PICore: Physics-Informed Unsupervised Coreset Selection for Data Efficient Neural Operator Training

Anirudh Satheesh
Department of Computer Science
University of Maryland
College Park, MD 20742
anirudhs@terpmail.umd.edu

Anant Khandelwal
Georgia Institute of Technology
Atlanta, GA 30332
akhandelwal79@gatech.edu

Mucong Ding
Department of Computer Science
University of Maryland
College Park, MD 20742
mcding@umd.edu

Radu Balan
Department of Mathematics
Center for Scientific Computation and Mathematical Modeling
University of Maryland, College Park
College Park, MD 20742
rvbalan@umd.edu

Abstract

Neural operators offer a powerful paradigm for solving partial differential equations (PDEs) that cannot be solved analytically by learning mappings between function spaces. However, there are two main bottlenecks in training neural operators: they require a significant amount of training data to learn these mappings, and this data needs to be labeled, which can only be accessed via expensive simulations with numerical solvers. To alleviate both of these issues simultaneously, we propose PICore, an unsupervised coreset selection framework that identifies the most informative training samples without requiring access to ground-truth PDE solutions. PICore leverages a physics-informed loss to select unlabeled inputs by their potential contribution to operator learning. After selecting a compact subset of inputs, only those samples are simulated using numerical solvers to generate labels, reducing annotation costs. We then train the neural operator on the reduced labeled dataset, significantly decreasing training time as well. Across four diverse PDE benchmarks and multiple coreset selection strategies, PICore achieves up to 78% average increase in training efficiency relative to supervised coreset selection methods with minimal changes in accuracy.

1 Introduction

Partial differential equations (PDEs) are foundational to modeling complex physical systems across science and engineering, from fluid dynamics to quantum mechanics. Most PDEs are non-analytic and need to be solved numerically via Finite Difference Methods (FDMs), Finite Element Methods (FEMs), and Finite Volume Methods (FVMs) Cyrus et al. [1968], Johnson [1988], Eriksson and Johnson [1995], LeVeque [2002]. However, while these approaches yield high accuracy, they are

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: AI for Science.

computationally expensive because they require a simulation to be run to obtain a solution. This is especially true for high-resolution or multi-resolution PDEs, where simulations need to be re-run for each resolution.

Operator learning has emerged as a tool for accelerating PDE solutions by developing data-driven approximations using neural networks instead of traditional grid-based discretizations. Neural operators [Kovachki et al., 2023] are a family of neural networks that learn mappings between function spaces, such as initial conditions to solutions, which allows for resolution-invariant predictions. Models such as Fourier Neural Operator (FNO) [Li et al., 2020] and U-Net Neural Operator (UNO) [Rahman et al., 2023] have shown state-of-the-art performance on various PDE benchmarks, and the ability to generalize to higher-order resolutions with minimal performance drops. Additional work, such as Physics Informed Neural Operator (PINO) [Li et al., 2024c] and Markov Neural Operator (MNO) [Li et al., 2021b], incorporates additional losses into neural operator training to improve performance and increase convergence speed.

Despite these advantages, there are two main data limitations of neural operators. First, they require significant amounts of training data to learn these mappings. Since PDE solvers require high-resolution data over several time frames for accurate training, such training data can be several gigabytes large [Takamoto et al., 2022]. This poses a challenge for training in resource-constrained systems where such models would be trained and deployed, such as for weather prediction [Pathak et al., 2022, Bonev et al., 2023] and carbon storage [Tang et al., 2024]. Secondly, this training data needs to be labeled by including both the initial condition and the ground truth solution. While generating initial conditions is cheap, as they can usually be sampled from a prior distribution, generating ground truth data requires running the full simulation through numerical solvers.

Coreset selection [Agarwal et al., 2005, Sener and Savarese, 2017] is a data-efficient training strategy that identifies a subset of the original training data that is most informative for model learning. Once this subset is identified, training only needs to be done on this subset, significantly reducing training time. However, this requires the full labeled training data to select a subset, which does not alleviate the cost of collecting labels. On the other hand, active learning [Gu et al., 2021, Cao and Tsang, 2022] minimizes data annotation costs by only labeling a subset of the training data at each iteration. Active learning selects a subset by a proxy metric such as Bayesian [Zhao et al., 2021, Beluch et al., 2018] or representation-based methods [Yang and Loog, 2022, Kim and Shin, 2022] at each training iteration, and trains only on that subset. While this does decrease labeling costs at each iteration, active learning requires a significant portion of the dataset for training, reducing convergence speed [Li et al., 2024a]. Thus, we pose the following research question:

How can we simultaneously reduce training time and labeling ground-truth solutions for Neural Operator learning?

We address this problem using unsupervised coreset selection by identifying the most informative training samples based on the physics-informed loss [Li et al., 2024c], a criterion that does not require any ground truth labels. By leveraging this loss, we can prioritize samples likely to improve model performance without the need for expensive simulations. Ground truth labels are then generated only for this selected subset, significantly reducing the overall annotation cost. Finally, we train neural operator models on the reduced, high-quality dataset, leading to faster training times without compromising accuracy.

Our contributions are outlined as follows:

- **We propose PICore, a novel unsupervised framework that uniquely integrates physics-informed losses with coreset selection.** PICore eliminates the need for expensive ground-truth simulations during the data selection phase, simultaneously addressing the data annotation and training bottlenecks in neural operator training.
- **We present the first comprehensive benchmark for coreset selection in the context of neural operator learning.** Through extensive experiments on four diverse PDE datasets, we show that PICore achieves competitive accuracy to supervised methods while dramatically improving end-to-end training efficiency by up to 78% relative to supervised coreset selection and $5\times$ relative to non-coreset baselines.
- **We demonstrate the modularity and generality of the PICore framework.** Our method is not tied to a specific architecture or selection algorithm, and we show its effectiveness

across two different neural operators (FNO and UNO) and five distinct coreset selection strategies.

2 Related Work

2.1 Neural Operators

While typical deep neural nets are used to map and model finite-dimensional vector spaces, such as text embeddings or images, neural operators map infinite-dimensional vector spaces, such as the space of functions [Kovachki et al., 2021]. Neural operators are then widely used to represent differential equation solutions due to their ability to have a *family* of solutions. In the context of solving partial differential equations, a neural operator can take a function as an input (e.g. temperature at a point) and output a related function (e.g. heat over time at a point).

Among the first modern neural operators, DeepONet [Lu et al., 2021] uses the universal approximation theorem for operators with a branch and trunk network to model inputs and outputs. The Fourier Neural Operator (FNO) [Li et al., 2020] expands on this by performing kernel operations in Fourier space, which results in a more expressive model with better performance on more challenging PDE datasets, such as Navier Stokes. U-Net Neural Operator (UNO) [Rahman et al., 2023] expands on FNO by using a U-Net based structure to build deeper neural operators, and Convolutional Neural Operator (CNO) [Raonic et al., 2023] leverages convolutions to preserve the continuous structure of PDEs, even when discretized. Additional work improves training by incorporating additional losses. Physics Informed Neural Operator (PINO) [Li et al., 2024c] uses the physics informed loss to anchor the output to conform to the PDE dynamics, and Markov Neural Operator (MNO) [Li et al., 2022] uses dissipativity regularization to improve accuracy for more chaotic systems.

2.2 Coreset Selection

For problems where training is too expensive or slow, coreset selection can accelerate training while preserving accuracy. Coreset selection methods can be largely categorized into two types: training-free methods that leverage the geometric properties of the data, and training-based methods that use model-specific information to score data points. Training-free methods involve random [Guo et al., 2022, Gupta et al., 2023] and geometry-informed selection [Welling, 2009, Chen et al., 2012]. Recent work on training-based methods can be split into three groups: (i) submodular approaches to maximize the coverage of the selected dataset [Wei et al., 2015, Mirzasoleiman et al., 2020, Pooladzandi et al., 2022], (ii) gradient-based approaches to exactly find the influence of a data point [Killamsetty et al., 2021a, Paul et al., 2021], and (iii) bilevel optimization methods to improve generalization performance [Killamsetty et al., 2021c,b].

2.3 Data Efficiency for Neural Operators

The closest existing work to our own is Chen et al. [2024], which develops an unsupervised pretraining strategy that leverages Masked Autoencoders (MAEs) to learn effective unsupervised representations, which are then used to fine-tune with a smaller ground-truth dataset. However, this indirectly addresses issues with training efficiency and data labeling costs using a two-stage training process, whereas PICore directly addresses both problems in a single training cycle. Hemmasian and Farimani [2024] avoid running expensive simulations on high-resolution data by pretraining neural operators in low dimensions, but this requires a factorized neural operator such as Factorized Fourier Neural Operator (FFNO) [Tran et al., 2021]. In contrast, our method is independent of the operator architecture. Li et al. [2024b] uses an active learning strategy to reduce labeling costs from running simulations by maximizing a utility cost ratio. However, this is specific to FNO and only addresses the cost of data annotation and not training efficiency.

3 Preliminaries

3.1 Neural Operators for PDE Solution Generation

Many physical systems can be modeled using partial differential equations (PDEs), which describe the evolution of a function $u \in \mathcal{U}$ over a domain. A general PDE can be expressed as

$$\mathcal{F}(u, a) = 0, \quad \text{on } \Omega \subset \mathbb{R}^d, \tag{1}$$

where $a \in \mathcal{A}$ represents input parameters such as boundary conditions, initial conditions, or physical coefficients; $\mathcal{F} : \mathcal{U} \times \mathcal{A} \to \mathcal{Z}$ is a differentiable and potentially nonlinear operator; and $\mathcal{A}$,$\mathcal{U}$ are Banach spaces over the bounded domain Ω.

