In-Context Multi-Operator Learning with DeepOSets

Shao-Ting Chiu*
Department of Electrical
and Computer Engineering
Texas A&M University
College Station, TX 77843
stchiu@tamu.edu

Aditya Nambiar*
Department of Mathematics
Texas A&M University
College Station, TX 77843
aditnambiar@tamu.edu

Ali Syed
Department of Mathematics
Texas A&M University
College Station, TX 77843
saabidi1@tamu.edu

Jonathan W. Siegel
Department of Mathematics
Texas A&M University
College Station, TX 77843
jwsiegel@tamu.edu

Ulisses Braga-Neto
Department of Electrical
and Computer Engineering
Texas A&M University
College Station, TX 77843
ulisses@tamu.edu

Abstract

An important application of neural networks in scientific computing is non-linear operator learning. In this framework, a neural network is trained to fit a non-linear map between infinite dimensional spaces, for example, the solution operator of ordinary and partial differential equations. Recently, inspired by the discovery of in-context learning for large language models, an even more ambitious paradigm has been explored, called multi-operator learning. In this approach, a neural network is trained to learn many different operators at the same time. At inference time, the network is passed a small number of example input and output pairs to disambiguate the desired operator. In this work, we provide a precise mathematical formulation of the multi-operator learning problem, propose a parameter-efficient, attention-free architecture, called DeepOSets, and prove its universality in the in-context multi-operator learning task. We perform experiments demonstrating the efficacy of DeepOSets on several benchmarks, including Burgers' equation, 1D and 2D Darcy flow, and a variety of initial-value and boundary-value differential equations. In a comparison with the transformer-based ICON model on the 1D Darcy problem, under a small training budget (10k steps), DeepOSets attained lower error at every context size with 19% of the parameters, while with an extended budget (100k steps), ICON attains 26–41% lower error, but at the cost of roughly 218$\times$ more training time, 12$\times$ higher inference latency, and 5.3$\times$ more parameters.

1 Introduction

Learning non-linear operators from data using neural networks has become a foundational application of deep learning in scientific computing [16, 15]. In learning the solution operator of a parametric PDE from sample data, one is not interested in merely obtaining a single solution for a single PDE, but instead learning the mapping between parameter and solution, so that new solutions corresponding to new parameter values can be obtained quickly and without further training. Furthermore, both the parameter and the solution are typically functions in infinite-dimensional spaces. The trained operator allows quick execution in many-query problems, such as design optimization,

*Equal contribution.

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

uncertainty quantification, or real-time processing, which is infeasible to do with expensive high-fidelity solvers.

On the other hand, many problems of interest in science and engineering involve solving complex PDEs with various unknown parameters or terms, in which case one has multiple PDE solution operators, and the neural network model needs to be trained on data from a mixture of PDEs. This is a multi-operator learning problem, also known as a PDE foundation model[20, 11, 30]. To disambiguate the correct operator during inference time, one needs additional data, which could be used to fine-tune the neural network, or could be presented in a prompt, in which case there are no weight updates. The latter approach is known as in-context learning (ICL). At this time, In-Context Operator Networks (ICON) [25], and its subsequent variants [28, 27, 31], is the best-known approach to in-context learning of PDE solution operators. ICON is based on a GPT-style architecture and thus has quadratic complexity in prompt length, due to the dot-product self-attention mechanism. This can introduce significant computational overhead and loss of accuracy with very long prompts.

A theoretical study of in-context multi-operator learning remains incomplete. In particular, it is not known whether existing architectures such as ICON are universal approximators. One key difficulty is that the lack of a precise mathematical formulation of the multi-operator learning problem. In this paper, we provide such a mathematical formulation, by framing the problem as learning a *compact set of operators* between Banach spaces. Furthermore, we propose an alternative approach to in-context learning of PDE solution operators, which processes the prompt non-autoregressively and without using self-attention, being therefore much faster during both training and inference time. This approach is based on DeepOSets, a neural architecture that combines set learning via the DeepSets architecture [29] with the DeepONets neural operator architecture [16]. In DeepOSets, the DeepSets layer acts as an encoder of the prompt into a latent representation that is the input, along with the query, to the branch path of a DeepONet. The DeepSets layer confers linear complexity in the length of the prompt, in contrast to the quadratic complexity of self-attention, and a built-in permutation-invariance inductive bias to the order of the examples in the prompt.

Crucially, we prove that the DeepOSets architecture is universal for the in-context multi-operator learning problem. Specifically, we prove that it can approximate *uniformly* any compact set of continuous solution operators. This means that DeepOSets are not only universal approximators of single solution operators, like DeepONet, but they are universal approximators of multiple operators, in the sense that a single DeepOSets architecture exists that can approximate *any* operator in the class to the same fixed degree of accuracy. This guarantees that DeepOSets can learn in-context the solution operators of multiple differential equations.

Experiments with the Burgers' equation, 1D and 2D Darcy flow, and a variety of initial-value and boundary-value differential equations demonstrate that DeepOSets can accurately predict the solution to a new differential equation not seen during training. Furthermore, we conducted a direct controlled comparison with ICON on the 1D Darcy flow problem considered previously, which showed that under a small training budget DeepOSets outperformed ICON at every context size while using 19% of the parameters, while at large number of training steps, ICON is more accurate, with a relative error reduction from 26% to 41%, but at 5.2 times more parameters and roughly 218 times more wall-clock training time than DeepOSets (8.74 hours versus 11.6 seconds).

1.1 Our Contributions

We precisely formulate the multi-operator learning problem, propose the DeepOSets architecture for solving this problem, and prove its universality. We also provide experiments demonstrating the efficacy of our approach.

1. We provide a *precise mathematical formulation* of the in-context multi-operator learning problem and define universal approximation for this problem within Section 3 and Definition 3.3 respectively.
2. We prove that the DeepOSets architecture defined in Section 4 is *universal* for in-context multi-operator learning within Section 5 in Theorem 5.1. This is to the best of our knowledge the first universal approximation result for in-context multi-operator learning.
3. We demonstrate experimentally that DeepOSets is capable of learning compact classes of operators, including the operators corresponding to Burgers' equation and Darcy flow with variable diffusion, in the Experimental Results (Section 6).

2 Related Work

The universal approximation theorem for multi-layer perceptrons, which forms the basis for more sophisticated universal approximation theorems, was proved in [9, 12]. Universal approximation of operators using neural networks was first proved in [6], following previous results from [4, 5]. This result was used to design the DeepONet operator learning architecture [16]. Subsequently, various universal architectures for operator learning have been proposed, a notable example being Fourier Neural Operators [15]. These architectures typically evaluate the input function on a *fixed* grid, and an architecture compatible with variable input grids was proposed and analyzed in [18].

MIONet is to the best of our knowledge the first approach to attempt multi-input operator learning, by encoding several input functions via multiple branch network (and a shared trunk network) of a DeepONet, though it still learns a single operator [13]. POSEIDON is a large foundation model for PDEs based on a scalable Operator Transformer (scOT) with a vision transformer architecture, is pretrained on compressible Euler and incompressible Navier-Stokes equations and applied to 15 downstream tasks, showing better sample efficiency and accuracy than baselines [11]. PROSE-PDE is a transformer-based multi-operator learning approach trained on one-dimensional time-dependent nonlinear PDEs that encodes both numerical and symbolic data types for forward and inverse PDE problems [20]. LeMON is based on pretraining and fine-tuning for multi-operator learning, and introduces low-rank adaptation to reduce computational cost [21]. MT-DeepONet introduces modifications to the branch network of a DeepONet to accommodate various functional forms of parameterized coefficients and multiple geometries [14].

In-context learning (ICL) is a phenomenon first observed in large language models [3, 22], whereby these models are able to learn new tasks without fine-tuning from examples presented to them in the prompt. This has sparked substantial theoretical research effort into the mechanism underlying this phenomenon. Transformers have been theoretically shown to exhibit ICL on a variety of linear models, including least squares, ridge regression, Lasso, generalized linear models and linear inverse problems [10, 1, 32, 24]. Generalization error for transformer-based ICL for data lying on a low-dimensional manifold has been studied in [19]. In-context learning for differential equation solution operators was introduced by ICON [25], with a follow-up study on 1D scalar conservation laws providing a detailed methodology for applying ICON to time-evolution PDEs [28]. ICON-LM integrates ICON concepts with language-model architectures [27]. GenICON introduces a generative, probabilistic approach to ICON [31]. In-context learning of linear PDEs using transformers has also been studied in [8].

3 Mathematical Formulation of In-context Multi-Operator Learning

In-context operator learning involves a neural network learning a class of operators, and identifying a specific operator from data, called in-context examples. In this section, we give a precise mathematical formulation of this inverse problem. Based upon this, we will be able to give a precise definition of universality for this problem.

Let $K_1, K_2 \subset \mathbb{R}^d$ be compact subsets of some Euclidean space. We will consider continuous maps between the Banach spaces $X := C(K_1)$ and $Y := C(K_2)$, whose domain is a compact subset $V \subset X$. Note that the dimensionality of K_1 and K_2 need not be the same in general, but to simplify the presentation we restrict to this case in the following. The minor adjustments required to handle the more general case where $K_1 \subset \mathbb{R}^{d_1}$ and $K_2 \subset \mathbb{R}^{d_2}$ are left to the reader.

In traditional operator learning, a continuous function $G : V \to Y$ is approximated using a neural network. The input function $u \in V$ is discretized at a fixed set of points $x_1, ..., x_k \in K_1$, and the values $u(x_1), ..., u(x_k)$ together with an evaluation point $y \in K_2$ are passed to a neural network whose output approximates $G(u)(y)$. Universality in this context means that for any G and $\varepsilon > 0$, there exists a grid $\mathbf{x} := \{x_1, ..., x_k\} \in K_1$ (depending on G and ε) and a neural network G_ε such that

$$|G_\varepsilon(u(\mathbf{x}), y) - G(u)(y)| < \varepsilon$$

for any $u \in V$ and $y \in K_2$ [6, 16]. Here we denote by

$$u(\mathbf{x}) := (u(x_i))_{i=1}^k \in \mathbb{R}^k$$

the function evaluated at the grid $\mathbf{x}$.

