) becomes

$$N^\star(B) = \frac{B}{C_{\delta_B}(1-r)}.$$

A point estimate is informative when the positive cost gap is well resolved relative to timing variability. As r approaches 1, small timing changes can produce large changes in $N^\star$; an uncertainty interval or a timing-sensitive flag is then more appropriate. If timing uncertainty spans $r = 1$, both finite amortization and non-amortization remain possible. There is no universal numerical cutoff for r, since the useful threshold depends on the measurement uncertainty. Timing uncertainty is distinct from variation due to training seeds or test trajectories.

Therefore, we treat breakeven complexity primarily as a cost-accounting tool. It highlights how training, inference, and baseline costs interact, exposing cases where accuracy-only evaluations are misleading. In practice, users can recalibrate Eq. (1) for their specific target workloads and runtime environments.

F Full Experimental Results Tables

We present the full experimental data in Sec. 4 here. Tables 2–5 show detailed results for Fig. 3. For budgets of 1K–8K seconds, we report the best configurations from parameter sweeps; for larger budgets, we train using the scaling-predicted number of trajectories. The scaling fit procedure is described in Appendix C. Data % denotes the percentage of budget allocated to data generation. Worst-case results are reported in blue cells, while average-case results are reported in green cells. Tables 6– 11 show detailed results for Fig. 7.

Navier-Stokes		Supervised Model				Foundation Model			
Budget	**Metric**	FFNO	EddyFormer	DPOT	DISCO	Poseidon T	Poseidon B	Poseidon L	HalfWalrus
1e3	Data %	12.19%	2.17%	17.42%	4.35%	9.05%	6.59%	5.07%	2.18%
	$\text{nRMSE}_{\text{worst}}$	0.0472	0.0490	0.0942	0.3520	0.0161	0.0168	0.0326	0.0497
	$\text{nRMSE}_{\text{avg}}$	0.0073	0.0093	0.0460	0.1955	0.0059	0.0077	0.0168	0.0127
	$N^{\star}_{\text{worst}}$	42717	INF	46431	INF	31014	31226	41960	44122
	$N^{\star}_{\text{avg}}$	32158	INF	41661	1369566	27918	30668	38791	32950
2e3	Data %	8.70%	1.74%	17.42%	4.35%	6.10%	4.61%	2.86%	2.18%
	$\text{nRMSE}_{\text{worst}}$	0.0348	0.0179	0.0941	0.3108	0.0072	0.0089	0.0200	0.0144