For stationary (time-independent) PDEs, the problem takes the form

$$\begin{aligned} \mathcal{F}(u, a) &= 0, \quad \text{on } \Omega \subset \mathbb{R}^d, \\ u &= h, \quad \text{on } \partial\Omega, \end{aligned} \tag{2}$$

where h defines the boundary condition on the domain boundary $\partial\Omega$.

For dynamic (time-dependent) PDEs, the input a is restricted to the initial condition $u|_{t=0}$, and the operator $\mathcal{F}$ is defined on the spatiotemporal domain $\Omega \times \mathcal{T}$:

$$\begin{aligned} \mathcal{F}(u, a) &= 0, \quad \text{on } \Omega \times \mathcal{T}, \\ u &= h, \quad \text{on } \partial\Omega \times \mathcal{T}, \\ u &= a, \quad \text{on } \Omega \times \{0\}, \end{aligned} \tag{3}$$

where $\mathcal{T} = (0, T)$ denotes the time domain. Examples of both stationary and dynamic PDEs are provided in Section A.

Unlike conventional neural networks that learn pointwise mappings, neural operators approximate solutions by learning mappings between infinite-dimensional function spaces:

$$\mathcal{G} : \mathcal{A} \to \mathcal{U}. \tag{4}$$

In practice, a PDE dataset consists of pairs $\{(a_i, u_i)\}_{i=1}^N$, where each (a_i, u_i) corresponds to an input-output solution of the PDE. The neural operator $\mathcal{G}$ is approximated by $\mathcal{G}_\theta$ through the optimization

$$\mathcal{G}_\theta = \arg\min_{\theta \in \Theta} \frac{1}{N} \sum_{i=1}^{N} \|\mathcal{G}_\theta(a_i) - u_i\|^2_{L^2(\Omega)}, \tag{5}$$

where Θ is a finite-dimensional parameter space.

3.2 Coreset Selection

Given a dataset $D = \{(x_i, y_i)\}_{i=1}^N$, coreset selection aims to find a subset $S \subseteq D$ such that

$$S = \arg\min_{S' \subset D, |S'| = \beta N} \mathbb{E}_{(x_i, y_i) \sim S'}[\mathcal{L}(x_i, y_i; \theta^{S'})] \tag{6}$$

where β is the percentage of the original dataset selected and $\theta^{S'}$ is the model trained on S. However, there are $O(2^N)$ possible subsets of size βN, so evaluating this objective directly is infeasible for large datasets. Instead, some works leverage a submodular function $f : 2^D \to \mathbb{R}$ which ensures the diminishing return property

$$f(S \cup \{z\}) - f(S) \geq f(T \cup \{z\}) - f(T), \quad \forall S \subseteq T \subseteq D, z \notin T \tag{7}$$

This results in a greedy selection procedure, significantly reducing the subset search space. Another way to perform coreset selection is to use a scoring function and select the top-k data points. Finally, coreset selection can be represented as a bilevel optimization problem, resulting in the following form

$$S = \arg\min_{S' \subset D,\ |S'| = \beta N} \mathcal{L}\big(\theta^*(S')\big) \quad \text{s.t.} \quad \theta^*(S') = \arg\min_{\theta \in \Theta} \sum_{(x_i, y_i) \in S'} \mathcal{L}(x_i, y_i; \theta) \tag{8}$$

4 PICore

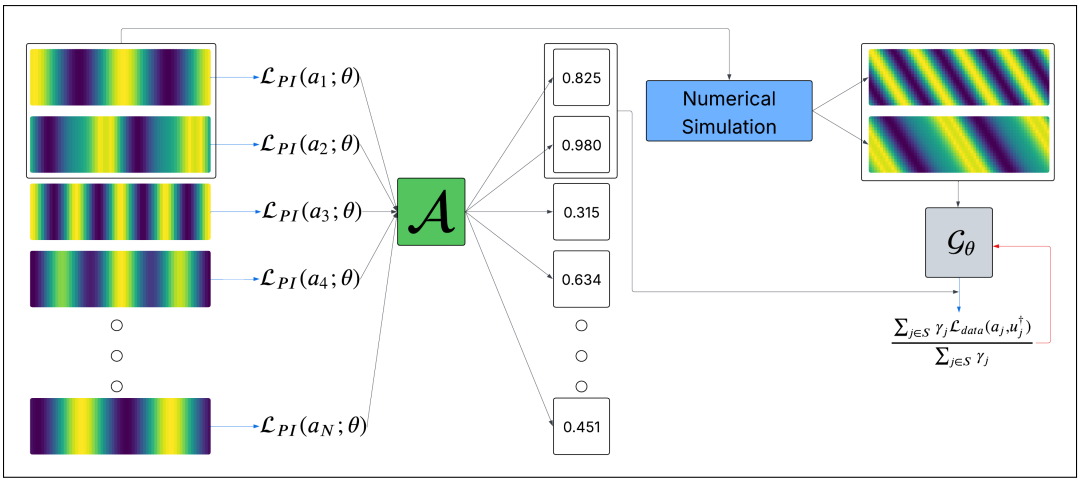


Figure 1: **Overview of the PICore Framework.** Starting with initial conditions and a warm-started neural operator $\mathcal{G}_\theta$, we compute the physics-informed loss $\mathcal{L}_{PI}(a_i;\theta)$ for each condition. A coreset selection algorithm $\mathcal{A}$ picks the most informative samples, assigns weights γ_j, and simulates them using a numerical solver. The resulting labeled subset updates $\mathcal{G}_\theta$ via weighted loss, enabling efficient training on the most impactful data points.

Algorithm 1 PICore: Physics-Informed Coreset Selection for Neural Operators

Require: Unlabeled dataset $D = \{a_i\}_{i=1}^N$; coreset size $k = \beta N$; learning rate α; operator $\mathcal{G}_\theta$; physics-informed loss $\mathcal{L}_{PI}(a;\theta)$; coreset selection algorithm $\mathcal{A}_{\text{select}}$; warmup steps T_w; training steps T

for $t = 1$ to T_w **do**
  for each $a_i \in D$ **do**
    $\theta \leftarrow \theta - \alpha\nabla_\theta\mathcal{L}_{PI}(a_i;\theta)$
for each $a_i \in D$ **do**
  $\ell_i \leftarrow \mathcal{L}_{PI}(a_i;\theta)$
$S \leftarrow \mathcal{A}_{\text{select}}(\{\ell_i\}_{i=1}^N, k)$
$D_c \leftarrow \emptyset$
for each $i \in S$ **do**
  $u_i^\dagger \leftarrow \mathcal{G}^\dagger(a_i)$ {Run numerical simulation}
  $D_c \leftarrow D_c \cup \{(a_i, u_i^\dagger)\}$
for $t = 1$ to T **do**
  for each $(a_i, u_i^\dagger) \in D_c$ **do**
    $\theta \leftarrow \theta - \alpha\nabla_\theta\mathcal{L}_{\text{data}}(a_i, u_i^\dagger;\theta)$

To address both issues of training time and data labeling costs for Neural Operator learning, we introduce PICore, an unsupervised coreset selection method that leverages a physics-informed loss to bypass the need for labeled training data during coreset selection.

Instead of using the ground truth PDE solution and supervised losses, the physics-informed loss evaluates the degree to which operator approximation $\mathcal{G}_\theta(a)$ satisfies the governing PDEs defined in either the stationary form or the dynamic form. The physics-informed loss penalizes violations of the PDE (PDE residual) in the interior of the domain, as well as deviations from the given boundary and initial conditions. For neural operators, the physics-informed loss is defined as

$$\mathcal{L}_{PI}(a;\theta) = \|\mathcal{F}(\mathcal{G}_\theta(a), a)\|^2_{L^2(\Omega)} + \lambda\,\|\mathcal{G}_\theta(a) - h\|^2_{L^2(\partial\Omega)} \tag{9}$$

for stationary PDEs and

$$\mathcal{L}_{PI}(a;\theta) = \|\mathcal{F}(\mathcal{G}_\theta(a), a)\|^2_{L^2(\Omega\times\mathcal{T})} + \lambda\,\|\mathcal{G}_\theta(a) - h\|^2_{L^2(\partial\Omega\times\mathcal{T})} + \mu\,\left\|\mathcal{G}_\theta(a)\big|_{t=0} - a\right\|^2_{L^2(\Omega)} \tag{10}$$

for dynamic PDEs.

Given solely an unlabeled dataset $D = \{a_i\}_{i=1}^N$ that can be cheaply generated (usually by sampling from a prior distribution or sensor readings), PICore selects a coreset of D by solving

$$S = \underset{S' \subset D, |S'| = \beta N}{\arg\min} \mathbb{E}_{a_i \sim S'} \left[\mathcal{L}_{PI} \left(a_i; \theta^{S'} \right) \right] \tag{11}$$

using any existing coreset selection algorithm where $\theta^{S'}$ is the operator trained on S'. After selecting the coreset S, we simulate the true solutions $u_i^\dagger = \mathcal{G}(a_i)$ for each $a_i \in S$ using a traditional numerical solver, which forms the labeled subset $D_c = \{(a_i, u_i^\dagger)\}_{a_i \in S}$. Finally, we train the neural operator $\mathcal{G}_\theta$ on D_c for T epochs with the standard supervised data loss

$$\mathcal{L}_{\text{data}}(a_i, u_i^\dagger) = \|\mathcal{G}_\theta(a_i) - u_i^\dagger\|^2_{L^2(\Omega \times \mathcal{T})} \tag{12}$$

Before coreset selection, we warm-start the neural operator with the physics-informed loss over the full dataset for a small number of epochs $T_w << T$. Warm starting is common in prior coreset selection methods [Killamsetty et al., 2021a] and is necessary as most coreset selection algorithms require gradient information, which is unusable with a randomly initialized model. We provide the full algorithm in Algorithm 1.