Instead of learning a single operator $G : V \to Y$, in-context multi-operator learning considers an entire class of operators. To formulate this, we consider the space of continuous operators:

$$\mathcal{O} := \{G : V \to Y \text{ continuous}\}$$

equipped with the induced operator norm given by

$$\|G\|_{\mathcal{O}} := \sup_{x \in V} \|G(x)\|_Y.$$

We let $\mathcal{K} \subset \mathcal{O}$ be a compact subset of this space of operators. $\mathcal{K}$ is the class of operators that we wish to learn. In order to disambiguate the desired operator $G \in \mathcal{K}$, we pass the network a sequence of *in-context examples* $(u_1, G(u_1)), ..., (u_m, G(u_m))$ in addition to the input function $u \in K$ and the evaluation point $y \in K_2$. All of the functions u_i, u, and $G(u_i)$ are discretized on fixed grids $\mathbf{x} = (x_1, ..., x_k)$ and $\mathbf{y} = (y_1, ..., y_n)$ in K_1 and K_2, respectively. We remark that one could additionally consider variable grids in K_1 and K_2, but we leave this to future work.

3.1 Examples of in-context operator learning

Let us begin by giving some examples of compact sets of operators and instances of in-context operator learning.

Finite sets of operators. Any finite set of operators $\mathcal{K} = \{G_1, ..., G_p\}$ is a compact set. In this case, in-context operator learning would identify and evaluate the correct operator $G = G_q$ for some q from the in-context examples. This is rather trivial example of in-context multi-operator learning.

Continuously parameterized sets of operators. If the set of operators is continuously parameterized by a compact set, i.e., $\mathcal{K} = \{G_\mu : \mu \in K'\}$ where K' is a compact set (typically in $\mathbb{R}^d$ or in a Banach space), then $\mathcal{K}$ is compact. This is the typical application that we have in mind. As an example, consider the Darcy flow with variable permeability:

$$-\nabla \cdot (k(x)\nabla u(x)) = f(x) \text{ in } \Omega, \;\; u(x) = 0 \text{ on } \partial\Omega. \tag{1}$$

The solution operator mapping $f \to u$ is parameterized by the permeability k, and it is known that this parameterization is continuous with respect to the L_∞-norm [2]. Hence, if the permeability coefficients lie in compact subset of L_∞, then the corresponding class of solution operators is compact. This can be applied as follows. Suppose that a set of forcing terms f and their corresponding solutions u are observed for an unknown permeability k. Using the observed data as in-context data, the network would be able to predict the solution for a new forcing term f. We demonstrate this application in Section 6.2.

Finite unions of compact sets of operators. Given any finite collection of sets of compact operators, their union is again compact and in-context multi-operator learning can be applied. For example, instead of considering only a single parameterized PDE as in the previous paragraph, multiple parameterized PDEs can be learned at the same time. We demonstrate this application in Section 6.4.

3.2 Universality for in-context operator learning

We now formulate what it means for a neural network architecture to be universal for in-context operator learning. Roughly speaking, this means that the network can learn any compact set of operators to any desired accuracy provided enough in-context examples.

Recall that, although the grids $\mathbf{x}$ and $\mathbf{y}$ are fixed, we want to allow the in-context examples to be given for arbitrary functions $u_1, ..., u_m \in V$ (even the number of functions m should also be arbitrary). Of course, we need to make some assumption which guarantees that the u_i sufficiently cover the domain V so that the desired operator $G \in \mathcal{K}$ can be determined to the desired accuracy from the outputs $G(u_i)$. For technical reasons, we also need to choose the u_i to be sufficiently uniformly distributed within V. This is captured in the following definition.

Definition 3.1 ((δ, C)-discretization)**.** Let $\delta > 0$ and $C > 1$. A (δ, C)-discretization of a compact metric space V is a set of points $\mathbf{u} = \{u_1, \cdots, u_m\}$ such that

$$|\{u_i \in B(u, \delta/2)\}| < C|\{u_i \in B(v, \delta)\}|$$

for any $u, v \in V$.

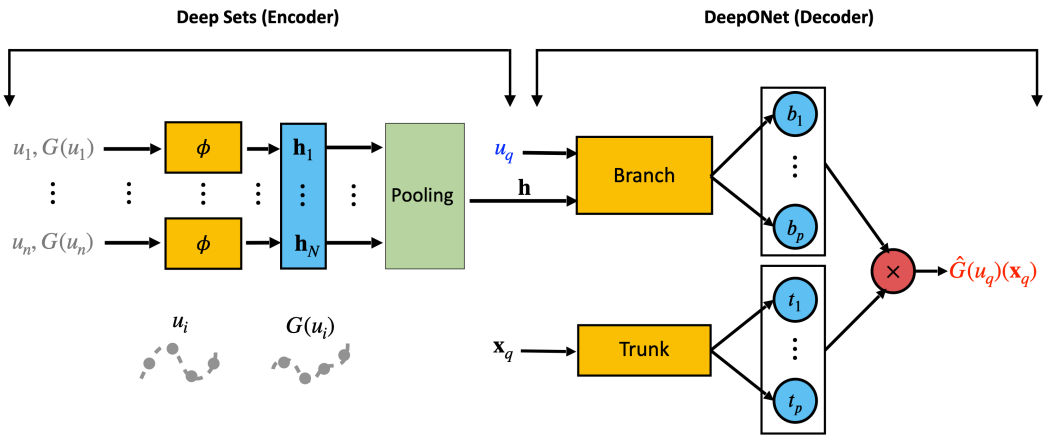


Figure 1: DeepOSets architecture for in-context multi-operator learning.

Intuitively, a (δ, C)-discretization is a δ-net of V which is not too unbalanced, in the sense that the density of points is relatively uniform. Let us first note that this definition is not vacuous.

Lemma 3.2. *Let V be a compact set in a metric space, $\delta > 0$ and $C > 1$. Then there always exists a (δ, C)-discretization of V.*

We can now formulate the concept of universality for in-context multi-operator learning.

Definition 3.3 (Mathematical formulation of In-Context Operator Learning)**.** We define universality for in-context operator learning to mean that for any $\mathcal{K} \subset \mathcal{O}$, $\epsilon > 0$ and $C > 1$, there exist grids $\mathbf{x} = (x_1, ..., x_k)$ and $\mathbf{y} = (y_1, ..., y_n)$ in K_1 and K_2, respectively, a $\delta > 0$ and a network K_ϵ such that

$$|K_\varepsilon(u_1(\mathbf{x}), G(u_1)(\mathbf{y}), ..., u_m(\mathbf{x}), G(u_m)(\mathbf{y}), u(\mathbf{x}), y) - G(u)(y)| < \varepsilon\,, \tag{2}$$

for any $G \in \mathcal{K}$, $u \in V$, $y \in K_2$, and (δ, C)-discretization $\mathbf{u} = (u_1, ..., u_m)$ of V. In other words, any operator of $\mathcal{K}$ can be approximated to arbitrary accuracy provided that the in-context examples are sufficiently dense in V.

Note that the network architecture should be able to process an arbitrary, i.e., not a priori specified, number of in-context examples. This can be achieved using a transformer architecture [23], for example. In the next section, we describe a simple architecture based upon Deep Sets [29], called DeepOSets [7] for this purpose.

4 The DeepOSets Architecture for In-Context Operator Learning

The DeepOSets model is a hybrid neural network comprising of a DeepSets encoder and a DeepONet decoder, as seen in Figure 1.

The DeepSets module encodes the discretized parameter and solution function example pairs in the prompt. Each pair $(u_i, G(u_i))$ is first mapped to an individual fixed-length embedding $\mathbf{h_i}$ by a shared multi-layer perceptron (MLP) denoted Φ. These intermediate representations are then combined by a pooling layer into the embedding vector $\mathbf{h}$. This introduces a permutation invariance inductive bias to the prompt examples, as long as the pooling layer is a permutation invariant operation, such as min, max, mean or set attention pooling. The permutation invariance is a key inductive bias, since the order of the in-context examples does not matter.

The DeepONet takes the concatenation of the embedding $\mathbf{h}$ and the discretized query parameter function u_q at its branch path and the query coordinates x_q at its trunk path, and outputs the predicted solution function $\hat{G}(u_q)$ evaluated at x_q. The output is therefore entirely mesh-free. We denote the branch network as ρ, and the trunk network as Ψ. With this notation, the output of the DeepOSets architecture using mean pooling, given input x_q, can be mathematically expressed as,

$$\hat{G}(u_q)(x_q) := \Psi(x_q) \cdot \rho\left(u_q(\mathbf{x}), \frac{1}{m}\sum_{i=1}^{m} \Phi(u_i(\mathbf{x}), G(u_i)(\mathbf{y}))\right). \tag{3}$$

5 Universality of DeepOSets for In-Context Operator Learning

In this section, we prove that DeepOSets is a universal *uniform* approximator for any compact set of continuous operators in the sense of Definition 3.3, i.e., a single DeepOSets neural network exists that bounds the supremum of the error over the entire class to any desired fixed accuracy. This shows that a unique DeepOSets neural network could be learned that approximates in-context any continuous operator to the desired accuracy, given an appropriate number of examples in the prompt.

Theorem 5.1 (Uniform universality for In-Context Operator Learning)**.** *Given $\varepsilon > 0$, $C > 1$ and a compact set $\mathcal{K} \subset \mathcal{O}$, there exist integers $N, M \geq 1$, $\delta > 0$, fixed discretizations $\boldsymbol{x} = \{x_1, ..., x_k\}$ of K_1 and $\boldsymbol{y} = \{y_1, ..., y_n\}$ of K_2, and continuous functions $\Psi : \mathbb{R}^d \to \mathbb{R}^M$, $\rho : \mathbb{R}^{k+N} \to \mathbb{R}^M$, and $\Phi : \mathbb{R}^{k+n} \to \mathbb{R}^N$, such that*

$$\sup_{G \in \mathcal{K}} \left| G(u_q)(x_q) \; - \; \Psi(x_q) \cdot \rho \left(u_q(\boldsymbol{x}), \frac{1}{m} \sum_{i=1}^{m} \Phi(u_i(\boldsymbol{x}), G(u_i)(\boldsymbol{y})) \right) \right| < \varepsilon. \tag{4}$$

for any (δ, C)-discretization $\boldsymbol{u} := (u_1, \cdots, u_m)$ of V, any $u_q \in V$, and any $x_q \in K_2$.