	$\text{nRMSE}_{\text{avg}}$	0.0056	0.0061	0.0436	0.1763	0.0030	0.0048	0.0102	0.0078
	$N^{\star}_{\text{worst}}$	85427	INF	92862	INF	55841	62453	77674	65901
	$N^{\star}_{\text{avg}}$	57689	10101000	83322	2739133	49450	53181	63397	58960
4e3	Data %	6.53%	1.08%	17.42%	4.35%	3.36%	2.67%	1.54%	1.52%
	$\text{nRMSE}_{\text{worst}}$	0.0257	0.0166	0.0876	0.2821	0.0040	0.0058	0.0096	0.0058
	$\text{nRMSE}_{\text{avg}}$	0.0041	0.0041	0.0413	0.1469	0.0014	0.0026	0.0058	0.0037
	$N^{\star}_{\text{worst}}$	157929	INF	185725	INF	98907	112369	126918	117920
	$N^{\star}_{\text{avg}}$	101796	829222	166645	2311927	93240	93723	113896	103769
8e3	Data %	4.35%	0.76%	10.88%	2.18%	1.52%	2.50%	0.89%	1.09%
	$\text{nRMSE}_{\text{worst}}$	0.0219	0.0155	0.0797	0.1505	0.0023	0.0027	0.0050	0.0055
	$\text{nRMSE}_{\text{avg}}$	0.0033	0.0026	0.0390	0.1203	0.0007	0.0010	0.0043	0.0034
	$N^{\star}_{\text{worst}}$	315858	INF	369307	10956532	197800	198888	201436	235840
	$N^{\star}_{\text{avg}}$	203692	632117	331565	4623855	186481	187447	201281	207538
16e3	Data %	3.18%	0.54%	6.80%	1.09%	0.93%	1.59%	0.49%	0.78%
	$\text{nRMSE}_{\text{worst}}$	0.0162	0.0090	0.0769	0.1427	0.0011	0.0016	0.0027	0.0042
	$\text{nRMSE}_{\text{avg}}$	0.0025	0.0017	0.0369	0.1145	0.0003	0.0004	0.0011	0.0022
	$N^{\star}_{\text{worst}}$	632197	INF	738614	21913064	372962	374895	402554	415077
	$N^{\star}_{\text{avg}}$	407383	1264233	663130	9247710	372962	374895	379374	390225
32e3	Data %	2.27%	0.37%	4.25%	0.55%	0.51%	1.13%	0.27%	0.56%
	$\text{nRMSE}_{\text{worst}}$	0.0125	0.0063	0.0726	0.1224	0.0006	0.0009	0.0014	0.0031
	$\text{nRMSE}_{\text{avg}}$	0.0019	0.0011	0.0350	0.0845	0.0001	0.0003	0.0007	0.0019
	$N^{\star}_{\text{worst}}$	1264395	161611001	1477229	18495420	745924	749791	758749	780450
	$N^{\star}_{\text{avg}}$	814768	1750351	1326260	7626332	745924	749791	758749	780450
64e3	Data %	1.62%	0.26%	2.65%	0.28%	0.28%	0.80%	0.15%	0.40%
	$\text{nRMSE}_{\text{worst}}$	0.0096	0.0045	0.0699	0.1211	0.0003	0.0005	0.0008	0.0021
	$\text{nRMSE}_{\text{avg}}$	0.0014	0.0007	0.0331	0.0612	0.0001	0.0003	0.0004	0.0009
	$N^{\star}_{\text{worst}}$	2059435	5056935	2954459	36990840	1491848	1499582	1517499	1560901
	$N^{\star}_{\text{avg}}$	1629536	2586780	2652520	45252665	1491848	1499582	1517499	1560901