5 Experimental Details

We conduct experiments on four representative PDE benchmarks spanning both stationary and time-dependent dynamics widely used in the neural operator literature: the time-dependent 1D Advection and Burger Equations, the stationary 2D Darcy Flow Equation, and the 2D time-dependent Navier-Stokes Incompressible Equation. Additional information on the datasets can be found in Section A. We use the Fourier Neural Operator (FNO) [Li et al., 2020] and U-Net Neural Operator [Rahman et al., 2023] as the base models for all experiments due to their implementation simplicity and performance. However, PICore can work out of the box with any neural operator. We also use 5 coreset selection algorithms in our experiments: CRAIG [Mirzasoleiman et al., 2020], GradMatch [Killamsetty et al., 2021a], AdaCore [Pooladzandi et al., 2022], EL2N [Paul et al., 2021] and graNd [Paul et al., 2021]. CRAIG, AdaCore, and GradMatch are submodular methods that try to match the gradient sum of the coreset to the gradient sum of the entire dataset. GraNd and EL2N are score based methods that use the gradient or the loss. Additional information on these coreset selection algorithms can be found in Section B. We use coreset selection percentages of 20%, 30%, 40%, 60%, and 80%.

We report the results of each experiment with the normalized root mean square error loss (NRMSE):

$$\frac{\|\mathcal{G}_\theta(a_i) - u_i^\dagger\|^2_{L^2(\Omega \times \mathcal{T})}}{\|u_i^\dagger\|^2_{L^2(\Omega \times \mathcal{T})}}$$

used in Takamoto et al. [2022]. We use this as a normalized version of the data loss because the value of the $u_i^\dagger$ at each spatiotemporal point is very small, resulting in small MSE values and potential gradient vanishing during training. We also use the uniform spatiotemporal discretization at an input resolution of 64 for Ω. Since FNO and UNO are resolution invariant, we also evaluate at higher resolutions for zero-shot super resolution in Section **??**. For all experiments we use $\lambda = 1$ and $\mu = 1$, but this is relatively arbitrary, we did not conduct any hyperparameter tuning.

We use $T_w = 25$ warmup epochs and reset the neural operator to its initialization to ensure fair comparisons between supervised and physics-informed coreset selection. Then, we train neural operators for $T = 500$ epochs and report the average NRMSE over 5 seeds on a held-out test set at the input resolution. All experiments were run on a single RTX A4000 GPU. We calculate the acceleration as the total time taken for supervised coreset selection / PICore (including data generation, warm starting, and training time) divided by the total time for the non-coreset baseline.

6 Results

We report the core findings for PICore and supervised coreset selection across the four representative PDE datasets in Tables 1, 2, 3, and 4. We also compare PICore to random subset selection and an active

learning baseline based on uncertainty. Since most active learning baselines are for classification problems, we extend loss-as-uncertainty methods in [Liu and Li, 2023] to neural operators. In addition to the average test NRMSE over the best coreset selection algorithm for each method, we show the decrease in full training time (including data annotation costs through simulation) relative to the non-coreset selection baseline. Our results demonstrate that PICore consistently achieves competitive test performance compared to supervised coreset selection while providing substantial computational efficiency gains, primarily by reducing expensive data annotation (simulation) costs during the coreset selection phase.

Table 1: Advection NRMSE at resolution 64

Operator	Method	20.0%	30.0%	40.0%	60.0%	80.0%	100.0%
FNO	Random	$\mathbf{3.39} \pm 0.07 \times 10^{-2}$ (5.10×)	$\mathbf{2.89} \pm 0.03 \times 10^{-2}$ (3.32×)	$\underline{2.68} \pm 0.02 \times 10^{-2}$ (2.56×)	$2.47 \pm 0.03 \times 10^{-2}$ (1.72×)	$2.37 \pm 0.04 \times 10^{-2}$ (1.28×)	$2.22 \pm 0.05 \times 10^{-2}$ (1.00×)
	Active Learning	$8.32 \pm 0.58 \times 10^{-2}$ (5.04×)	$6.29 \pm 0.40 \times 10^{-2}$ (3.28×)	$4.78 \pm 0.27 \times 10^{-2}$ (2.52×)	$3.51 \pm 0.16 \times 10^{-2}$ (1.69×)	$2.96 \pm 0.07 \times 10^{-2}$ (1.26×)	$2.22 \pm 0.05 \times 10^{-2}$ (1.00×)
	Supervised (graNd)	$\underline{3.42} \pm 0.12 \times 10^{-2}$ (4.70×)	$\underline{2.96} \pm 0.09 \times 10^{-2}$ (3.15×)	$\mathbf{2.64} \pm 0.03 \times 10^{-2}$ (2.45×)	$\underline{2.42} \pm 0.03 \times 10^{-2}$ (1.66×)	$\underline{2.25} \pm 0.02 \times 10^{-2}$ (1.26×)	$2.22 \pm 0.05 \times 10^{-2}$ (1.00×)
	PICore (graNd)	$3.46 \pm 0.13 \times 10^{-2}$ (5.06×)	$3.04 \pm 0.15 \times 10^{-2}$ (3.27×)	$2.69 \pm 0.05 \times 10^{-2}$ (2.54×)	$\mathbf{2.40} \pm 0.04 \times 10^{-2}$ (1.68×)	$\mathbf{2.25} \pm 0.04 \times 10^{-2}$ (1.26×)	$2.22 \pm 0.05 \times 10^{-2}$ (1.00×)
UNO	Random	$1.59 \pm 0.02 \times 10^{-1}$ (5.08×)	$1.50 \pm 0.01 \times 10^{-1}$ (3.35×)	$1.44 \pm 0.007 \times 10^{-1}$ (2.55×)	$1.42 \pm 0.12 \times 10^{-1}$ (1.70×)	$1.32 \pm 0.12 \times 10^{-1}$ (1.28×)	$7.27 \pm 0.28 \times 10^{-2}$ (1.00×)
	Active Learning	$1.96 \pm 0.04 \times 10^{-1}$ (5.05×)	$1.59 \pm 0.05 \times 10^{-1}$ (3.32×)	$\mathbf{9.20} \pm 0.73 \times 10^{-2}$ (2.52×)	$\mathbf{7.49} \pm 0.47 \times 10^{-2}$ (1.68×)	$\mathbf{6.83} \pm 0.05 \times 10^{-2}$ (1.26×)	$7.27 \pm 0.28 \times 10^{-2}$ (1.00×)
	Supervised (gradmatch)	$\mathbf{1.55} \pm 0.02 \times 10^{-1}$ (4.84×)	$\underline{1.48} \pm 0.01 \times 10^{-1}$ (3.23×)	$\underline{1.42} \pm 0.02 \times 10^{-1}$ (2.47×)	$\underline{1.17} \pm 0.14 \times 10^{-1}$ (1.67×)	$\underline{8.69} \pm 1.23 \times 10^{-2}$ (1.25×)	$7.27 \pm 0.28 \times 10^{-2}$ (1.00×)
	PICore (gradmatch)	$\mathbf{1.55} \pm 0.01 \times 10^{-1}$ (5.07×)	$\mathbf{1.47} \pm 0.01 \times 10^{-1}$ (3.34×)	$1.43 \pm 0.008 \times 10^{-1}$ (2.53×)	$1.26 \pm 0.09 \times 10^{-1}$ (1.69×)	$9.06 \pm 1.07 \times 10^{-2}$ (1.26×)	$7.27 \pm 0.28 \times 10^{-2}$ (1.00×)

Table 2: Burgers NRMSE at resolution 64

Operator	Method	20.0%	30.0%	40.0%	60.0%	80.0%	100.0%
FNO	Random	$1.85 \pm 0.09 \times 10^{-2}$ (5.07×)	$1.18 \pm 0.06 \times 10^{-2}$ (3.31×)	$8.23 \pm 0.21 \times 10^{-3}$ (2.56×)	$5.82 \pm 0.21 \times 10^{-3}$ (1.72×)	$4.75 \pm 0.13 \times 10^{-3}$ (1.28×)	$3.95 \pm 0.10 \times 10^{-3}$ (1.00×)
	Active Learning	$8.76 \pm 2.52 \times 10^{-2}$ (5.03×)	$4.57 \pm 0.95 \times 10^{-2}$ (3.25×)	$3.30 \pm 0.57 \times 10^{-2}$ (2.52×)	$2.06 \pm 0.07 \times 10^{-2}$ (1.68×)	$1.37 \pm 0.18 \times 10^{-2}$ (1.26×)	$3.95 \pm 0.10 \times 10^{-3}$ (1.00×)
	Supervised (gradmatch)	$\mathbf{1.71} \pm 0.16 \times 10^{-2}$ (3.28×)	$\underline{1.12} \pm 0.09 \times 10^{-2}$ (2.52×)	$\mathbf{7.68} \pm 0.29 \times 10^{-3}$ (2.11×)	$\mathbf{5.24} \pm 0.14 \times 10^{-3}$ (1.55×)	$\underline{4.13} \pm 0.08 \times 10^{-3}$ (1.22×)	$3.95 \pm 0.10 \times 10^{-3}$ (1.00×)
	PICore (el2n)	$\underline{1.81} \pm 0.08 \times 10^{-2}$ (5.05×)	$\mathbf{1.12} \pm 0.07 \times 10^{-2}$ (3.30×)	$\underline{8.07} \pm 0.33 \times 10^{-3}$ (2.53×)	$\underline{5.49} \pm 0.08 \times 10^{-3}$ (1.68×)	$\mathbf{4.07} \pm 0.10 \times 10^{-3}$ (1.26×)	$3.95 \pm 0.10 \times 10^{-3}$ (1.00×)
UNO	Random	$\underline{2.92} \pm 0.05 \times 10^{-2}$ (4.99×)	$2.55 \pm 0.05 \times 10^{-2}$ (3.34×)	$2.25 \pm 0.05 \times 10^{-2}$ (2.55×)	$1.83 \pm 0.03 \times 10^{-2}$ (1.70×)	$1.58 \pm 0.01 \times 10^{-2}$ (1.27×)	$1.49 \pm 0.04 \times 10^{-2}$ (1.00×)
	Active Learning	$5.42 \pm 0.41 \times 10^{-2}$ (5.01×)	$4.12 \pm 0.11 \times 10^{-2}$ (3.30×)	$3.62 \pm 0.20 \times 10^{-2}$ (2.51×)	$3.03 \pm 0.23 \times 10^{-2}$ (1.68×)	$2.51 \pm 0.13 \times 10^{-2}$ (1.26×)	$1.49 \pm 0.04 \times 10^{-2}$ (1.00×)
	Supervised (gradmatch)	$2.93 \pm 0.10 \times 10^{-2}$ (3.77×)	$\underline{2.42} \pm 0.06 \times 10^{-2}$ (2.77×)	$\underline{2.08} \pm 0.05 \times 10^{-2}$ (2.23×)	$\underline{1.73} \pm 0.03 \times 10^{-2}$ (1.59×)	$\mathbf{1.54} \pm 0.02 \times 10^{-2}$ (1.23×)	$1.49 \pm 0.04 \times 10^{-2}$ (1.00×)
	PICore (graNd)	$\mathbf{2.84} \pm 0.05 \times 10^{-2}$ (5.05×)	$\mathbf{2.36} \pm 0.05 \times 10^{-2}$ (3.33×)	$\mathbf{2.06} \pm 0.04 \times 10^{-2}$ (2.52×)	$\mathbf{1.72} \pm 0.05 \times 10^{-2}$ (1.69×)	$\underline{1.57} \pm 0.03 \times 10^{-2}$ (1.26×)	$1.49 \pm 0.04 \times 10^{-2}$ (1.00×)