Remark 5.2. Note that constructing a (δ, C)-discretization may be difficult in practice. In addition, a (δ, C)-discretization may need to be very large, which would require a large number of in-context examples. However, the universality result is an existence result, and in practice a much smaller number of in-context examples may be sufficient to obtain good accuracy.

Proof. For the full detailed proof, please see the Appendix.

Here, we present an overview of the proof, in which the approximation error is decomposed into two inequalities. The compactness assumption enables this decomposition to hold uniformly over the class of continuous operators by reducing the problem to finite-dimensional approximations via discretizations on the grids $\mathbf{x}$ and $\mathbf{y}$.

For the first inequality, using a partition of unity Lemma (in Appendix), we show that there exists a grid $\mathbf{y}$ of K_2, and $\{\gamma_i\}_{i=1}^n \subset C(K_2)$, such that

$$\|G(u_q) - \sum_{i=1}^{n} G(u_q)(y_i)\gamma_i\|_{C(K_2)} < \varepsilon/2,$$

for every $G \in \mathcal{K}$ and $u_q \in V$.

For the second inequality, we show that there exists a grid $\mathbf{x}$ of K_1 such that $G(u_q)(\mathbf{y}) := (G(u_q)(y_1), \cdots, G(u_q)(y_n))$ can be approximated well by functions of the form,

$$\rho \left(u_q(\mathbf{x}), \frac{1}{m} \sum_{i=1}^{m} \Phi(u_i(\mathbf{x}), G(u_i)(\mathbf{y})) \right), \tag{5}$$

where $u_1, ..., u_m$ is a (δ, C)-discretization of V.

We construct Φ using a partition-of-unity argument to be localized/subordinate over a discretization of the input space, and define ρ by forming normalized local averages of the corresponding operator output samples evaluated on the output grid and recombining them into a continuous finite-dimensional approximation.

We then let $\Psi : \mathbb{R}^d \to \mathbb{R}^n$, (here $n = M$), be defined as,

$$\Psi(x) = \{\gamma_i(x)\}_{i=1}^n.$$

Taking the dot product with (5) and applying the triangle inequality together with the two bounds yields the desired approximation inequality.

Finally, we verify that the constructions Φ and ρ are continuous. Continuity of Φ is immediate from its definition, while continuity of ρ follows from uniform lower bounds on the normalization factors ensured by the balanced (δ, C)-discretization of V. □

The continuous functions guaranteed by Theorem 5.1 can be approximated using neural networks by invoking standard universal approximation theorems for MLPs [9, 12, 17]. Here it is important that we are averaging the outputs of Φ to ensure that the function ρ only needs to be approximated on a compact set.

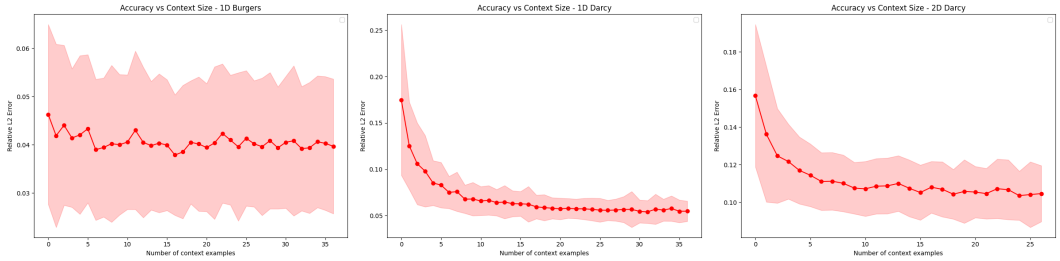


Figure 2: Relative L_2 error vs in-context examples for 1D Burgers', 1D Darcy Flow, and 2D Darcy Flow, with red shading representing one standard deviation, all experiments show a decrease in relative error as context increases, however it is much more pronounced in the Darcy examples, perhaps due to the more complicated operator paramatrization.

6 Experimental Results

In this section, we demonstrate the in-context operator learning paradigm on the one-dimensional Burgers' equation, one and two dimensional Darcy flow, and a collection of different forward and inverse differential equation operators. We do not compare our method against single-operator learning methods, since our multi-operator DeepOSets has the fundamentally different goal of learning a compact class of operators with no knowledge of the underlying PDE parameters, which are never passed to the network. We also provide a controlled comparison with the ICON model [26, 28]. In contrast to ICON, we remark that our method is capable of directly handling two dimensional parametric PDEs, such as the two-dimensional Darcy flow in Section 6.3. The data generation process for all experiments is described in Appendix B. Source code is available at `https://github.com/stevengogogo/in-context-multi-operator-learning`

6.1 Burgers' equation

First, we study Burgers' equations,

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu \frac{\partial^2 u}{\partial x^2}, \tag{6}$$

$$u(x, 0) = f(x) \tag{7}$$

on $(x, t) \in [0, 1]^2$, with $f(0) = f(1) = 0$.

For a given ν, let G_ν represent the solution operator for 6, which given an initial condition $u(x, 0)$, returns the final time solution $u(x, 1)$. The class of operators we seek to approximate can then be described as,

$$\{G_\nu : \nu \in [\alpha, \beta]\} \tag{8}$$

for constants α, β. For our experiments, we let $\nu \in [0.001, 0.1]$, and for data generation used log-uniform sampling to select ν.

Log-uniform sampling is important to mention, since for Burgers' equation, as ν tends towards 0, the solution will develop shocks, but for large ν the solution will be smooth, and so we need our generated training data to include adequate examples of both. For each sampled ν, we generate 40 initial conditions u_0, and corresponding FDM solutions u.

During training, the network is given 39 in-context examples, and predicts the 40th one. At testing time we vary the number of in-context examples given from 3 to 39, and test 100 different operators, randomly selected from our test dataset, to calculate the relative L_2 error.

As illustrated in 2, the relative error even with just 3 in-context examples is 0.0453, and this reduces to 0.0391 for 39 in-context examples. The overall small reduction with increasing in-context size, and relatively high variance, suggests that the DeepOSets architecture is able to identify the underlying operator with relatively few in-context examples. This intuitively makes sense for the class of Burgers' operators 8, since it is parametrized by a varying constant ν instead of a more complex function. Experimentally, our architecture is able to capture both smooth solutions, and developing shock solutions of Burgers', with only a few in-context examples as illustrated in 4.

6.2 1D Darcy flow

We study the 1D Darcy flow,

$$-\frac{d}{dx}(k(x)\frac{d}{dx}u(x)) = f(x), x \in (0,1) \tag{9}$$

$$u(0) = u(1) = 0 \tag{10}$$

Here, given some k, our solution operator can be described as G_k. We generate multiple smooth $k's$, and for each one generate 40 functions f, and the corresponding FDM solution for u, which will then serve as our data. The training and testing were conducted in the same manner as Burgers', in terms of in context examples provided during training/testing, and how average L_2 error was calculated. For 1D Darcy flow, we see a much more drastic decrease in average relative L_2 error on the test dataset, 2, with smaller variance, as the in-context examples increase, from 0.0866 to 0.0449. Since in this case, the class of operators is parametrized by a function $k(x)$, as opposed to a constant like in Burgers', the network greatly benefits from added in-context, as seen in 5, and with enough context can perform well experimentally. Additional results can be found in Appendix E.

6.3 2D Darcy Flow

We now study the 2D Darcy flow,

$$-\text{div}(k(x) \cdot \nabla u(x)) = f(x),\ x \in (0,1)^2 := \Omega \tag{11}$$

$$u(x) = 0, x\ \in \partial\Omega \tag{12}$$

The class of operators can be described in the exact same manner as in 1D, and the data generation, training, and testing, all follow similar protocols as in 1D Darcy flow and Burgers' equation. The relative L2 vs. in-context examples can be seen in 2. Again, similarly to 1D Darcy, we see a noticeable reduction in relative L_2 error as the number of in-context examples increases, however this time it reduces from 0.1289 to 0.1163. Given the fact that the $k(x)$ functions can be much more complicated in the 2D case as compared to the 1D case, this is expected. Representative example solutions can be found in Appendix E. However, the reduction in error does demonstrate the network is understanding how to disambiguate the operator.

6.4 Multiple differential equation operators

A union of compact sets is also compact, and so it stands to reason that DeepOSets could feasibly learn multiple different operators for multiple PDEs with unknown parameters. In this section, we demonstrate that the DeepOSets architecture successfully learns the solution operator for various initial-value and boundary-value problems. Our evaluation spans 12 distinct tasks, comprising forward and inverse versions of each problem. The problems are parameterized differential equations, and our compact set in this case consists of the union of the operators obtained for various values of the parameters. See Table 1 for a summary of all the problems and the results. All of the implementation details and details on the training of the network can be found in Appendix B.

6.5 Extrapolation to Prompt Sizes not Seen During Training.

A distinct advantage of the DeepOSets framework is its permutation invariance and flexibility regarding the number of examples in the prompt. To evaluate this extrapolation capability, we used different number of in-context examples ranging from $N = 3$ to $N = 20$ at inference time. As illustrated in Figure 3, the model successfully generalizes to context sizes significantly larger than the training distribution ($N_{\text{test}} > N_{\text{train}}$). Specifically, for the Poisson, "porous" (linear reaction diffusion) and "cubic" (nonlinear reaction diffusion) forward operators, we observe a consistent monotonic decrease in Mean Squared Error (MSE) as the number of context examples increases. This behavior confirms that the model is not overfitting to the specific training sequence length; rather, it effectively performs in-context learning, leveraging additional data points to refine the operator approximation. For the inverse problems, the model maintains performance stability across varying context sizes without divergence, demonstrating robust generalization.