Table 2: Full results on Navier-Stokes for Fig.3. Worst-case results are shown in blue cells, and average-case results are shown in green cells.

Kuramoto-Sivashinsky		Supervised Model				Foundation Model			
Budget	**Metric**	FFNO	EddyFormer	DPOT	DISCO	Poseidon T	Poseidon B	Poseidon L	HalfWalrus
1e3	Data %	6.67%	3.33%	6.67%	6.67%	5.66%	7.34%	13.35%	3.34%
	$\text{nRMSE}_{\text{worst}}$	0.4228	0.3451	0.1499	1.3481	0.0330	0.0353	0.0561	0.8980
	$\text{nRMSE}_{\text{avg}}$	0.1471	0.0236	0.0558	0.8650	0.0189	0.0182	0.0313	0.3699
	$N^{\star}_{\text{worst}}$	INF	INF	172131	INF	46999	47038	49363	INF
	$N^{\star}_{\text{avg}}$	533930	INF	118618	INF	45862	45875	50058	INF
2e3	Data %	4.67%	2.33%	13.35%	6.67%	4.88%	6.67%	11.34%	3.34%
	$\text{nRMSE}_{\text{worst}}$	0.2617	0.1684	0.1148	1.2241	0.0252	0.0254	0.0319	0.7438
	$\text{nRMSE}_{\text{avg}}$	0.0674	0.0169	0.0339	0.7581	0.0122	0.0123	0.0145	0.1837
	$N^{\star}_{\text{worst}}$	993589	INF	332805	INF	93998	94077	95143	INF
	$N^{\star}_{\text{avg}}$	903789	INF	190939	INF	87702	87725	94082	INF
4e3	Data %	3.33%	1.66%	13.35%	13.35%	4.67%	6.34%	10.89%	2.34%
	$\text{nRMSE}_{\text{worst}}$	0.1726	0.1620	0.1141	0.9211	0.0169	0.0163	0.0212	0.1379
	$\text{nRMSE}_{\text{avg}}$	0.0349	0.0109	0.0189	0.6274	0.0075	0.0084	0.0107	0.0811
	$N^{\star}_{\text{worst}}$	863227	INF	665610	INF	187996	188154	197453	INF
	$N^{\star}_{\text{avg}}$	598991	INF	381879	INF	175405	179759	188906	INF
8e3	Data %	2.33%	1.16%	1.67%	13.35%	4.58%	4.67%	6.67%	1.67%
	$\text{nRMSE}_{\text{worst}}$	0.0641	0.0518	0.0561	0.7674	0.0103	0.0097	0.0183	0.0695
	$\text{nRMSE}_{\text{avg}}$	0.0137	0.0063	0.0164	0.4775	0.0049	0.0049	0.0113	0.0463
	$N^{\star}_{\text{worst}}$	1726455	INF	933011	INF	375993	376309	383547	INF
	$N^{\star}_{\text{avg}}$	1197982	INF	753401	INF	343967	344056	380574	INF
16e3	Data %	1.65%	0.82%	0.21%	13.35%	4.16%	4.34%	6.02%	1.19%
	$\text{nRMSE}_{\text{worst}}$	0.0410	0.0355	0.0506	0.7343	0.0078	0.0058	0.0100	0.0472
	$\text{nRMSE}_{\text{avg}}$	0.0067	0.0043	0.0143	0.4067	0.0035	0.0031	0.0073	0.0212
	$N^{\star}_{\text{worst}}$	3452906	INF	1866020	INF	718841	688111	761148	INF
	$N^{\star}_{\text{avg}}$	2243547	INF	1325361	INF	687934	679346	726583	INF
32e3	Data %	1.16%	0.58%	0.03%	13.35%	3.89%	3.77%	4.87%	0.85%
	$\text{nRMSE}_{\text{worst}}$	0.0223	0.0200	0.0437	0.4432	0.0060	0.0032	0.0079	0.0311
	$\text{nRMSE}_{\text{avg}}$	0.0031	0.0028	0.0113	0.3201	0.0024	0.0019	0.0048	0.0124
	$N^{\star}_{\text{worst}}$	5680026	INF	3732041	INF	1375868	1358347	1453166	INF
	$N^{\star}_{\text{avg}}$	3935282	INF	2650722	INF	1342382	1342721	1423833	INF
64e3	Data %	0.82%	0.41%	0.00%	13.35%	3.63%	3.28%	3.94%	0.61%
	$\text{nRMSE}_{\text{worst}}$	0.0122	0.0113	0.0303	0.4017	0.0029	0.0031	0.0052	0.0142
	$\text{nRMSE}_{\text{avg}}$	0.0018	0.0018	0.0099	0.2941	0.0016	0.0016	0.0035	0.0075
	$N^{\star}_{\text{worst}}$	9583855	INF	6027208	INF	2751736	2717387	2847666	INF
	$N^{\star}_{\text{avg}}$	7346417	INF	5301444	INF	2684765	2685443	2810154	INF

Table 3: Full results on Kuramoto-Sivashinsky for Fig.3. Worst-case results are shown in blue cells, and average-case results are shown in green cells.