Table 3: Darcy NRMSE at resolution 64

Operator	Method	20.0%	30.0%	40.0%	60.0%	80.0%	100.0%
FNO	Random	$1.34 \pm 0.03 \times 10^{-1}$ (5.00×)	$1.15 \pm 0.01 \times 10^{-1}$ (3.36×)	$9.99 \pm 0.11 \times 10^{-2}$ (2.53×)	$7.94 \pm 0.04 \times 10^{-2}$ (1.69×)	$7.07 \pm 0.16 \times 10^{-2}$ (1.27×)	$6.18 \pm 0.09 \times 10^{-2}$ (1.00×)
	Active Learning	$2.01 \pm 0.16 \times 10^{-1}$ (4.99×)	$1.58 \pm 0.08 \times 10^{-1}$ (3.31×)	$1.25 \pm 0.06 \times 10^{-1}$ (2.50×)	$8.94 \pm 0.33 \times 10^{-2}$ (1.66×)	$7.19 \pm 0.25 \times 10^{-2}$ (1.25×)	$6.18 \pm 0.09 \times 10^{-2}$ (1.00×)
	Supervised (el2n)	$\underline{1.26} \pm 0.01 \times 10^{-1}$ (1.98×)	$\mathbf{1.07} \pm 0.007 \times 10^{-1}$ (1.76×)	$\mathbf{9.43} \pm 0.09 \times 10^{-2}$ (1.59×)	$\underline{7.83} \pm 0.18 \times 10^{-2}$ (1.33×)	$\mathbf{6.59} \pm 0.09 \times 10^{-2}$ (1.14×)	$6.18 \pm 0.09 \times 10^{-2}$ (1.00×)
	PICore (el2n)	$\mathbf{1.25} \pm 0.02 \times 10^{-1}$ (5.00×)	$\underline{1.12} \pm 0.02 \times 10^{-1}$ (3.32×)	$\underline{9.44} \pm 0.12 \times 10^{-2}$ (2.50×)	$\mathbf{7.77} \pm 0.18 \times 10^{-2}$ (1.67×)	$\underline{6.84} \pm 0.18 \times 10^{-2}$ (1.25×)	$6.18 \pm 0.09 \times 10^{-2}$ (1.00×)
UNO	Random	$1.45 \pm 0.02 \times 10^{-1}$ (5.03×)	$1.22 \pm 0.03 \times 10^{-1}$ (3.37×)	$1.10 \pm 0.02 \times 10^{-1}$ (2.53×)	$9.23 \pm 0.22 \times 10^{-2}$ (1.69×)	$8.78 \pm 0.30 \times 10^{-2}$ (1.27×)	$7.57 \pm 0.13 \times 10^{-2}$ (1.00×)
	Active Learning	$1.87 \pm 0.14 \times 10^{-1}$ (5.04×)	$1.54 \pm 0.08 \times 10^{-1}$ (3.35×)	$1.27 \pm 0.06 \times 10^{-1}$ (2.52×)	$1.02 \pm 0.03 \times 10^{-1}$ (1.68×)	$8.63 \pm 0.19 \times 10^{-2}$ (1.26×)	$7.57 \pm 0.13 \times 10^{-2}$ (1.00×)
	Supervised (gradmatch)	$\mathbf{1.28} \pm 0.03 \times 10^{-1}$ (2.23×)	$\underline{1.14} \pm 0.01 \times 10^{-1}$ (1.93×)	$\underline{9.84} \pm 0.16 \times 10^{-2}$ (1.71×)	$\underline{8.60} \pm 0.11 \times 10^{-2}$ (1.38×)	$\underline{7.70} \pm 0.10 \times 10^{-2}$ (1.16×)	$7.57 \pm 0.13 \times 10^{-2}$ (1.00×)
	PICore (graNd)	$\mathbf{1.28} \pm 0.03 \times 10^{-1}$ (5.01×)	$\mathbf{1.12} \pm 0.01 \times 10^{-1}$ (3.33×)	$\mathbf{9.67} \pm 0.16 \times 10^{-2}$ (2.50×)	$\mathbf{8.42} \pm 0.14 \times 10^{-2}$ (1.67×)	$\mathbf{7.61} \pm 0.11 \times 10^{-2}$ (1.25×)	$7.57 \pm 0.13 \times 10^{-2}$ (1.00×)

PICore significantly improves training efficiency through reduced simulation costs. Across four representative PDE datasets—Advection, Burgers, Darcy, and Navier-Stokes Incompressible—PICore consistently reduces the total training time by cutting down expensive simulation-based annotation. These efficiency gains become especially significant as the complexity of the PDE increases: Across the four datasets, PICore achieves average training time reductions of 0.9%, 9.8%, 30.1%, and 78.0% compared to supervised coreset selection, calculated by averaging the relative acceleration improvements at each selection percentage (20%, 30%, 40%, 60%, and 80%). For example, at a 20% coreset size, PICore achieves a 5.01× speedup on Darcy Flow (vs. 2.24× for supervised methods) and a 5.00× speedup on Navier-Stokes (vs. 1.14×) using UNO.

Table 4: Navier Stokes Incompressible NRMSE at resolution 64

Operator	Method	20.0%	30.0%	40.0%	60.0%	80.0%	100.0%
FNO	Random	$2.74 \pm 0.45 \times 10^{-1}$ (5.00×)	$5.59 \pm 0.80 \times 10^{-2}$ (3.34×)	$1.33 \pm 0.03 \times 10^{-2}$ (2.50×)	$9.06 \pm 0.24 \times 10^{-3}$ (1.67×)	$6.87 \pm 0.13 \times 10^{-3}$ (1.25×)	$5.66 \pm 0.11 \times 10^{-3}$ (1.00×)
	Active Learning	**9.32** $\pm 6.96 \times 10^{-2}$ (5.00×)	$2.16 \pm 0.66 \times 10^{-2}$ (3.33×)	$1.27 \pm 0.03 \times 10^{-2}$ (2.50×)	**7.86** $\pm 0.14 \times 10^{-3}$ (1.67×)	**6.24** $\pm 0.21 \times 10^{-3}$ (1.25×)	$5.66 \pm 0.11 \times 10^{-3}$ (1.00×)
	Supervised (el2n)	$\underline{9.57} \pm 3.87 \times 10^{-2}$ (1.05×)	**1.75** $\pm 0.07 \times 10^{-2}$ (1.05×)	**1.18** $\pm 0.04 \times 10^{-2}$ (1.04×)	$\underline{7.94} \pm 0.18 \times 10^{-3}$ (1.03×)	$\underline{6.28} \pm 0.16 \times 10^{-3}$ (1.01×)	$5.66 \pm 0.11 \times 10^{-3}$ (1.00×)
	PICore (graNd)	$1.12 \pm 0.45 \times 10^{-1}$ (5.00×)	$\underline{1.81} \pm 0.12 \times 10^{-2}$ (3.33×)	$\underline{1.23} \pm 0.05 \times 10^{-2}$ (2.50×)	$8.00 \pm 0.23 \times 10^{-3}$ (1.67×)	$6.34 \pm 0.14 \times 10^{-3}$ (1.25×)	$5.66 \pm 0.11 \times 10^{-3}$ (1.00×)
UNO	Random	$2.72 \pm 0.04 \times 10^{-2}$ (5.02×)	$2.24 \pm 0.02 \times 10^{-2}$ (3.34×)	$1.95 \pm 0.01 \times 10^{-2}$ (2.51×)	$1.61 \pm 0.006 \times 10^{-2}$ (1.67×)	**1.38** $\pm 0.004 \times 10^{-2}$ (1.25×)	$1.24 \pm 0.004 \times 10^{-2}$ (1.00×)
	Active Learning	$2.91 \pm 0.05 \times 10^{-2}$ (5.00×)	$2.36 \pm 0.03 \times 10^{-2}$ (3.33×)	$2.06 \pm 0.02 \times 10^{-2}$ (2.50×)	$1.61 \pm 0.01 \times 10^{-2}$ (1.67×)	$1.40 \pm 0.009 \times 10^{-2}$ (1.25×)	$1.24 \pm 0.004 \times 10^{-2}$ (1.00×)
	Supervised (el2n)	$\underline{2.60} \pm 0.02 \times 10^{-2}$ (1.14×)	**2.19** $\pm 0.02 \times 10^{-2}$ (1.12×)	$\underline{1.93} \pm 0.01 \times 10^{-2}$ (1.10×)	**1.57** $\pm 0.009 \times 10^{-2}$ (1.07×)	**1.38** $\pm 0.010 \times 10^{-2}$ (1.03×)	$1.24 \pm 0.004 \times 10^{-2}$ (1.00×)
	PICore (gradmatch)	**2.59** $\pm 0.03 \times 10^{-2}$ (5.00×)	$\underline{2.20} \pm 0.009 \times 10^{-2}$ (3.33×)	**1.92** $\pm 0.009 \times 10^{-2}$ (2.50×)	$\underline{1.60} \pm 0.005 \times 10^{-2}$ (1.67×)	$1.40 \pm 0.007 \times 10^{-2}$ (1.25×)	$1.24 \pm 0.004 \times 10^{-2}$ (1.00×)