Problem	Equations	Parameters	Test MSE
Forward ODEConst	$\frac{du(t)}{dt} = a_1 c(t) + a_2$	a_1, a_2	2.65e-03 ± 9.30e-03
Inverse ODEConst			1.15e-02 ± 2.71e-02
Forward ODE1	$\frac{du(t)}{dt} = a_1 u(t) c(t) + a_2$	a_1, a_2	1.65e-03 ± 8.39e-03
Inverse ODE1			1.44e-02 ± 3.86e-02
Forward ODE2	$\frac{du(t)}{dt} = a_1 u(t) + a_2 c(t) + a_3$	a_1, a_2, a_3	1.19e-02 ± 6.99e-02
Inverse ODE2			1.40e-02 ± 9.15e-02
Forward Poisson	$\frac{d^2 u(x)}{dx^2} = f(x)$	None	3.04e-03 ± 1.91e-02
Inverse Poisson			4.60e-03 ± 1.35e-02
Forward Linear React-Diff	$-a\frac{d^2 u(x)}{dx^2} + k(x)u(x) = c$	$u(0), u(1), a, c$	2.11e-03 ± 4.25e-03
Inverse Linear React-Diff			1.45e-02 ± 6.01e-02
Forward Nonlinear React-Diff	$-a\frac{d^2 u(x)}{dx^2} + ku(x)^3 = c(x)$	$u(0), u(1), a, c$	2.77e-03 ± 5.69e-03
Inverse Nonlinear React-Diff			1.19e-02 ± 3.51e-02

Table 1: Summary of tasks. The reported Mean Squared Error (MSE) corresponds to the model trained on all 12 problems simultaneously. All models were tested on individual tasks.

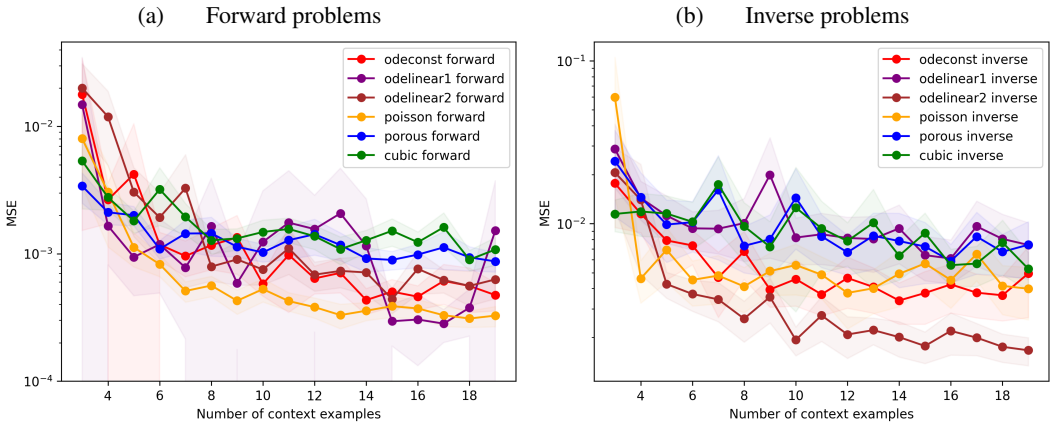


Figure 3: Zero-shot generalization to larger context sets. The model was trained with a fixed context size of $N = 4$. During inference, we evaluate performance on context sizes ranging from 3 to 20. (a) Forward operators show a clear trend where providing more examples monotonically decreases the MSE, indicating the model successfully leverages additional data. (b) Inverse operators maintain stability across varying context sizes. Shaded regions denote one standard deviation.

6.6 Comparison with ICON

To compare DeepOSets against an existing in-context multi-operator learning method, we carried out a controlled benchmark of DeepOSets and ICON [26] on a 1D Darcy flow (variable-coefficient Poisson) boundary-value problem, $-\frac{d}{dx}\left(k(x)\frac{du}{dx}\right) = f(x)$ on $(0, 1)$ with $u(0) = u(1) = 0$, where each operator corresponds to an independently sampled diffusivity field $k(x)$. The dataset consists of 5,000 operators with 40 (f, u) pairs each, discretized on 50 interior grid points, with a 90%/10% operator-level train/test split. Full details of the dataset, architectures, and training protocol are given in Appendix B.5.

The results are summarized in Tables 2 and 3. Under a small training budget of 10,000 steps, DeepOSets outperforms ICON at every context size while using only 19% of ICON's parameters; ICON requires substantially more training steps to converge, given its much larger sequence length and parameter count. When the training schedule is extended to 100,000 steps, ICON attains a lower mean relative L_2 error than DeepOSets at every tested context size, with a relative reduction in error ranging from approximately 26% to 41%. This accuracy comes at a cost: under the equal-step schedule, ICON required approximately 218 times more wall-clock training time than DeepOSets (8.74 hours versus 144.6 seconds), roughly 12 times higher per-sample inference latency, and about 5.3 times

Metric	DeepOSets	ICON
Trainable parameters	894,465	4,744,193
Training time, 10k steps	42.5 s	2.15 h
Training time, 100k steps	144.6 s	8.74 h
Mean time per training step	1.446 ms	314.78 ms
Training throughput	691.7 steps/s	3.18 steps/s
Inference time per sample	0.229 ms	2.750 ms

Table 2: Computational cost of DeepOSets and ICON on the 1D Darcy flow benchmark. Inference time is the mean GPU wall-clock time per forward pass over 500 timed calls (100 per context size) for the models trained for 100k steps.

	10k training steps		**100k training steps**	
n_{ctx}	DeepOSets	ICON	DeepOSets	ICON
1	**0.2831**	0.7913	0.2096	**0.1342**
2	**0.2085**	0.8056	0.1180	**0.0826**
3	**0.1881**	0.6299	0.0976	**0.0589**
4	**0.1740**	0.6843	0.0821	**0.0572**
5	**0.1459**	0.7058	0.0819	**0.0603**

Table 3: Mean relative L_2 test error of DeepOSets and ICON on the 1D Darcy flow benchmark as a function of the number of in-context examples n_{ctx}, averaged over 100 held-out test samples per condition, under equal-step training schedules of 10k and 100k steps.

more parameters. The training-time gap is consistent with the cost of self-attention over a token sequence of length 850 at each step, in contrast to the DeepOSets encoder, whose cost is linear in the number of context pairs and which processes a short fixed-size input. In summary, the comparison reveals a clear trade-off between predictive accuracy and computational cost: ICON achieves better accuracy with extended training, while DeepOSets is substantially more computationally efficient and attains comparable accuracy at a small fraction of the cost.

7 Discussion and Limitations

A limitation of our work is that the universality result we have proved is not quantitative. In particular, we do not know how large the network must be, nor how many in-context examples are required, to learn a given compact class of operators. For operators arising from PDEs, this as an important direction for future research. Our experimental results demonstrate the viability of DeepOSets on one and two-dimensional cases with both finite and infinite-dimensional unknown parameters. We also show that DeepOSets is competitive with ICON, a much more complex transformer-based architecture. While ICON is more accurate at 100K training steps, the difference with respect to DeepOSets is small at 5 in-context examples, while DeepOSets is much faster to train.

8 Conclusion

Multi-operator learning involves training a neural network to learn a class of operators simultaneously. In-context multi-operator learning follows an amortization approach, where a neural network is trained on a large number of examples sampled from the universe of operators and, at inference time, a prediction is produced quickly by disambiguating the correct operator using a small number of examples provided in a prompt, without any neural network weight updates. We have studied in-context multi-operator learning through the DeepOSets architecture, which we prove to be a universal approximator over a compact class of operators and demonstrate to be viable in practice through a comprehensive set of experiments with operators arising from PDEs. Pushing the boundaries of what is possible using this approach is an important, and promising, area of research.

References

[1] Yu Bai, Fan Chen, Huan Wang, Caiming Xiong, and Song Mei. Transformers as statisticians: Provable in-context learning with in-context algorithm selection. *Advances in neural information processing systems*, 36:57125–57211, 2023.

[2] Andrea Bonito, Ronald A DeVore, and Ricardo H Nochetto. Adaptive finite element methods for elliptic problems with discontinuous coefficients. *SIAM Journal on Numerical Analysis*, 51(6):3106–3134, 2013.

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

[4] Tianping Chen and Hong Chen. Approximations of continuous functionals by neural networks with application to dynamic systems. *IEEE Transactions on Neural networks*, 4(6):910–918, 1993.

[5] Tianping Chen and Hong Chen. Approximation capability to functions of several variables, nonlinear functionals, and operators by radial basis function neural networks. *IEEE Transactions on Neural Networks*, 6(4):904–910, 1995.

[6] Tianping Chen and Hong Chen. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE transactions on neural networks*, 6(4):911–917, 1995.

[7] Shao-Ting Chiu, Junyuan Hong, and Ulisses Braga-Neto. Deeposets: Non-autoregressive in-context learning of supervised learning operators. *arXiv preprint arXiv:2410.09298*, 2024.

[8] Frank Cole, Yulong Lu, Riley ONeill, and Tianhao Zhang. Provable in-context learning of linear systems and linear elliptic pdes with transformers. *arXiv preprint arXiv:2409.12293*, 2024.

[9] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[10] Shivam Garg, Dimitris Tsipras, Percy S Liang, and Gregory Valiant. What can transformers learn in-context? a case study of simple function classes. *Advances in neural information processing systems*, 35:30583–30598, 2022.

[11] Maximilian Herde, Bogdan Raoni, Tobias Rohner, Roger Käppeli, Roberto Molinaro, Emmanuel de Bézenac, and Siddhartha Mishra. Poseidon: Efficient foundation models for pdes. 2024. arXiv preprint arXiv:2405.19101.

[12] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[13] Pengzhan Jin, Shuai Meng, and Lu Lu. Mionet: Learning multiple-input operators via tensor product. *SIAM Journal on Scientific Computing*, 44(6):A3490–A3514, 2022.