Gray-Scott		Supervised Model				Foundation Model			
Budget	**Metric**	FFNO	EddyFormer	DPOT	DISCO	Poseidon T	Poseidon B	Poseidon L	HalfWalrus
1e3	Data %	23.75%	11.87%	23.75%	23.75%	31.73%	33.39%	32.96%	47.51%
	$\text{nRMSE}_{\text{worst}}$	0.6393	0.6444	0.4040	12.7105	0.0361	0.0364	0.0561	0.8677
	$\text{nRMSE}_{\text{avg}}$	0.3432	0.3669	0.2988	3.5880	0.0075	0.0106	0.0147	0.5164
	$N^{\star}_{\text{worst}}$	43146	33962	INF	INF	4479	4524	4625	INF
	$N^{\star}_{\text{avg}}$	34271	28212	576238	INF	2614	2664	3643	INF
2e3	Data %	16.62%	17.81%	23.75%	23.75%	21.06%	19.53%	21.63%	47.51%
	$\text{nRMSE}_{\text{worst}}$	0.5674	0.4790	0.3829	7.4014	0.0250	0.0260	0.0334	0.7692
	$\text{nRMSE}_{\text{avg}}$	0.2179	0.2145	0.2753	1.6723	0.0051	0.0082	0.0106	0.4752
	$N^{\star}_{\text{worst}}$	54918	46312	INF	INF	5668	5704	6790	INF
	$N^{\star}_{\text{avg}}$	35571	32003	731156	INF	5294	5545	5785	INF
4e3	Data %	9.72%	11.87%	23.75%	23.75%	14.55%	13.71%	16.07%	47.51%
	$\text{nRMSE}_{\text{worst}}$	0.2592	0.1489	0.3459	2.4451	0.0206	0.0181	0.0231	0.7050
	$\text{nRMSE}_{\text{avg}}$	0.1571	0.0724	0.2488	0.9755	0.0047	0.0061	0.0080	0.4224
	$N^{\star}_{\text{worst}}$	57810	43135	INF	INF	11337	11409	11610	INF
	$N^{\star}_{\text{avg}}$	46262	38150	1462313	INF	10850	11091	11570	INF
8e3	Data %	5.93%	8.31%	23.75%	23.75%	8.67%	8.10%	8.63%	33.36%
	$\text{nRMSE}_{\text{worst}}$	0.2451	0.1188	0.3451	2.2996	0.0149	0.0151	0.0161	0.5446
	$\text{nRMSE}_{\text{avg}}$	0.1168	0.0556	0.2319	0.7551	0.0032	0.0053	0.0056	0.3122
	$N^{\star}_{\text{worst}}$	115620	76301	16127867	INF	22675	22819	22487	INF
	$N^{\star}_{\text{avg}}$	92524	59907	2924626	INF	22178	22182	22196	INF
16e3	Data %	3.79%	8.31%	23.75%	23.75%	6.89%	5.19%	6.03%	23.75%
	$\text{nRMSE}_{\text{worst}}$	0.1561	0.0571	0.3297	1.1566	0.0119	0.0114	0.0126	0.4909
	$\text{nRMSE}_{\text{avg}}$	0.0790	0.0247	0.2177	0.6737	0.0026	0.0036	0.0031	0.2823
	$N^{\star}_{\text{worst}}$	185048	119813	5849253	INF	45394	45632	46264	INF
	$N^{\star}_{\text{avg}}$	138107	91481	709758	INF	41900	42630	43181	INF
32e3	Data %	2.37%	7.17%	23.75%	23.75%	4.72%	3.28%	3.91%	16.45%
	$\text{nRMSE}_{\text{worst}}$	0.1082	0.0306	0.3276	1.2234	0.0083	0.0081	0.0072	0.4232
	$\text{nRMSE}_{\text{avg}}$	0.0554	0.0126	0.2134	0.6279	0.0017	0.0018	0.0028	0.2399
	$N^{\star}_{\text{worst}}$	308477	182962	11698506	INF	90788	91264	92528	INF
	$N^{\star}_{\text{avg}}$	221385	101407	1419517	INF	67869	68135	85280	INF
64e3	Data %	1.48%	6.19%	23.75%	23.75%	3.24%	2.07%	2.54%	11.55%
	$\text{nRMSE}_{\text{worst}}$	0.0751	0.0164	0.3177	0.9675	0.0068	0.0064	0.0048	0.3849
	$\text{nRMSE}_{\text{avg}}$	0.0388	0.0064	0.2020	0.5895	0.0015	0.0010	0.0018	0.2060
	$N^{\star}_{\text{worst}}$	552428	198286	23397013	INF	169688	170520	172724	INF
	$N^{\star}_{\text{avg}}$	396305	188096	2839035	INF	135739	136271	137674	INF

Table 4: Full results on Gray-Scott for Fig.3. Worst-case results are shown in blue cells, and average-case results are shown in green cells.