As shown in Tables 5, 6, 7, and 8, the relative contributions of training and data generation speedups vary by dataset difficulty. For simpler datasets such as Advection and Burgers, efficiency gains are driven primarily by reductions in training time. For example, Advection achieves a 79.7% improvement in training time but only a 1.47% improvement in data generation time at the 20% coreset level. In contrast, for more challenging datasets, the impact of training time reductions diminishes, while reductions in data generation time play a more significant role in overall efficiency gains. These results show that PICore scales well to high-dimensional scientific problems where data annotation costs dominate training.

PICore matches supervised coreset methods in test accuracy at reduced data budgets. Despite significant efficiency gains, PICore remains competitive with supervised baselines in test NRMSE. At a 20% coreset size, it achieves 3.46×10^{-2} for Advection (FNO) and 2.84×10^{-2} for Burgers (UNO), close to the supervised values of 3.42×10^{-2} and 2.93×10^{-2}. This trend holds across coreset sizes, with many cases showing PICore outperforming supervised selection. Not all selection algorithms perform equally well, however, CRAIG and AdaCore often yield higher NRMSE due to convexity assumptions and Hessian approximations. Thus, we find that results typically favor GradMatch, GraNd, or EL2N for both PICore and supervised selection.

Coreset Selection methods outperform Random and Active Learning baselines on most datasets. Random subset selection consistently underperforms relative to both PICore and subset selection, and this difference increases as we increase the complexity of the dataset and decrease the selection percentage. For example, random selection has an nRMSE of 2.74×10^{-1} on the Navier Stokes Incompressible dataset at a 20% coreset selection percentage, where as PICore has an nRMSE of 1.12×10^{-2} and Supervised Coreset Selection has an nRMSE of 9.57×10^{-2}. This shows that using PDE specific information (either supervised loss or the physics informed loss) is necessary to achieve a more accurate solution with less data. Interestingly, we see that active learning outperforms the PICore on the Advection dataset with UNO on medium coreset selection percentages (40-60%) and on the Navier Stokes Incompressible dataset with FNO. However, it is much worse on all other dataset and model combinations by a considerable margin.

There is a tradeoff between efficiency and absolute test accuracy. While PICore offers strong performance and efficiency, one tradeoff is that the absolute test accuracy relative to training on 100% of the data is lower. For example, on the Advection dataset with FNO, the 100% training baseline yields an NRMSE of 2.13×10^{-2}, while PICore at 20% yields 3.77×10^{-2}. However, this is an inherent tradeoff for all coreset selection algorithms, as the selected coreset simply contains less information for training. Additionally, this is not specific to PICore, as similar reductions in accuracy hold for supervised coreset selection. In practice, one may want to select a higher selection percentage, such as 40%, which would yield higher accuracy (2.69×10^{-2}) while still maintaining a competitive efficiency gain ($2.54\times$).

7 Limitations and Future Work

One limitation of PICore is its reliance on existing coreset selection algorithms. Methods like CRAIG and AdaCore, designed for convex losses and image classification, may perform suboptimally on

complex, non-convex PDE datasets, especially at low selection ratios. For instance, using Hutchinson Hessian approximations on the last layer in AdaCore often yields poorer accuracies. Thus, we recommend that practitioners use GradMatch, EL2N, or GraNd with PICore, since they make fewer data and model assumptions. Future work could develop coreset algorithms tailored for neural operators and extend PICore to multi-resolution or irregular geometries to improve generalization while preserving efficiency.

8 Conclusion

In this work, we introduced PICore, a physics-informed unsupervised coreset selection framework designed to enhance the data efficiency of neural operator training. By leveraging the physics-informed loss to identify the most informative samples without requiring labeled data, PICore significantly reduces both the computational cost of numerical simulations and the time required for training. Our experiments across four PDE benchmarks demonstrate that PICore achieves competitive accuracy while reducing training costs by up to 78% compared to supervised coreset selection methods. Although PICore inherits some limitations from existing selection methods, we believe its ability to reduce labeling costs and accelerate training makes it a promising tool for large-scale scientific machine learning.

References

Pankaj K Agarwal, Sariel Har-Peled, Kasturi R Varadarajan, et al. Geometric approximation via coresets. *Combinatorial and computational geometry*, 52(1):1–30, 2005.

William H. Beluch, Tim Genewein, Andreas Nurnberger, and Jan M. Kohler. The power of ensembles for active learning in image classification. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9368–9377, 2018. doi: 10.1109/CVPR.2018.00976.

Boris Bonev, Thorsten Kurth, Christian Hundt, Jaideep Pathak, Maximilian Baust, Karthik Kashinath, and Anima Anandkumar. Spherical fourier neural operators: Learning stable dynamics on the sphere. In *International conference on machine learning*, pages 2806–2823. PMLR, 2023.

Xiaofeng Cao and Ivor W. Tsang. Shattering distribution for active learning. *IEEE Transactions on Neural Networks and Learning Systems*, 33(1):215–228, 2022. doi: 10.1109/TNNLS.2020.3027605.

Wuyang Chen, Jialin Song, Pu Ren, Shashank Subramanian, Dmitriy Morozov, and Michael W Mahoney. Data-efficient operator learning via unsupervised pretraining and in-context learning. *Advances in Neural Information Processing Systems*, 37:6213–6245, 2024.

Yutian Chen, Max Welling, and Alex Smola. Super-samples from kernel herding. *arXiv preprint arXiv:1203.3472*, 2012.

N.J. Cyrus, R.E. Fulton, United States. National Aeronautics, Space Administration, and Langley Research Center. *Accuracy Study of Finite Difference Methods*. NASA technical note. National Aeronautics and Space Administration, 1968. URL `https://books.google.com/books?id=zMSFxfAasQMC`.

Kenneth Eriksson and Claes Johnson. Adaptive finite element methods for parabolic problems iv: Nonlinear problems. *SIAM Journal on Numerical Analysis*, 32(6):1729–1749, 1995. doi: 10.1137/0732078. URL `https://doi.org/10.1137/0732078`.

Bin Gu, Zhou Zhai, Cheng Deng, and Heng Huang. Efficient active learning by querying discriminative and representative samples and fully exploiting unlabeled data. *IEEE Transactions on Neural Networks and Learning Systems*, 32(9):4111–4122, 2021. doi: 10.1109/TNNLS.2020.3016928.

Chengcheng Guo, Bo Zhao, and Yanbing Bai. Deepcore: A comprehensive library for coreset selection in deep learning. In *International Conference on Database and Expert Systems Applications*, pages 181–195. Springer, 2022.

Animesh Gupta, Irtiza Hasan, Dilip K Prasad, and Deepak K Gupta. Data-efficient training of cnns and transformers with coresets: A stability perspective. *arXiv preprint arXiv:2303.02095*, 2023.

Daniel Haimovich, Dima Karamshuk, Fridolin Linder, Niek Tax, and Milan Vojnovic. On the convergence of loss and uncertainty-based active learning algorithms. *Advances in Neural Information Processing Systems*, 37:122770–122810, 2024.

AmirPouya Hemmasian and Amir Barati Farimani. Pretraining a neural operator in lower dimensions. *arXiv preprint arXiv:2407.17616*, 2024.

Claes Johnson. *Numerical solution of partial differential equations by the finite element method*. Cambridge University Press, Cambridge, England, January 1988.

Krishnateja Killamsetty, Sivasubramanian Durga, Ganesh Ramakrishnan, Abir De, and Rishabh Iyer. Grad-match: Gradient matching based data subset selection for efficient deep model training. In *International Conference on Machine Learning*, pages 5464–5474. PMLR, 2021a.

Krishnateja Killamsetty, Durga Sivasubramanian, Ganesh Ramakrishnan, and Rishabh Iyer. Glister: Generalization based data subset selection for efficient and robust learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 8110–8118, 2021b.

Krishnateja Killamsetty, Xujiang Zhao, Feng Chen, and Rishabh Iyer. Retrieve: Coreset selection for efficient and robust semi-supervised learning. *Advances in neural information processing systems*, 34:14488–14501, 2021c.

Yeachan Kim and Bonggun Shin. In defense of core-set: A density-aware core-set selection for active learning. In *Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining*, pages 804–812, 2022.

Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces with applications to pdes. *Journal of Machine Learning Research*, 24(89):1–97, 2023.

Nikola B. Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew M. Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces, 2021. URL `https://arxiv.org/abs/2108.08481`.