[14] Varun Kumar, Somdatta Goswami, Katiana Kontolati, Michael D Shields, and George Em Karniadakis. Synergistic learning with multi-task deeponet for efficient pde problem solving. *Neural Networks*, 184:107113, 2025.

[15] 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.

[16] Lu Lu, Pengzhan Jin, and George Em Karniadakis. DeepONet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, March 2021. arXiv:1910.03193 [cs, stat].

[17] Allan Pinkus. Approximation theory of the mlp model in neural networks. *Acta numerica*, 8:143–195, 1999.

[18] Michael Prasthofer, Tim De Ryck, and Siddhartha Mishra. Variable-input deep operator networks. *arXiv preprint arXiv:2205.11404*, 2022.

[19] Zhaiming Shen, Alexander Hsu, Rongjie Lai, and Wenjing Liao. Understanding in-context learning on structured manifolds: Bridging attention to kernel methods. In *International Conference on Learning Representations*, volume 2026, pages 42067–42103, 2026.

[20] Jingmin Sun, Yuxuan Liu, Zecheng Zhang, and Hayden Schaeffer. Towards a foundation model for partial differential equations: Multioperator learning and extrapolation. *Physical Review E*, 111(3):035304, 2025.

[21] Jingmin Sun, Zecheng Zhang, and Hayden Schaeffer. Lemon: Learning to learn multi-operator networks. 2024. arXiv preprint arXiv:2408.16168.

[22] Maria Tsimpoukelli, Jacob L Menick, Serkan Cabi, S. M. Ali Eslami, Oriol Vinyals, and Felix Hill. Multimodal few-shot learning with frozen language models. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 200–212. Curran Associates, Inc., 2021.

[23] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[24] Johannes Von Oswald, Eyvind Niklasson, Ettore Randazzo, João Sacramento, Alexander Mordvintsev, Andrey Zhmoginov, and Max Vladymyrov. Transformers learn in-context by gradient descent. In *International Conference on Machine Learning*, pages 35151–35174. PMLR, 2023.

[25] Liu Yang, Siting Liu, Tingwei Meng, and Stanley J. Osher. In-context operator learning with data prompts for differential equation problems. *Proceedings of the National Academy of Sciences*, 120(39):e2310142120, 2023.

[26] Liu Yang, Siting Liu, Tingwei Meng, and Stanley J. Osher. In-context operator learning with data prompts for differential equation problems. *Proceedings of the National Academy of Sciences*, 120(39):e2310142120, 2023.

[27] Liu Yang, Siting Liu, and Stanley J. Osher. Fine-tune language models as multi-modal differential equation solvers. *arXiv preprint arXiv:2308.05061*, 2023.

[28] Liu Yang and Stanley J. Osher. PDE generalization of in-context operator networks: A study on 1d scalar nonlinear conservation laws. *Journal of Computational Physics*, 519:113379, 2024.

[29] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017.

[30] Benjamin J. Zhang, Siting Liu, Stanley J. Osher, and Markos A. Katsoulakis. Probabilistic operator learning: Generative modeling and uncertainty quantification for foundation models of differential equations. 2025. arXiv preprint arXiv:2509.05186.

[31] Benjamin J. Zhang, Siting Liu, Stanley J. Osher, and Markos A. Katsoulakis. Probabilistic operator learning: generative modeling and uncertainty quantification for foundation models of differential equations. *arXiv preprint arXiv:2509.05186*, 2025.

[32] Ruiqi Zhang, Spencer Frei, and Peter L Bartlett. Trained transformers learn linear models in-context. *Journal of Machine Learning Research*, 25(49):1–55, 2024.

A Proofs of Theorem 5.1 and Lemma 3.2

Let us begin by proving Lemma 3.2 showing that (δ, C)-discretizations always exist for any compact metric space V.

Proof of Lemma 3.2. Let $V \subset X$ be compact, and let $\delta > 0$ be given. Choose a maximal δ-separated subset of V, i.e., a subset $\mathcal{X}_\delta$ such that $d(u_i, u_j) \geq \delta$ for all $i \neq j$, and which is maximal with respect to this property. It is clear that the balls of radius δ around the points in $\mathcal{X}_\delta$ give a covering of V (since any point not covered by such a ball could be added to $\mathcal{X}_\delta$, contradicting the maximality), and so by compactness $\mathcal{X}_\delta$ must be a finite set. We claim that $\mathcal{X}_\delta$ is a (δ, C)-covering for any $C > 1$.

First, note that any ball of radius $\delta/2$ contains at most one point of $\mathcal{X}_\delta$. Indeed, fix $z \in V$ and suppose $u_i, u_j \in \mathcal{X}_\delta$ both lie in $B(z, \delta/2)$ with $i \neq j$. Then, by the triangle inequality,

$$d(u_i, u_j) \;\leq\; d(u_i, z) + d(z, u_j) \;<\; \frac{\delta}{2} + \frac{\delta}{2} \;=\; \delta,$$

which contradicts the δ–separation of $\mathcal{X}_\delta$. Hence, for every $z \in V$,

$$\big|\{u_i \in \mathcal{X}_\delta : u_i \in B(z, \delta/2)\}\big| \leq 1.$$

Next, since balls of radius δ around the points in $\mathcal{X}_\delta$ cover V, for every $v \in V$ there exists some $u_j \in \mathcal{X}_\delta$ with $d(v, u_j) < \delta$, i.e. $u_j \in B(v, \delta)$. Thus, for all $v \in V$,

$$\big|\{u_i \in \mathcal{X}_\delta : u_i \in B(v, \delta)\}\big| \geq 1.$$

Combining these two observations, we obtain, for all $u, v \in V$,

$$\big|\{u_i \in \mathcal{X}_\delta : u_i \in B(u, \delta/2)\}\big| \;<\; C\,\big|\{u_i \in \mathcal{X}_\delta : u_i \in B(v, \delta)\}\big|,$$

for any $C > 1$. Hence $\mathcal{X}_\delta$ is a (δ, C)–discretization of V. □

Next, we state a partition of unity approximation Lemma for a compact subset of $C(K)$, which we will use in the proof of Theorem 5.1.

Lemma A.1. *Let K be a compact metric space and $V \subset C(K)$ be compact. Given $\varepsilon > 0$, there exist $\mathbf{y} := \{y_1, \cdots, y_n\} \in K$, and $\varphi_1, \cdots, \varphi_n \in C(K)$ with $\|\varphi_i\| \leq 1$, such that*

$$\|f - \sum_{i=1}^{n} f(y_i)\varphi_i\|_{C(K)} < \varepsilon$$

for all $f \in V$. In particular, for any $f, g \in V$ we have

$$\|f - g\|_{C(K)} \leq \|f - g\|_{\ell^\infty(\mathbf{y})} + 2\varepsilon. \tag{13}$$

Proof. (of Lemma A.1) Since V is compact, by the Arzela-Ascoli theorem it is also equicontinuous, and so there exists $\delta > 0$ such that if $|x - y| < \delta$, then $|f(x) - f(y)| < \varepsilon$ for all $f \in V$. Since K is also compact, we can pick $y_1, \cdots, y_n \in K$ such that $\{B(y_i, \delta)\}_{i=1}^n$ form a covering of K. Finally, let $\{\varphi_i\}_{i=1}^n$ be a partition of unity subordinate to the balls $\{B(y_i, \delta)\}_{i=1}^n$.

Given $y \in K$, for any $f \in V$, we have that,

$$\begin{aligned}
|f(y) - \sum_{i=1}^{n} f(y_i)\varphi_i(y)| &= |\sum_{i=1}^{n}(f(y) - f(y_i))\varphi_i(y)| \\
&< |\sum_{i, |y - y_i| < \delta} (f(y) - f(y_i)) \cdot \varphi_i(y)| \\
&< |\sum_{i, |y - y_i| < \delta} \varepsilon \cdot \varphi_i(y))| \leq \varepsilon.
\end{aligned}$$

Using this, if $f, g \in V$ we have

$$\begin{aligned}
\|f - g\|_{C(K)} &\leq \|f - \sum_{i=1}^{n} f(y_i)\varphi_i\|_{C(K)} + \|f - g\|_{\ell^\infty(\mathbf{y})} \max_i \|\varphi_i\|_{C(K)} + \|g - \sum_{i=1}^{n} g(y_i)\varphi_i\|_{C(K)} \\
&\leq \|f - g\|_{\ell^\infty(\mathbf{y})} + 2\varepsilon.
\end{aligned} \tag{14}$$

□

We can now return to the proof of Thm 5.1.

Proof. (of Theorem 5.1) Fix $\varepsilon > 0$. Since $\mathcal{K}$ is compact and V is compact, it follows that the set

$$\{G(u) : \ G \in \mathcal{K}, \ u \in V\} \subset C(K_2) \tag{15}$$

is compact. Since K_2 is also compact, we can apply Lemma A.1 to get that there exists a grid $\mathbf{y}$ of K_2, and $\{\gamma_i\}_{i=1}^n \subset C(K_2)$, such that

$$\|G(u_q) - \sum_{i=1}^{n} G(u_q)(y_i)\gamma_i\|_{C(K_2)} < \varepsilon/2,$$

for every $G \in \mathcal{K}$ and $u_q \in V$. So let $\Psi : \mathbb{R}^d \to \mathbb{R}^n$ be defined as,

$$\Psi(y) = \{\gamma_i(y)\}_{i=1}^n.$$

Using the notation $G(u_q)(\mathbf{y}) := (G(u_q)(y_1), \cdots, G(u_q)(y_n))$, we have that

$$\|G(u_q) - G(u_q)(\mathbf{y}) \cdot \Psi\|_{C(K_2)} < \varepsilon/2. \tag{16}$$

for every $G \in \mathcal{K}$.

We now construct a grid $\mathbf{x} = (x_1, ..., x_k)$ in K_1 and continuous functions Φ and ρ, such that $G(u_q)(\mathbf{y})$ is approximated to accuracy $\varepsilon/2$ by

$$\rho\left(u_q(\mathbf{x}), \frac{1}{m}\sum_{i=1}^{m} \Phi(u_i(\mathbf{x}), G(u_i)(\mathbf{y}))\right), \tag{17}$$

for any (δ, C)-discretization $u_1, ..., u_m$ of V.