BreakFlow		Supervised Model				Foundation Model			
Budget	**Metric**	FFNO	EddyFormer	DPOT	DISCO	Poseidon T	Poseidon B	Poseidon L	HalfWalrus
16e4	Data %	64.17%	64.17%	64.17%	64.17%	85.56%	85.56%	85.56%	64.17%
	$\mathrm{nRMSE}_{\mathrm{worst}}$	5.9541	1.2171	2.1317	3.0537	0.6455	0.5411	0.5009	1.0234
	$\mathrm{nRMSE}_{\mathrm{avg}}$	0.1524	0.4245	0.0642	1.3549	0.0853	0.0691	0.0694	0.2118
	$N^{\star}_{\mathrm{worst}}$	10695	10692	10673	12053	10686	10700	10700	11809
	$N^{\star}_{\mathrm{avg}}$	1991	10240	1327	12053	1327	1327	1328	5841
25e4	Data %	54.76%	54.76%	54.76%	54.76%	68.45%	68.45%	82.14%	68.45%
	$\mathrm{nRMSE}_{\mathrm{worst}}$	5.7931	0.9283	1.9345	2.8313	0.5856	0.5271	0.4762	0.9798
	$\mathrm{nRMSE}_{\mathrm{avg}}$	0.1430	0.3490	0.0569	1.1879	0.0798	0.0653	0.0630	0.1836
	$N^{\star}_{\mathrm{worst}}$	16711	16707	16677	18833	16697	16719	16724	18451
	$N^{\star}_{\mathrm{avg}}$	2139	16001	1826	18833	1826	1827	1827	5626
33e4	Data %	41.48%	41.48%	62.22%	62.22%	62.22%	62.22%	72.60%	62.22%
	$\mathrm{nRMSE}_{\mathrm{worst}}$	5.6499	0.6715	1.7593	2.6337	0.5324	0.5146	0.4543	0.9410
	$\mathrm{nRMSE}_{\mathrm{avg}}$	0.1356	0.2818	0.0504	1.0394	0.0749	0.0619	0.0574	0.1787
	$N^{\star}_{\mathrm{worst}}$	22058	22053	22014	22053	22041	22069	22070	24356
	$N^{\star}_{\mathrm{avg}}$	3096	18746	2737	22053	2737	2737	2738	5453
42e4	Data %	40.74%	40.74%	57.04%	57.04%	57.04%	57.04%	65.19%	57.04%
	$\mathrm{nRMSE}_{\mathrm{worst}}$	5.4479	0.2712	1.3122	1.4679	0.5137	0.5146	0.4408	0.8368
	$\mathrm{nRMSE}_{\mathrm{avg}}$	0.1121	0.1586	0.0410	0.8147	0.0695	0.0544	0.0526	0.1594
	$N^{\star}_{\mathrm{worst}}$	28074	28068	28018	28068	28052	28088	28089	30998
	$N^{\star}_{\mathrm{avg}}$	3068	5227	3483	28068	3483	3483	3484	5321

Table 5: Full nRMSE and breakeven-complexity results on BreakFlow for Fig.3. Worst-case results are shown in blue cells, and average-case results are shown in green cells. For BreakFlow, the larger average–worst gap may be related to flow separation around multiple obstacles, nonperiodic boundaries, and substantial variation in difficulty across Reynolds-number bins; differences in vortex timing or phase can also produce large pointwise nRMSE. Limited training-data coverage due to expensive simulation may further contribute to this gap, although these are plausible explanations rather than isolated causal findings.