Randall J. LeVeque. *Finite Volume Methods for Hyperbolic Problems*. Cambridge Texts in Applied Mathematics. Cambridge University Press, 2002.

Dongyuan Li, Zhen Wang, Yankai Chen, Renhe Jiang, Weiping Ding, and Manabu Okumura. A survey on deep active learning: Recent advances and new frontiers. *IEEE Transactions on Neural Networks and Learning Systems*, 2024a.

Shibo Li, Xin Yu, Wei Xing, Robert Kirby, Akil Narayan, and Shandian Zhe. Multi-resolution active learning of fourier neural operators. In *International Conference on Artificial Intelligence and Statistics*, pages 2440–2448. PMLR, 2024b.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.

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

Zongyi Li, Miguel Liu-Schiaffini, Nikola Kovachki, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Learning dissipative dynamics in chaotic systems. *arXiv preprint arXiv:2106.06898*, 2021b.

Zongyi Li, Miguel Liu-Schiaffini, Nikola Borislavov Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Learning chaotic dynamics in dissipative systems. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL `https://openreview.net/forum?id=1C36tFZn7sR`.

Zongyi Li, Hongkai Zheng, Nikola Kovachki, David Jin, Haoxuan Chen, Burigede Liu, Kamyar Azizzadenesheli, and Anima Anandkumar. Physics-informed neural operator for learning partial differential equations. *ACM/JMS Journal of Data Science*, 1(3):1–27, 2024c.

Shang Liu and Xiaocheng Li. Understanding uncertainty sampling. *arXiv preprint arXiv:2307.02719*, 2023.

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

Baharan Mirzasoleiman, Jeff Bilmes, and Jure Leskovec. Coresets for data-efficient training of machine learning models. In *International Conference on Machine Learning*, pages 6950–6960. PMLR, 2020.

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

Mansheej Paul, Surya Ganguli, and Gintare Karolina Dziugaite. Deep learning on a data diet: Finding important examples early in training. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 20596–20607. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/ac56f8fe9eea3e4a365f29f0f1957c55-Paper.pdf.

Omead Pooladzandi, David Davini, and Baharan Mirzasoleiman. Adaptive second order coresets for data-efficient machine learning. In *International Conference on Machine Learning*, pages 17848–17869. PMLR, 2022.

Md Ashiqur Rahman, Zachary E Ross, and Kamyar Azizzadenesheli. U-NO: U-shaped neural operators. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=j3oQF9coJd.

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

Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. *arXiv preprint arXiv:1708.00489*, 2017.

Makoto Takamoto, Timothy Praditia, Raphael Leiteritz, Dan MacKinlay, Francesco Alesiani, Dirk Pflüger, and Mathias Niepert. PDEBench: An Extensive Benchmark for Scientific Machine Learning. In *36th Conference on Neural Information Processing Systems (NeurIPS 2022) Track on Datasets and Benchmarks*, 2022. URL https://arxiv.org/abs/2210.07182.

Hewei Tang, Qingkai Kong, and Joseph P Morris. Multi-fidelity fourier neural operator for fast modeling of large-scale geological carbon storage. *Journal of Hydrology*, 629:130641, 2024.

Alasdair Tran, Alexander Mathews, Lexing Xie, and Cheng Soon Ong. Factorized fourier neural operators. *arXiv preprint arXiv:2111.13802*, 2021.

Kai Wei, Rishabh Iyer, and Jeff Bilmes. Submodularity in data subset selection and active learning. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1954–1963, Lille, France, 07–09 Jul 2015. PMLR. URL https://proceedings.mlr.press/v37/wei15.html.

Max Welling. Herding dynamical weights to learn. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 1121–1128, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605585161. doi: 10.1145/1553374.1553517. URL https://doi.org/10.1145/1553374.1553517.

Yazhou Yang and Marco Loog. To actively initialize active learning. *Pattern Recognition*, 131: 108836, 2022. ISSN 0031-3203. doi: https://doi.org/10.1016/j.patcog.2022.108836. URL https://www.sciencedirect.com/science/article/pii/S003132032200317X.

Zhewei Yao, Peng Xu, Farbod Roosta-Khorasani, and Michael W. Mahoney. Inexact non-convex newton-type methods, 2018. URL https://arxiv.org/abs/1802.06925.

Guang Zhao, Edward Dougherty, Byung-Jun Yoon, Francis Alexander, and Xiaoning Qian. Efficient active learning for gaussian process classification by error reduction. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 9734–9746. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/50d2e70cdf7dd05be85e1b8df3f8ced4-Paper.pdf.

A PDE Datasets

For our experiments, we use several differential equation training sets to evaluate our algorithm. Each of these is used at an input grid resolution of 64. For the Advection, Burgers, and Darcy Flow equations, we generate datasets using code provided by Takamoto et al. [2022]. For the Navier-Stokes Incompressible equation dataset, we generate data from Li et al. [2020]. Each dataset has 1000 generated trajectories, with 900 that can be used for training (varying based on the coreset selection percentage) and 100 for testing, which is comparable to existing neural operator literature [Li et al., 2021a, 2024c]. We generate 20 timesteps forward for Advection and Burgers, and only 10 timesteps for the Navier Stokes Incompressible dataset due to GPU memory limits.

One challenge with using the physics-informed loss is computing the PDE residual $\mathcal{F}(\mathcal{G}_\theta(a), a)$. The residual requires computing derivatives of the neural operator with respect to the dimensional parameters, such as $\frac{\partial^2 \mathcal{G}_\theta}{\partial x \partial t}$. Li et al. [2024c] uses a function-wise differentiation method via Fourier differentiation to compute these values exactly, but this does not extend to a general class of neural operators. We also tried auto-differentiation methods, but these were highly computationally expensive, increasing the coreset selection time. Thus, we settled on simply using finite difference methods, which are efficient with linear time complexity in the input resolution.

A.1 Advection

We construct our dataset by numerically solving the linear advection equation on the periodic domain $(0, 1)$:

$$\partial_t u(t, x) + \beta\, \partial_x u(t, x) = 0, \quad t \in (0, 2],\ x \in (0, 1), \tag{13}$$

The initial condition is defined as a superposition of sinusoidal modes,

$$u_0(x) \ = \ \sum_{i=1}^{N} A_i\, \sin\big(k_i x + \phi_i\big), \quad k_i = \frac{2\pi n_i}{L_x}, \tag{14}$$

where each n_i is drawn uniformly from the range of integers from 1 to 8, N is the number of waves, and the amplitudes $A_i \in [0, 1]$ and phases $\phi_i \in (0, 2\pi)$ are chosen at random. After assembly of $u_0(x)$, we apply with 10% probability each a pointwise absolute-value operation or multiplication by a smooth window function.

A.2 Burger's Equation

We are interested in the one-dimensional viscous Burgers equation on the unit interval with periodic boundary conditions:

$$\partial_t u(t, x) + \partial_x\big(\tfrac{1}{2}\, u^2(t, x)\big) = \frac{\nu}{\pi}\, \partial_{xx} u(t, x), \quad x \in (0, 1),\ t \in (0, 2], \tag{15}$$

subject to the initial condition

$$u(0, x) = u_0(x), \quad x \in (0, 1). \tag{16}$$

Here $\nu > 0$ is a constant diffusion coefficient. We use the nondimensional Reynolds number

$$R \;=\; \frac{\pi\, u_L}{\nu},$$

where u_L is a characteristic velocity scale. In analogy with the Navier–Stokes equations, $R > 1$ indicates a regime dominated by nonlinear steepening and potential shock formation, whereas $R < 1$ corresponds to diffusion-dominated smooth dynamics.

A.3 Darcy Flow

We obtain the steady-state solution of Darcy's equation on the unit square by evolving a time-dependent problem until convergence. The target elliptic problem is

$$-\nabla\cdot\big(a(x)\,\nabla u(x)\big) = f(x), \qquad x \in (0,1)^2, \tag{17}$$

$$u(x) = 0, \qquad x \in \partial(0,1)^2, \tag{18}$$

where $a(x)$ is the spatially varying coefficient and $f(x) \equiv \beta$ is a constant forcing that scales the solution amplitude.

Rather than solving equation 17, we integrate the parabolic problem

$$\partial_t u(x,t) - \nabla\cdot\big(a(x)\,\nabla u(x,t)\big) = \beta, \quad x \in (0,1)^2,\ t > 0, \tag{19}$$

with an appropriate random-field initial condition and homogeneous Dirichlet boundary data. We use the strong form $\nabla \cdot (a\nabla u) - f$ for the residual as in Li et al. [2024c].

A.4 Navier-Stoker Equation

We consider the vorticity formulation on the periodic domain $(0, 1)^2$:

$$\partial_t\omega + u\cdot\nabla\omega = \nu\,\Delta\omega + f, \quad \nabla\cdot u = 0, \quad \omega(x,0) \sim \mathcal{N}\big(0,\ 7^{3/2}(-\Delta + 49I)^{-2.5}\big),$$

with forcing

$$f(x) = 0.1\big[\sin 2\pi(x_1 + x_2) + \cos 2\pi(x_1 + x_2)\big].$$

The solution is obtained on a 256×256 grid via a Fourier pseudospectral scheme: first, we solve $\Delta\psi = -\omega$ in Fourier space to recover the stream function ψ and velocity u, then compute the nonlinear advection term $u \cdot \nabla\omega$ in physical space with a $2/3$-dealiasing filter, and finally advance in time using Crank–Nicolson for diffusion coupled with an explicit update for the nonlinear term.

B Coreset Selection Algorithms

In this section, we provide an overview of the coreset selection algorithms used. All implementations are our own, but are based on Guo et al. [2022].