Since $\mathcal{K}$ is compact, it is equi-continuous, so there exists a δ such that if $\|u - v\|_{C(K_1)} < 37\delta/6$, then $\|G(u) - G(v)\|_{C(K_2)} < \varepsilon/(2 \cdot \|\Psi\|_{C(K_2)})$ for every $G \in \mathcal{K}$. By Lemma A.1, there exists $\mathbf{x} = \{x_1, \cdots, x_k\} \in K_1$ such that

$$\|u - v\|_{C(K_1)} \leq \|u - v\|_{\ell^\infty(\mathbf{x})} + \frac{\delta}{6}. \tag{18}$$

We let $\mathbf{x}$ be our grid in K_1 and δ be the value given in the Theorem.

Let $u_1, ..., u_m \in V$ be a (δ, C)-discretization of V, and consider the restrictions $u_i(\mathbf{x})$ of u_i to the grid $\mathbf{x}$, which lie in the compact set $V(\mathbf{x}) \subset \mathbb{R}^k$. Let $u \in K$ and observe that by (18) we have

$$\|u - u_i\|_{C(K_1)} - \frac{\delta}{6} \leq \|u - u_i\|_{\ell^\infty(\mathbf{x})} \leq \|u - u_i\|_{C(K_1)} \tag{19}$$

This implies that

$$\{i : u_i(\mathbf{x}) \in B(u(\mathbf{x}), \delta/3)\} \subset \{i : u_i \in B(u, \delta/2)\} \tag{20}$$

and that

$$\{i : u_i \in B(u, \delta)\} \subset \{i : u_i(\mathbf{x}) \in B(u(\mathbf{x}), \delta)\}. \tag{21}$$

Hence, since $u_1, .., u_m$ is a (δ, C)-discretization, when restricted to the grid $\mathbf{x}$ we still have

$$|\{u_i(\mathbf{x}) \in B(u(\mathbf{x}), \delta/3)\}| \leq C|\{u_i(\mathbf{x}) \in B(v(\mathbf{x}), \delta)\}| \tag{22}$$

for any $u, v \in V$.

Next, we choose a maximal 3δ-separated subset $\widetilde{u}_1(\mathbf{x}), ..., \widetilde{u}_l(\mathbf{x})$ of the set $V(\mathbf{x}) \subset \mathbb{R}^k$, and consider the open cover of $V(\mathbf{x})$ given by

$$U_j := B(\widetilde{u}_j(\mathbf{x}), 3\delta) \setminus \bigcup_{i=1}^{j} \overline{B(\widetilde{u}_i(\mathbf{x}), \delta)}. \tag{23}$$

These sets are open since we are removing a finite union of closed sets from the open ball $B(\widetilde{u}_j(\mathbf{x}), 3\delta)$. By maximality, the balls $B(\widetilde{u}_j(\mathbf{x}), 3\delta)$ cover $V(\mathbf{x})$, and so each point $v(\mathbf{x}) \in V(\mathbf{x})$ is contained in $B(\widetilde{u}_j(\mathbf{x}), 3\delta)$ for some j. If

$$v(\mathbf{x}) \notin \bigcup_{i=1}^{l} \overline{B(\widetilde{u}_i(\mathbf{x}), \delta)}, \tag{24}$$

then $v(\mathbf{x}) \in U_j$. Otherwise, $v(\mathbf{x}) \in \overline{B(\widetilde{u}_i(\mathbf{x}), \delta)}$ for some i, and hence for exactly one i since the balls $\overline{B(\widetilde{u}_i(\mathbf{x}), \delta)}$ are disjoint (because the $\widetilde{u}_i(\mathbf{x})$ are 3δ-separated). In this case, we have $v(\mathbf{x}) \in U_i$. Hence, the sets U_j cover $V(\mathbf{x})$.

Let $\varphi_j : \mathbb{R}^k \to \mathbb{R}_{\geq 0}$ be a partition of unity subordinate to the covering U_j. By construction, the balls of radius δ about $\widetilde{u}_j(\mathbf{x})$ are contained in U_j and disjoint from U_i for $i \neq j$, and so we have

$$\varphi_j(v(\mathbf{x})) = 1 \text{ whenever } \|v - \widetilde{u}_j\|_{\ell^\infty(\mathbf{x})} \leq \delta. \tag{25}$$

In addition, by construction we have

$$\varphi_j(v(\mathbf{x})) = 0 \text{ whenever } \|v - \widetilde{u}_j\|_{\ell^\infty(\mathbf{x})} \geq 3\delta. \tag{26}$$

Let $\Phi : \mathbb{R}^{k+n} \to \mathbb{R}^{l(n+1)}$ (so that $N = l(n+1)$) be defined by

$$\Phi(\mathbf{u}, \mathbf{v}) = \{\varphi_j(\mathbf{u}) \cdot \mathbf{v}, \varphi_j(\mathbf{u})\}_{j=1}^l.$$

where $\mathbf{u} \in \mathbb{R}^k$ and $\mathbf{v} \in \mathbb{R}^n$. We then have for any $G \in \mathcal{K}$ and $u_1, ..., u_m \in V$ that

$$\frac{1}{m}\sum_{i=1}^m \Phi(u_i(\mathbf{x}), G(u_i)(\mathbf{y})) = \left\{ \frac{1}{m}\sum_{i=1}^m \varphi_j(u_i(\mathbf{x})) \cdot G(u_i)(\mathbf{y}), \frac{1}{m}\sum_{i=1}^m \varphi_j(u_i(\mathbf{x})) \right\}_{j=1}^l. \tag{27}$$

We claim that if $u_1, ..., u_m$ is a (δ, C)-discretization, then the sum

$$\frac{1}{m}\sum_{i=1}^m \varphi_j(u_i(\mathbf{x})) \tag{28}$$

is bounded uniformly from below.

To prove this, let $v_1, ..., v_t \in V$ be chosen so that $B(v_1(\mathbf{x}), \delta/3), ..., B(v_t(\mathbf{x}), \delta/3)$ is a covering of $V(\mathbf{x})$. It follows that one of these balls must contain at least $\lceil m/t \rceil$ elements $u_i(\mathbf{x})$ (note that the size m of the (δ, C)-discretization is not controlled). Using (22) this means each ball $B(u_i(\mathbf{x}), \delta)$ must contain at least $\frac{1}{C}\lceil m/t \rceil$ elements of the (δ, C)-discretization. Finally, using (25) this gives

$$\frac{1}{m}\sum_{i=1}^m \varphi_j(u_i(\mathbf{x})) \geq \frac{1}{m}\frac{1}{C}\left\lceil \frac{m}{t} \right\rceil \geq \frac{1}{Ct} > 0. \tag{29}$$

Now, define $\xi : \mathbb{R}^{l(n+1)} \to \mathbb{R}^{ln}$ to be

$$\xi(z_1, \cdots, z_{l(n+1)}) = \left\{ \frac{(z_j, z_{j+1}, \cdots, z_{j+n-1})}{z_{j+n}} \right\}_{j \in [1, (n+1)+1, 2(n+1)+1, \cdots, (l-1)(n+1)+1]},$$

when $|z_{j+n}| \geq 1/(Ct)$ for each j, and extend it continuously to the rest of $\mathbb{R}^{l(n+1)}$ using the Tietze extension theorem. Then, given any input from (27) where $u_1, ..., u_m$ is a (δ, C)-discretization, using (29) we will have

$$\xi\left(\frac{1}{m}\sum_{i=1}^m \Phi(u_i(\mathbf{x}), G(u_i)(\mathbf{y}))\right) = \left\{ \frac{\frac{1}{m}\sum_{i=1}^m \varphi_j(u_i(\mathbf{x})) \cdot G(u_i)(\mathbf{y})}{\frac{1}{m}\sum_{i=1}^m \varphi_j(u_i(\mathbf{x}))} \right\}_{j=1}^l := \{\overline{G(\widetilde{u}_j)(\mathbf{y})}\}_{j=1}^l.$$

Here the $\overline{G(\widetilde{u}_j)(\mathbf{y})}$ notation is used just to emphasize that this is just the weighted average of the $G(u_i)(\mathbf{y})$ samples that are in the ball of radius δ centered at $\widetilde{u}_j$. We then define, $\rho : \mathbb{R}^{k+l(n+1)} \to \mathbb{R}^n$, as

$$\rho(x, y) = \sum_{j=1}^l (\xi(y))_j \cdot \varphi_j(x), \tag{30}$$

so that we have

$$\rho\left(u_q(\mathbf{x}), \frac{1}{m}\sum_{i=1}^m \Phi(u_i(\mathbf{x}), G(u_i)(\mathbf{y}))\right) = \sum_{j=1}^l \overline{G(\widetilde{u}_j)(\mathbf{y})} \cdot \varphi_j(u_q(\mathbf{x})). \tag{31}$$

Now, fix any $G \in \mathcal{K}$. We will show that the above expression approximates $G(u_q)(\mathbf{y})$. We calculate

$$\left\| G(u_q)(\mathbf{y}) - \sum_{j=1}^{l} \overline{G(\widetilde{u}_j)(\mathbf{y})} \cdot \varphi_j(u_q(\mathbf{x})) \right\|_{\ell^\infty(\mathbf{y})} = \left\| \sum_{j=1}^{l} G(u_q)(\mathbf{y}) \cdot \varphi_j(u_q(\mathbf{x})) - \sum_{j=1}^{l} \overline{G(\widetilde{u}_j)(\mathbf{y})} \cdot \varphi_j(u_q(\mathbf{x})) \right\|_{\ell^\infty(\mathbf{y})}$$

$$= \left\| \sum_{j, \|u_q - \widetilde{u_j}\|_{\ell^\infty(\mathbf{x})} < 3\delta} (G(u_q)(\mathbf{y}) - \overline{G(\widetilde{u}_j)(\mathbf{y}))} \cdot \varphi_j(u_q(\mathbf{x})) \right\|_{\ell^\infty(\mathbf{y})}$$