Budget	**1D Kuramoto–Sivashinsky**		**2D Kuramoto–Sivashinsky**		**3D Kuramoto–Sivashinsky**	
	$\mathrm{nRMSE}_{\mathrm{avg}}$	$N^{\star}_{\mathrm{avg}}$	$\mathrm{nRMSE}_{\mathrm{avg}}$	$N^{\star}_{\mathrm{avg}}$	$\mathrm{nRMSE}_{\mathrm{avg}}$	$N^{\star}_{\mathrm{avg}}$
$1e3$	0.002345	INF	0.1471	533930	0.4886	INF
$2e3$	0.002224	INF	0.0674	903789	0.1212	INF
$4e3$	0.002164	INF	0.0349	598991	0.0599	6531
$8e3$	0.002112	INF	0.0137	1197982	0.0305	14799
$16e3$	0.002030	INF	0.0067	2243547	0.0107	29637
$32e3$	0.001964	INF	0.0031	3935282	0.0043	59274
$64e3$	0.001715	INF	0.0018	7346417	0.0020	94378

Table 6: FFNO results across PDE dimensionality for Kuramoto–Sivashinsky equation for Fig.7.

Budget	**Rollout length = 25**		**Rollout length = 50**		**Rollout length = 100**		**Rollout length = 200**	
	$\mathrm{nRMSE}_{\mathrm{avg}}$	$N^{\star}_{\mathrm{avg}}$	$\mathrm{nRMSE}_{\mathrm{avg}}$	$N^{\star}_{\mathrm{avg}}$	$\mathrm{nRMSE}_{\mathrm{avg}}$	$N^{\star}_{\mathrm{avg}}$	$\mathrm{nRMSE}_{\mathrm{avg}}$	$N^{\star}_{\mathrm{avg}}$
$1e3$	0.0413	73188	0.1189	42804	0.1947	33964	0.3432	34271
$2e3$	0.0262	134418	0.0704	75827	0.1408	47072	0.2179	35571
$4e3$	0.0149	229795	0.0462	132580	0.0791	69130	0.1571	46262
$8e3$	0.0121	432644	0.0342	238220	0.0583	125464	0.1168	92524
$16e3$	0.0073	721022	0.0213	389005	0.0371	204212	0.0790	138107
$32e3$	0.0048	1266775	0.0141	641305	0.0244	377058	0.0554	221385
$64e3$	0.0031	2533550	0.0093	1194665	0.0158	678000	0.0388	396305

Table 7: FFNO results across rollout lengths for Gray-Scott equation for Fig.7.

Budget	$Re \in (10, 40]$		$Re \in (40, 90]$		$Re \in (90, 160]$	
	$\mathrm{nRMSE}_{\mathrm{avg}}$	$N^{\star}_{\mathrm{avg}}$	$\mathrm{nRMSE}_{\mathrm{avg}}$	$N^{\star}_{\mathrm{avg}}$	$\mathrm{nRMSE}_{\mathrm{avg}}$	$N^{\star}_{\mathrm{avg}}$
16e4	0.0088	4965	0.0820	2416	0.2897	956
25e4	0.0077	7720	0.0602	2871	0.2892	1493
33e4	0.0071	9874	0.0497	3707	0.2889	1972
42e4	0.0043	8783	0.0386	4242	0.2840	2510

Table 8: FFNO results across Reynolds bins on BreakFlow for Fig. 7.

Budget	**1D Kuramoto–Sivashinsky**		**2D Kuramoto–Sivashinsky**		**3D Kuramoto–Sivashinsky**	
	$\mathrm{nRMSE}_{\mathrm{avg}}$	$N^{\star}_{\mathrm{avg}}$	$\mathrm{nRMSE}_{\mathrm{avg}}$	$N^{\star}_{\mathrm{avg}}$	$\mathrm{nRMSE}_{\mathrm{avg}}$	$N^{\star}_{\mathrm{avg}}$
$4e3$	0.0025	INF	0.0107	188906	0.3026	6070
$16e3$	0.0009	INF	0.0073	726583	0.0937	24281
$64e3$	0.0002	INF	0.0035	2810154	0.0432	97061

Table 9: Poseidon-L results across PDE dimensionality for the Kuramoto–Sivashinsky equation.