Adacore

AdaCore augments CRAIG with second–order curvature so that difficult, high–influence samples are favoured even when first–order gradients look similar. In practice we **estimate only the diagonal** of the Hessian with *10 Hutchinson probes* per mini-batch, then pre-condition the last-layer gradient $\nabla\ell_i$ by element-wise division. Similarities are computed on these pre-conditioned vectors and the same stochastic-greedy routine as CRAIG is applied. The extra cost is the time to compute the approximation by deriving multiplications of the Hessian and arbitrary vectors via the Hessian-Free method [Yao et al., 2018], the time of Hutchinson's method to find the diagonal, and the time to apply the diagonal to the gradients of the last layer.

EL2N

Our EL2N (Error L2-Norm) coreset selection method follows from the premise that samples that are most worthwhile for the model have the highest losses. EL2N conducts a full training pass, where for each minibatch x_i, we calculate the loss without reduction for each individual sample, and calculate the norm for x_i's loss vector. At the end of the epoch, we take the top k minibatches by loss norm and return them with equal weight.

CRAIG

CRAIG (**C**oresets for **A**ccelerating **I**ncremental **G**radient-descent) selects a weighted subset of size k whose gradients cover (i.e. represent) all per-example gradients. Let $g_i = \nabla_\theta \ell_i(\theta) \in \mathbb{R}^d$ be the gradient for example i. CRAIG finds a near optimal solution to the following problem.

$$A^* = \arg\min_{A \subset V} |S|, \ \sum_{n \in V} \min_{m \in S} \max_{\theta} ||g_n - g_m||$$

so every g_i is "covered" by its most similar selected gradient. CRAIG selects the smallest subset S such that every example gradient is close (in $\mathcal{L}_2$) to at least one gradient in S. We approximate the coverage objective with the stochastic-greedy algorithm applied to the pairwise Euclidean similarity matrix of last-layer gradients. Greedy (or stochastic-greedy) selection gives a $(1-1/e)$-approximation in finite similarity evaluations. After S is chosen, CRAIG sets integer weights

$$\gamma_j \ = \ \big|\{\, i : \arg\max_{m \in S} s_{im} = j\}\big|, \quad j \in S,$$

so the weighted coreset gradient $\sum_{j \in S} \gamma_j g_j$ closely matches the full gradient $\sum_{i=1}^{n} g_i$ at each optimisation step. In practice the method is applied to last-layer gradients to reduce dimensionality without degrading the approximation quality.

GradMatch

Let the last–layer per-example gradients be concatenated as $A = [g_1 \, g_2 \, \dots \, g_n] \in \mathbb{R}^{d \times n}$ and define the full-batch gradient $b = \frac{1}{n} \sum_{i=1}^{n} g_i$. GRADMATCH casts coreset selection as the sparse approximation problem.

$$\min_{x \in \mathbb{R}^n} \ \|Ax - b\|_2^2 \quad \text{s.t.} \quad \|x\|_0 \le k, \ x \ge 0.$$

OMP builds the weight vector x greedily. Starting with residual $r = b$ and empty support S: (i) choose the column $j^\star = \arg\max_{j \notin S} A_j^\top r$; (ii) add $j^\star$ to S; (iii) refit the coefficients by non-negative least squares $x_S = \arg\min_{x \ge 0} \|A_S x - b\|_2^2 + \lambda \|x\|_2^2$; (iv) update $r = b - A_S x_S$. The loop terminates after k selections, giving a coreset $S = \mathrm{supp}(x)$ with weights $\gamma_j = x_j$.

During training we replace the full loss by the weighted loss $\sum_{j \in S} \gamma_j \ell_j \big/ \sum_{j \in S} \gamma_j$, ensuring the mini-batch gradient of the coreset closely follows the full-batch gradient throughout optimisation.

GraNd

GraNd is similar to EL2N, but simply orders samples by the norm of their individual gradients and keeps the top k. We piggy-back on the same per-sample gradient collection already needed for CRAIG/GradMatch, but *stop after the first backward call*. We can rapidly sort these norms on the CPU, and use the selected indices for our coreset.

C Additional Results

C.1 Component-wise PICore Acceleration

Coreset %	FNO			UNO		
	Train	Data	Warm-up	Train	Data	Warm-up
20.0%	+79.7%	+1.4%	-0.9%	+80.4%	+0.8%	-4.5%
30.0%	+69.7%	+1.2%	-1.4%	+70.7%	+0.7%	-4.5%
40.0%	+61.3%	+1.1%	-1.8%	+61.7%	+0.6%	-4.5%
60.0%	+42.8%	+0.7%	-2.8%	+43.2%	+0.4%	-4.5%
80.0%	+24.2%	+0.4%	-3.7%	+24.2%	+0.2%	-4.5%

Table 5: Advection PICore component speedup.

Coreset %	FNO			UNO		
	Train	Data	Warm-up	Train	Data	Warm-up
20.0%	+70.3%	+10.7%	-0.8%	+74.4%	+6.6%	-4.5%
30.0%	+61.5%	+9.4%	-1.3%	+65.4%	+5.8%	-4.5%
40.0%	+54.0%	+8.1%	-1.6%	+57.1%	+5.0%	-4.5%
60.0%	+37.7%	+5.4%	-2.4%	+39.9%	+3.3%	-4.5%
80.0%	+21.4%	+2.7%	-3.3%	+22.3%	+1.7%	-4.5%

Table 6: Burgers PICore component speedup.

Coreset %	FNO			UNO		
	Train	Data	Warm-up	Train	Data	Warm-up
20.0%	+50.2%	+30.4%	-0.6%	+55.9%	+24.8%	-3.3%
30.0%	+44.2%	+26.6%	-0.9%	+49.2%	+21.7%	-3.3%
40.0%	+38.4%	+22.8%	-1.2%	+42.7%	+18.6%	-3.3%
60.0%	+26.7%	+15.2%	-1.8%	+29.7%	+12.4%	-3.3%
80.0%	+14.9%	+7.6%	-2.4%	+16.5%	+6.2%	-3.3%

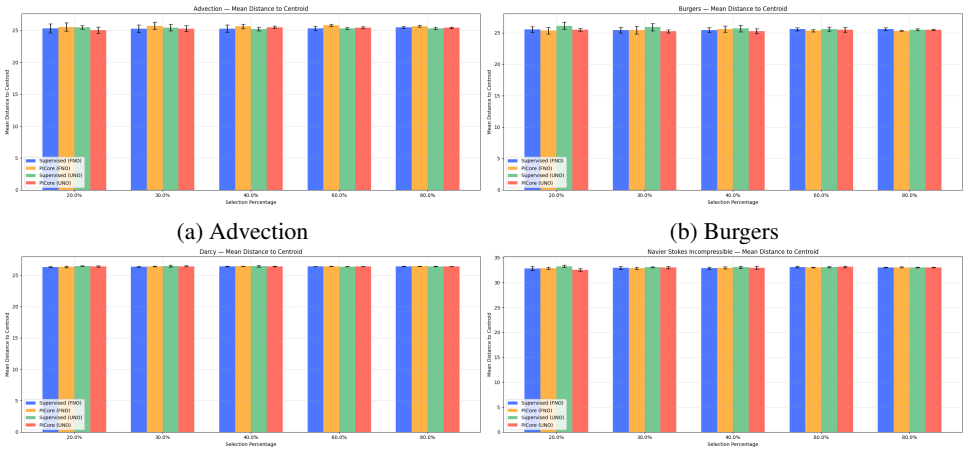
Table 7: Darcy PICore component speedup.

Coreset %	FNO			UNO		
	Train	Data	Warm-up	Train	Data	Warm-up
20.0%	+5.2%	+74.9%	-0.1%	+12.7%	+67.5%	-0.8%
30.0%	+4.5%	+65.5%	-0.1%	+11.2%	+59.0%	-0.8%
40.0%	+3.9%	+56.2%	-0.1%	+9.7%	+50.6%	-0.8%
60.0%	+2.7%	+37.4%	-0.2%	+6.7%	+33.7%	-0.8%
80.0%	+1.5%	+18.7%	-0.2%	+3.7%	+16.9%	-0.8%

Table 8: Navier Stokes PICore component speedup.

C.2 Comparison between Supervised Coreset Selection and PICore

To better understand the differences between supervised coreset selection and PICore, we analyze how well each method covers the input space by computing the average distance from coreset points to their centroid, which serves as a proxy for spread or diversity. We compute this distance with respect to the $\|\cdot\|_{L^2(\Omega)}$ norm, where the centroid is the average data point element-wise and the average distance is the average norm between the centroid and the selected data points in the coreset. As shown in Figure 2, this distance is nearly identical across datasets and neural operators (FNO and UNO), with overlapping standard error bars with differences decreasing as the PDE complexity increases (Advection to Navier Stokes). This suggests that PICore selects coresets that are as well-distributed as those from supervised methods, despite not using labeled data. The comparable coverage indicates that differences in downstream performance likely arise from the type of points selected rather than their spatial distribution.



(a) Advection (b) Burgers

(c) Darcy (d) Navier Stokes

Figure 2: Average centroid distances across datasets for FNO and UNO.

C.3 Unsupervised Coreset Selection

We compare PICore to three unsupervised coreset selection methods: k-means clustering, cosine similarity, and Herding [Chen et al., 2012]. For k-means clustering we use $k = \beta N$ clusters, and choose the data points closest to those clusters. For cosine similarity, we evaluate the cosine similarity between all pairs of points, and perform greedy selection to choose the coreset. We report direct comparison of the test NRMSE for both methods in Figures 3, 4, 5, and 6 in Section **??**. The results show that PICore consistently matches or outperforms the unsupervised baselines across all tested coreset sizes (20% to 80%) and neural operator architectures (FNO and UNO). For instance, on the Advection dataset at 20% coreset size, PICore with the EL2N algorithm achieves a test NRMSE of 3.29×10^{-2}, outperforming cosine similarity 3.39×10^{-2} and herding 3.46×10^{-2}. Similar patterns are observed on the other datasets, indicating that PICore's selection strategy generalizes well across both time-dependent and stationary PDEs compared to other unsupervised coreset selection strategies. We also note that these trends hold across neural operator architectures, with PICore outperforming unsupervised methods with both FNO and UNO architectures. While FNO does consistently outperform UNO across datasets (except Navier Stokes Incompressible), this is due to the architecture differences and not due to PICore (as shown by the increase in NRMSE for UNO on the non-coreset baseline).