Then, note

$$\left\| G(u_q)(\mathbf{y}) - \overline{G(\widetilde{u}_j)(\mathbf{y})} \right\|_{\ell^\infty(\mathbf{y})} = \left\| G(u_q)(\mathbf{y}) - \frac{\sum_{i=1}^m \varphi_j(u_i(\mathbf{x})) \cdot G(u_i)(\mathbf{y})}{\sum_{i=1}^m \varphi_j(u_i(\mathbf{x}))} \right\|_{\ell^\infty(\mathbf{y})}$$

$$= \left\| G(u_q)(\mathbf{y}) - \sum_{i=1}^{m} \frac{\varphi_j(u_i(\mathbf{x}))}{\sum_{i=1}^m \varphi_j(u_i(\mathbf{x}))} \cdot G(u_i)(\mathbf{y}) \right\|_{\ell^\infty(\mathbf{y})}$$

$$= \left\| \sum_{i=1}^{m} \frac{\varphi_j(u_i(\mathbf{x}))}{\sum_{i=1}^m \varphi_j(u_i(\mathbf{x}))} \cdot (G(u_q)(\mathbf{y}) - G(u_i)(\mathbf{y})) \right\|_{\ell^\infty(\mathbf{y})}$$

$$\leq \varepsilon/(2 \cdot \|\Psi\|_{C(K_2)}) \left| \sum_{i=1}^{m} \frac{\varphi_j(u_i(\mathbf{x}))}{\sum_{i=1}^m \varphi_j(u_i(\mathbf{x}))} \right| = \varepsilon/(2 \cdot \|\Psi\|_{C(K_2)})$$

Where we use that $\|u_q - \widetilde{u}_j\|_{\ell^\infty(\mathbf{x})} < 3\delta$, $\|\widetilde{u}_j - u_i\|_{\ell^\infty(\mathbf{x})} < 3\delta$, and (18) imply that

$$\|u_q - u_i\|_{C(K_1)} \leq \|u_q - \widetilde{u}_j\|_{\ell^\infty(\mathbf{x})} + \|\widetilde{u}_j - u_i\|_{\ell^\infty(\mathbf{x})} + \delta/6 < 37\delta/6, \tag{32}$$

so that by the choice of δ, $\|G(u_q) - G(u_i)\|_{C(K_2)} < \varepsilon/(2 \cdot \|\Psi\|_{C(K_2)})$.

So we have

$$\left\| G(u_q)(\mathbf{y}) - \sum_{j=1}^{l} \overline{G(\widetilde{u}_j)(\mathbf{y})} \cdot \varphi_j(u_q(\mathbf{x})) \right\|_{\ell^\infty(\mathbf{y})} \leq \varepsilon/(2 \cdot \|\Psi\|_{C(K_2)}),$$

which combining with (16) and using (31) gives us,

$$\|G(u_q) - \rho\left(u_q(\mathbf{x}), \frac{1}{m}\sum_{i=1}^{m} \Phi(u_i(\mathbf{x}), G(u_i)(\mathbf{y}))\right) \cdot \Psi\|_{C(K_2)}$$

$$\leq \|G(u_q) - G(u_q)(\mathbf{y}) \cdot \Psi\|_{C(K_2)} + \|G(u_q)(\mathbf{y}) \cdot \Psi - \rho\left(u_q(\mathbf{x}), \frac{1}{m}\sum_{i=1}^{m} \Phi(u_i(\mathbf{x}), G(u_i)(\mathbf{y}))\right) \cdot \Psi\|_{C(K_2)}$$

$$< \varepsilon/2 + \varepsilon/2 = \varepsilon.$$

Since $\|.\|_{C(K_2)} := \sup_{x \in K_2} \|.(x)\|$, this result then holds for any $x_q \in K_2$, as desired. This completes the proof.

□

B Experimental Details

B.1 Burgers' equation

We first sample 200,000 different ν's using log uniform sampling on the interval $[\log(0.001), \log(0.1)]$. For each ν, we generate 40 different initial conditions $u(x, 0)$'s. To generate these, we draw from a Gaussian process with the radial basis function $K(x, x') = \sigma^2 \exp(\frac{-(x-x')^2}{2\ell^2})$. We pick $\sigma = 0.8, \ell = 0.2$ to ensure our initial conditions have sufficient variability. We then apply a window function of $\sin^2(\pi x)$ to ensure the 0 boundary conditions are satisfied.

For a $\nu, u(x,0)$ pair, to solve the resulting PDE and get our reference solutions $u(x,1)$ we use an implicit explicit (IMEX) finite difference method (FDM), where we use forward difference in time, handle the nonlinear advection term $u\frac{\partial u}{\partial x}$ explicitly, and the diffusion term $\nu\frac{\partial^2 u}{\partial x^2}$ implicitly. We use 100 points to discretize the space $x \in [0,1]$, and 1000 points to discretize the time $t \in [0,1]$, to ensure shocks can be adequately captured.

Once the data is generated, we use a 90/10 training/testing split. During training, we use mini-batches of 5 operators, each with 39 in context examples (where the network predicts the 40th query). The architecture is trained for 100k iterations, using an Adam optimizer with an initial learning rate of 1e-4 and cosine learning-rate decay. Training took approximately 15 minutes, using an Nvidia RTX 4060 Ti GPU, to achieve a loss of 8.7e-5 (mean over the last 1000 iterations).

The DeepOSets architecture has the following hyperparameters. For the DeepSets encoder, we use an MLP ϕ with 4 layers of 800 neurons each. The pooled latent space has dimension 800 with mean pooling. The branch and trunk networks of the DeepONet both have 5 layers of 800 neurons, and both embed into a dimension 800 space. All networks use ReLU activation functions.

Once trained, we evaluate the model by calculating the relative L2 error for 100 randomly sampled operators form the test dataset, and repeat this for various in context sizes to see how the error changes. Comments on performance are found in 6

B.2 1D Darcy Flow

We first sample 200,000 different $k(x)$ coefficient functions to create our operators. For each $k(x)$, we generate 40 different forcing conditions (the RHS of the PDE) $f(x)$'s. To generate both the k's and f's, we draw from a Gaussian process with the radial basis function $K(x,x') = \sigma^2 \exp\left(\frac{-(x-x')^2}{2\ell^2}\right)$. For k, we pick $\sigma = 0.5, \ell = 0.4$ to ensure our initial conditions have sufficient variability without extremely drastic functions. With similar reasoning, for f, we pick $\sigma = 2, \ell = 0.4$. These choices also guarantee a range of solution u outputs of varying size and scale, which will help show our architecture can capture varying solution sizes.

For a k, f pair, to solve the resulting PDE and get our reference solutions $u(x)$ we use a second order FDM scheme, where the diffusion coefficient is evaluated at cell interfaces using arithmetic averaging ($k_{i+1/2} = \frac{1}{2}(k_i + k_{i+1})$). We use 50 points to discretize the space $[0,1]$.

Once the data is generated, we proceed similarly to the Burgers' case, using the same 90/10 split, training steps, and learning rate. We additionally use the same architeture hyperparameters as in Burgers', to show our architecture can learn various classes of operators without excessive tuning. Training took approximately 7 minutes on the same GPU, to achieve a loss of 9.0e-5.

Once trained, we evaluate the model by calculating the relative L2 error for 100 randomly sampled operators form the test dataset, and repeat this for various in context sizes to see how the error changes. Comments on performance are found in 6

B.3 2D Darcy Flow

This experiment was run in a similar fashion to the 1D Darcy flow, however we used 50,000 operators, 40 examples for each operator, with a 90/10 split. Additionally for 2D problems, random Fourier feature approximations are used to sample smooth random fields to generate $k(x)$ and $f(x)$, avoiding the memory cost of the full GP covariance matrix.

For a k, f pair, to solve the PDE and get our reference solutions $u(x)$ we use an finite element method (FEM), using a square mesh with bilinear quadrilateral elements. We use a 32×32 discretization of the space. We again train for 100k steps using Adam with an initial learning rate of 1e-4 and cosine learning-rate decay, to achieve a loss of 3.0e-5. Training took approximately 21 minutes on the same GPU, despite the 2D problem having much more complicated operators than the 1D case, and a much larger input size 32×32.

However, for the 2D case the architecture hyperparameters have to be changed so that they can adequately capture the more complicated 2D fields. Hence, for the DeepSets encoder, we use an MLP ϕ with a first layer of 1500 neurons, and 3 layers of 1200 neurons each. The pooled latent space has dimension 1200 with mean pooling. The branch network has 2 layers of 1000 neurons,

and 3 layers of 800 neurons, and the trunk network has 5 layers of 1000 neurons, and both embed into a dimension 1200 space. All networks use ReLU activation functions. Once trained, we evaluate the model by calculating the relative L2 error for 100 randomly sampled operators form the test dataset, and repeat this for various in context sizes to see how the error changes. Comments on performance are found in 6

B.4 Multiple differential equation operators

We set $u(0) = 0$ for the forward version of the three initial value problems (ODEConst, ODE1, and ODE2). For the boundary value problems (Poisson, Linear Reaction-Diffusion, and Nonlinear Reaction-Diffusion), the boundary values $u(0)$ and $u(1)$ are unknown parameters in both the forward and inverse versions, and must be learned in-context. In our experiments, all parameters (i.e., equation coefficients and boundary values) are chosen randomly. The physical domain in all cases is the interval $[0, 1]$, which is discretized uniformly into 100 points. The prompts during both training and testing contain 4 example pairs of parameter and solution functions. The MLP ϕ in the DeepOSets contains 2 hidden layers of 500 neurons each, and the latent vector encoding the prompt data has size 500. The pooling operation is the sample mean. The branch and trunk networks in the DeepOSets contain hidden layers of 200, 500, and 200 neurons with intermediate layer of 500 neurons. The pooled latent space is 2000 with mean pooling. All neural networks use the ReLU activation function.