Budget	**Rollout length = 25**		**Rollout length = 50**		**Rollout length = 100**		**Rollout length = 200**	
	$\mathrm{nRMSE}_{\mathrm{avg}}$	$N^{\star}_{\mathrm{avg}}$	$\mathrm{nRMSE}_{\mathrm{avg}}$	$N^{\star}_{\mathrm{avg}}$	$\mathrm{nRMSE}_{\mathrm{avg}}$	$N^{\star}_{\mathrm{avg}}$	$\mathrm{nRMSE}_{\mathrm{avg}}$	$N^{\star}_{\mathrm{avg}}$
$4e3$	0.0062	71186	0.0066	50603	0.0068	17286	0.0080	11570
$16e3$	0.0021	284742	0.0023	202411	0.0027	69145	0.0031	43181
$64e3$	0.0013	1138970	0.0013	714297	0.0016	276579	0.0018	137674

Table 10: Poseidon-L results across rollout lengths for the Gray–Scott equation.

Budget	$Re \in (10, 40]$		$Re \in (40, 90]$		$Re \in (90, 160]$	
	$\mathrm{nRMSE}_{\mathrm{avg}}$	$N^{\star}_{\mathrm{avg}}$	$\mathrm{nRMSE}_{\mathrm{avg}}$	$N^{\star}_{\mathrm{avg}}$	$\mathrm{nRMSE}_{\mathrm{avg}}$	$N^{\star}_{\mathrm{avg}}$
$16e4$	0.0143	5371	0.0311	1656	0.1854	971
$33e4$	0.0091	11078	0.0261	3416	0.1452	2002
$42e4$	0.0064	12094	0.0231	4347	0.1122	2548

Table 11: Poseidon-L results across Reynolds bins on BreakFlow.

G Training-seed variability

We evaluate FFNO and Poseidon-T across all four benchmarks and training budgets using three distinct training seeds while keeping the evaluation set fixed. Table 12 reports the mean and standard deviation of nRMSE across seeds, along with the corresponding breakeven complexity.

We observe that $N^{\star}$ remains unchanged across seeds for every setting. Although predictive errors vary between runs, these variations do not affect the matched classical solver configuration. With a fixed inference cost, this leads to identical breakeven complexities across seeds, suggesting that our estimates are robust to training randomness.