These results highlight the advantage of incorporating PDE-specific information into the subset selection process. While clustering and similarity-based approaches may cover the input space evenly or preserve diversity, they do not necessarily target the data points where the model struggles. In contrast, PICore explicitly focuses on where the model's performance is likely to decrease by using the PDE's residual, resulting in improved predictive accuracy under minimal training data.

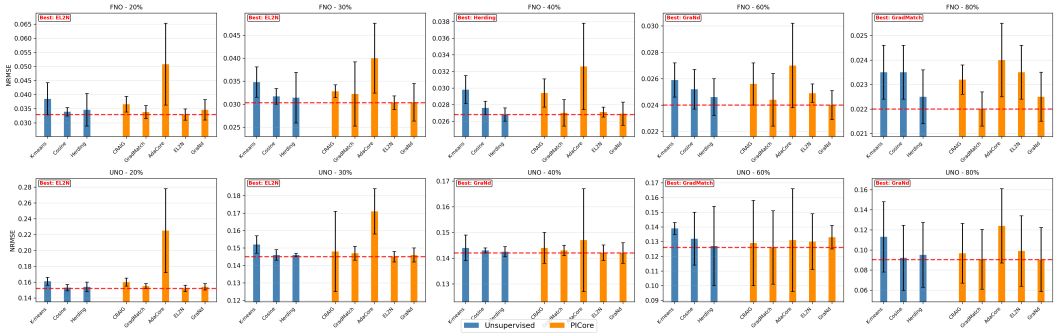


Figure 3: Test NRMSE on the Advection dataset at resolution 64 across varying coreset percentages (20%–100%) between unsupervised and PICore-based coreset selection methods using both FNO and UNO architectures.

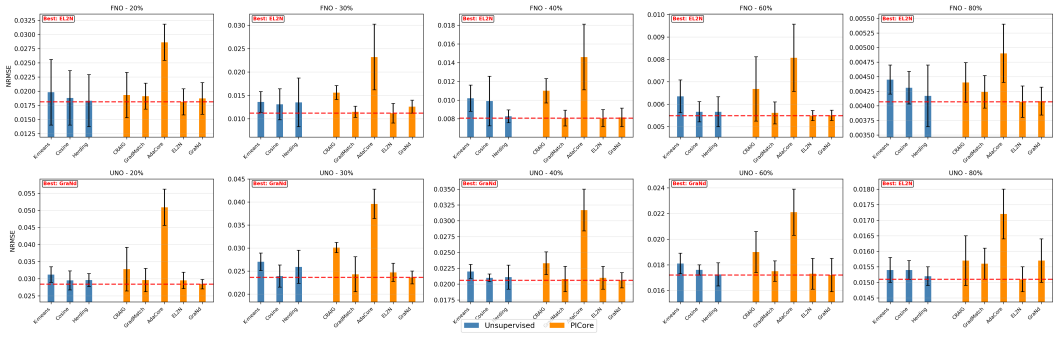


Figure 4: Test NRMSE on the Burgers dataset at resolution 64 across varying coreset percentages (20%–100%) between unsupervised and PICore-based coreset selection methods using both FNO and UNO architectures.

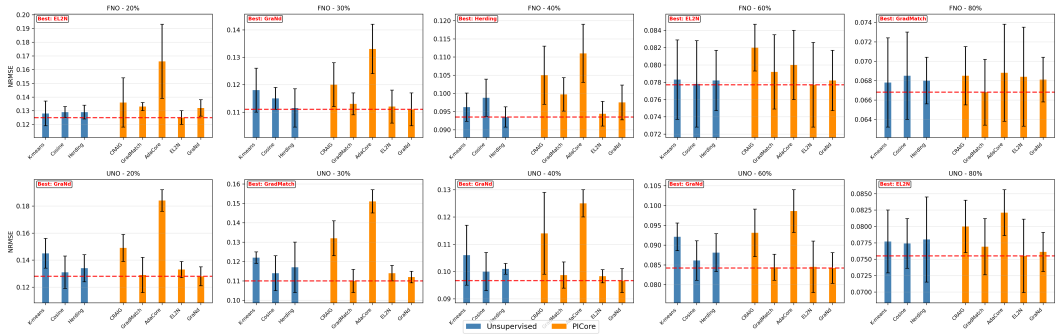


Figure 5: Test NRMSE on the Darcy dataset at resolution 64 across varying coreset percentages (20%–100%) between unsupervised and PICore-based coreset selection methods using both FNO and UNO architectures.

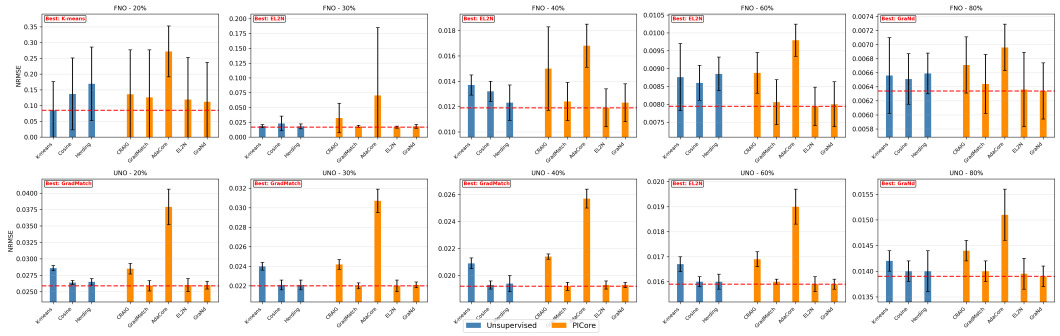


Figure 6: Test NRMSE on the Navier Stokes Incompressible dataset at resolution 64 across varying coreset percentages (20%–100%) between unsupervised and PICore-based coreset selection methods using both FNO and UNO architectures.

C.4 Convergence of Coreset Selection vs Active Learning

A key distinction between coreset selection and active learning lies in their approach to data selection, which in turn affects their convergence speed. This iterative nature can be suboptimal, as it can lead to selecting redundant data points [Li et al., 2024a]. While some works have shown superior convergence of active learning methods [Haimovich et al., 2024], these are under specific optimizer settings and in easier image classification domains.

Our empirical results largely validate this viewpoint, demonstrating that PICore's single-shot selection generally leads to better subset selection that converges faster than active learning baselines. Figures 7 and 8 show the training loss convergence of PICore's coreset selection methods compared to the active learning baseline. For both FNO and UNO, the active learning method converges much slower by 2-3×. The difference in loss convergence decreases for more complex datasets such as Navier Stokes, but this is due to learning a larger FNO-3D / UNO-3D model than due to the subset selection method itself.

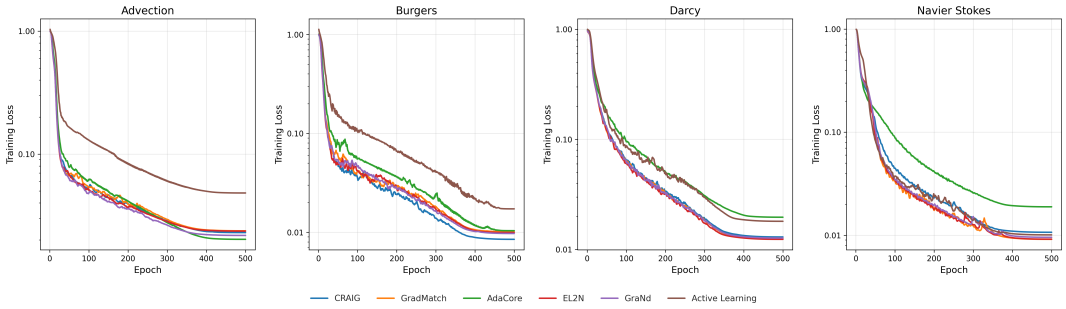


Figure 7: Training loss convergence of coreset selection methods in comparison to active learning using FNO at a 20% selection ratio.

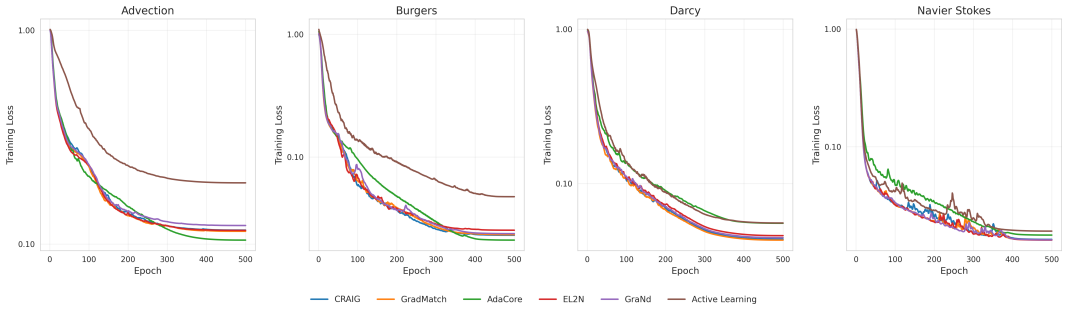


Figure 8: Training loss convergence of coreset selection methods in comparison to active learning using UNO at a 20% selection ratio.