Dataset	Model	Budget B (s)	nRMSE ($\pm$ SD)	$N^{\star}_{\mathrm{avg}}$
Navier–Stokes	FFNO	1,000	0.0070 ± 0.000933	32,158
		2,000	0.0058 ± 0.000344	57,689
		4,000	0.0041 ± 0.000221	101,796
		8,000	0.0034 ± 0.000038	203,692
		16,000	0.0025 ± 0.000036	407,383
		32,000	0.0020 ± 0.000034	814,768
		64,000	0.0013 ± 0.000030	1,629,536
	Poseidon-T	1,000	0.0060 ± 0.000664	27,918
		2,000	0.0031 ± 0.000385	49,450
		4,000	0.0014 ± 0.000359	93,240
		8,000	0.0008 ± 0.000098	186,481
		16,000	0.0003 ± 0.000075	372,962
		32,000	0.0001 ± 0.000075	745,924
		64,000	0.0001 ± 0.000068	1,491,848
2D Kuramoto–Sivashinsky	FFNO	1,000	0.1506 ± 0.005157	533,930
		2,000	0.0671 ± 0.005114	903,789
		4,000	0.0331 ± 0.001614	598,991
		8,000	0.0141 ± 0.000469	1,197,982
		16,000	0.0067 ± 0.000152	2,243,547
		32,000	0.0030 ± 0.000124	3,935,282
		64,000	0.0019 ± 0.000097	7,346,417
	Poseidon-T	1,000	0.0190 ± 0.001330	45,862
		2,000	0.0123 ± 0.000247	87,702
		4,000	0.0069 ± 0.000232	175,405
		8,000	0.0049 ± 0.000038	343,967
		16,000	0.0037 ± 0.000712	687,934
		32,000	0.0024 ± 0.000063	1,342,382
		64,000	0.0016 ± 0.000057	2,684,765
Gray–Scott	FFNO	1,000	0.3662 ± 0.017652	34,271
		2,000	0.2296 ± 0.015730	35,571
		4,000	0.1638 ± 0.005966	46,262
		8,000	0.1165 ± 0.004931	92,524
		16,000	0.0788 ± 0.005391	138,107
		32,000	0.0535 ± 0.003780	221,385
		64,000	0.0389 ± 0.001037	396,305
	Poseidon-T	1,000	0.0093 ± 0.002250	2,614
		2,000	0.0053 ± 0.000417	5,294
		4,000	0.0043 ± 0.000292	10,850
		8,000	0.0036 ± 0.000204	22,178
		16,000	0.0025 ± 0.000105	41,900
		32,000	0.0017 ± 0.000066	67,869
		64,000	0.0015 ± 0.000120	135,739
BreakFlow	FFNO	160,000	0.1445 ± 0.006902	1,991
		250,000	0.1418 ± 0.009156	2,139
		330,000	0.1334 ± 0.009034	3,096
		420,000	0.1189 ± 0.004910	3,068
	Poseidon-T	160,000	0.0821 ± 0.005060	1,327
		250,000	0.0795 ± 0.005840	1,826
		330,000	0.0745 ± 0.004545	2,737
		420,000	0.0715 ± 0.001633	3,483

Table 12: Three-seed robustness results across four PDE benchmarks for FFNO and Poseidon-T. Despite variations in nRMSE across training seeds, all three seeds match the same classical solver configuration at each budget, resulting in identical breakeven complexities. This demonstrates the robustness of breakeven complexity to training randomness.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We substantiate every claim made in the abstract and introduction with empirical results.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discussed the limitations as a sub-section in the conclusion section.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

Justification: [N/A]

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We provided code with submission, and detailed exhaustive experimental setups, hyperparameters and setup in the appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: Code is available at `https://github.com/yijingz02/breakeven_complexity`. Data at `https://huggingface.co/datasets/yijingz/breakeven_complexity`.Experimental details are in Appendices A–B. The HalfWalrus checkpoint was obtained directly from its authors and was not publicly available at the time of our experiments (Appendix B).

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: We provided all training and test details in the Appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We analyzed the error distribution and reported the worst case error, and reported multiseed results.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Provided in the appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We adhere to the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: The paper discusses broader impacts, while acknowledging that the societal impact of an evaluation framework for neural PDE solvers is difficult to predict.The paper discusses positive impacts in terms of improving the rigor of neural PDE solver evaluation, reducing overoptimistic speedup claims, and helping researchers with limited compute choose more affordable and reliable solvers in the conclusion section.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.

- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: [N/A]

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: Original authors are cited; the upstream APEBench, Exponax, FFNO, DISCO, and Walrus code is MIT-licensed, PyFR and ParaView use BSD 3-Clause licenses, Gmsh uses GPL v2 or later with its linking exception, and the Poseidon-T/B/L checkpoints use CC BY-NC 4.0. We do not claim complete license documentation: no explicit license was identified in the EddyFormer, DPOT, or Poseidon code repositories, and the non-public HalfWalrus checkpoint was supplied directly by its authors as described in Appendix B.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.

- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: Appendix A documents the benchmark construction, and the released HuggingFace dataset card documents the datasets and specifies the MIT license, including for BreakFlow.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: [N/A]

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: [N/A]

    Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: [N/A]

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.