To generate the training data for each of the problems, 100 parameter functions are sampled independently from a Gaussian process with the squared-exponential kernel (discretized on the uniform 100-point grid) with variance 2 and length scale 0.5. For each boundary-value problem, the boundary values u_0 and u_1 are sampled uniformly in the interval $[-1, 1]$. For the reaction-diffusion problems, the coefficients a and c are sampled uniformly in the intervals $[0.025, 0.075]$ and $[-2, 2]$, respectively. Then the equation is solved for the given parameters to obtain the ground truth solutions. The model was trained by minimizing the mean-squared error loss between predicted and ground truth solutions over a mini-batch of 100 operators sampled randomly from the training data at each iteration, using the Adam optimizer with learning rate 1e-3, for 50k iterations.

To evaluate how a broader task spectrum influences performance, we trained our model on configurations of 6 and 12 tasks. Training took 12min for the six-task experiment, achieving a training loss 8.4e-3, while the training time was 38 minutes for the 12-task experiment, achieving a training loss of 4.8e-3. Both experiments were run on an Nvidia 4060 Ti GPU with 16GB memory. We contrast this to the training time required by the complex architecture in [26], which is measured in tens of hours on the same computing hardware. Sample results are shown in Figure 9 and MSE values on test data from operators not seen during training are displayed in Table 1. DeepOSets achieves accurate predictions without weight updates at inference time, without knowing which task or the values of the equation coefficients and boundary conditions. All these unknowns are correctly disambiguated from the examples in the prompt.

B.5 ICON comparison

Both DeepOSets and ICON were implemented in PyTorch; the ICON implementation is based on the code released by the ICON authors (a 6-layer Transformer encoder with model dimension 256 and 8 attention heads, which processes the context pairs and the query serialized into a sequence of 850 tokens for $m = 5$ context pairs). The DeepOSets model uses 4-hidden-layer ReLU MLPs of width 256 for the set encoder, branch, and trunk networks. Both models are trained with Adam (learning rate 10^{-4}) on the mean-squared error for the same number of steps, and evaluated on 100 random test samples for $n_{\mathrm{ctx}} \in \{1, \ldots, 5\}$ in-context examples

The dataset for the benchmark is generated synthetically using finite differences and Gaussian process priors, with the settings in Table 4. The diffusivity is sampled as $k = \exp(\xi)$ with $\xi \sim \mathcal{GP}(0, k_{\mathrm{RBF}})$, $\sigma = 0.5$, $\ell = 0.4$, on the full grid of $[0, 1]$ (52 points including the boundaries). The source term $f(x)$ is sampled from $\mathcal{GP}(0, k_{\mathrm{RBF}})$ with $\sigma = 2.0$, $\ell = 0.4$ on the 50 interior points. The solution $u(x)$ is computed for each (k, f) pair with the Thomas algorithm for the tridiagonal finite-difference system.

Setting	Value
Grid points N	50 (interior)
Total operators	5,000
Pairs per operator	40
Train / test split (operator level)	90% / 10%
Random seed	42

Table 4: Dataset settings for the 1D Darcy-flow benchmark used in the ICON comparison.

The DeepOSets architecture used in the experiment was as follows. For a query (f_q, x) and context pairs $(f_i, u_i)_{i=1}^m$, the forward pass is

$$h = \frac{1}{m}\sum_{i=1}^{m} \phi([f_i; u_i]), \qquad \hat{u}(x) = \mathrm{branch}([f_q; h]) \cdot \mathrm{trunk}(x) + b.$$

All sub-networks are 4-hidden-layer MLPs with ReLU activations and 256 neurons per hidden layer: the set encoder ϕ maps $\mathbb{R}^{2N} = \mathbb{R}^{100}$ to $\mathbb{R}^{256}$, the branch network maps $\mathbb{R}^{N+256} = \mathbb{R}^{306}$ to $\mathbb{R}^{256}$, and the trunk network maps $\mathbb{R}$ to $\mathbb{R}^{256}$. The model has 894,465 trainable parameters.

The ICON architecture used in the experiment is described next. Context pairs and the query are serialized into a token sequence and processed by a Transformer encoder; the output is read from the final N tokens. For m context pairs with N grid points each, the total sequence length is $(3m + 2) \times N = 17 \times 50 = 850$ tokens at $m = 5$. Each token is a 2D vector (x_j, v_j) projected to the model dimension via a linear layer, and a learned positional embedding (indexed by role: f, u, placeholder, or query) is added. The hyperparameters are listed in Table 5; the model has 4,744,193 trainable parameters.

Hyperparameter	Value
Model dimension	256
Transformer layers	6
Attention heads	8
FFN widening factor	4
Activation	GELU
Normalization	Pre-norm (LayerNorm)
Sequence length	850

Table 5: ICON hyperparameters used in the comparison.

Both models were trained in isolated subprocesses to ensure fair GPU memory separation using Adam with learning rate 10^{-4} and the mean-squared error loss, with batch size 5 for DeepOSets and 32 for ICON. At each training step, a random operator index and a random permutation of its pairs are drawn; the first n_{ctx} pairs form the context and one additional pair is the query. Two equal-step schedules were run, of 10,000 and 100,000 training steps. Evaluation is performed over 100 random test samples for each $n_{\mathrm{ctx}} \in \{1, 2, 3, 4, 5\}$, using the relative ℓ^2 error on the output grid. Each inference forward pass is timed individually with `torch.cuda.synchronize()` before and after, to obtain accurate GPU wall-clock time; the reported value is the mean over all 500 timed calls (100 per n_{ctx} level). Table 6 reports the cost measurements of the 10k-step run, which complement the 100k-step measurements of Table 2.

Metric	DeepOSets	ICON
Trainable parameters	894,465	4,744,193
Training steps completed	10,000	10,000
Training time (s)	42.5	7,746.2
Inference time / sample (ms)	0.670	8.042

Table 6: Cost measurements for the 10k-step run of the ICON comparison.

C Prompt ablation

To verify that DeepOSets actually uses the in-context examples to disambiguate the operator, we performed prompt ablations on the trained Burgers', 1D Darcy and 2D Darcy models. *No context:* replacing the pooled prompt embedding by a zero vector causes performance to degrade sharply; the predictions collapse to (approximately) the spatial mean of all dataset solutions, yielding high relative L_2 errors in the range 0.45–0.62. *Mismatched context:* when prompt pairs $(u_i, G_{\mathrm{op}_1}(u_i))$ from one operator are provided while the query function u_q is evaluated under a different operator G_{op_2}, the model outputs the solution corresponding to G_{op_1}, confirming that the prompt controls the operator selection. *Shuffled context:* permuting the order of the context pairs $(u_i, G(u_i))_{i=1}^m$ yields identical predictions, up to numerical noise below 10^{-7}, as expected from the exact permutation invariance of the DeepSets encoder. *Mispaired context:* permuting the outputs $G(u_i)$ while keeping the inputs u_i fixed, so that each input is paired with the wrong output, degrades the predictions substantially.

D Nearest-neighbor baseline with sample size ablation

A natural question is whether, given a large training set, the task could be solved by simple lookup rather than by in-context learning. To test this, we implemented a parameter-free nearest-neighbor (NN) baseline on the 1D Darcy flow problem: given the n_{ctx} in-context (condition, solution) pairs describing a new test operator, the baseline searches the entire training set for the single operator whose example pairs are jointly closest (in both condition and solution) to the given in-context examples, and then predicts the query solution by returning, from that training operator's own data, the solution whose condition is closest to the query condition. The nearest-neighbor baseline has no learned parameters and proceeds in two steps. (1) Given the n_{ctx} in-context (condition, solution) pairs describing a new test operator, search the entire training set for the single operator whose in-context example pairs are jointly closest (matching both condition and solution in each pair) to the given in-context examples. (2) Predict the query solution by looking up, within that nearest training operator's own data, the pair whose condition is closest to the query condition, and returning its solution. For the training-set-size ablation, DeepOSets was retrained from scratch on nested training subsets of $K \in \{100, 500, 1000, 2000, 4500\}$ operators with a fixed held-out set of 500 test operators, and the NN baseline used the same nested subsets as its lookup pool. We also ran a training-set-size ablation, retraining DeepOSets from scratch on nested training subsets of $K \in \{100, 500, 1000, 2000, 4500\}$ operators (out of 5,000 total, with a fixed held-out set of 500 test operators never seen during training), and evaluating the NN baseline with the same nested subsets as its lookup pool, so that both methods see identical training data at every scale. The results are reported in Table 7.

K (training operators)	DeepOSets	NN baseline
100	0.081	0.557
500	0.071	0.572
1000	0.063	0.592
2000	0.082	0.545
4500 (full)	0.082	0.607

(a) Error at $n_{\mathrm{ctx}} = 5$ vs. training-set size K.

n_{ctx}	DeepOSets	NN baseline	ICON
1	0.210	0.692	0.134
2	0.118	0.613	0.083
3	0.098	0.563	0.059
4	0.082	0.654	0.057
5	0.082	0.607	0.060

(b) Error vs. n_{ctx} at full training-set size $K = 4500$.

Table 7: Mean relative L_2 test error on the 1D Darcy flow benchmark for DeepOSets and the nearest-neighbor (NN) baseline, on a fixed held-out set of 500 test operators. ICON (trained for 100k steps), is included in (b) as an additional point of reference.

The NN baseline's error remains high (relative L_2 error between 0.55 and 0.61) at every training-set size, and does not shrink as K grows from 100 to 4,500 operators; we see no evidence that the lookup becomes systematically easier simply because the pool is larger. In contrast, DeepOSets achieves a relative L_2 error of 0.06–0.08 at every training-set size tested, including the most data-scarce setting of $K = 100$ operators (2% of the full training pool). The margin between DeepOSets and the NN baseline is roughly 7–9$\times$ throughout and does not narrow as more training data is added. These results show that the task is not solved by DeepOSets simply because enough training data is

available to make a trivial lookup sufficient, and that the in-context performance of DeepOSets does not depend on having a very large training set.

E Additional Experiments

Figures 4–9 present additional plots and results of the experiments in the paper.

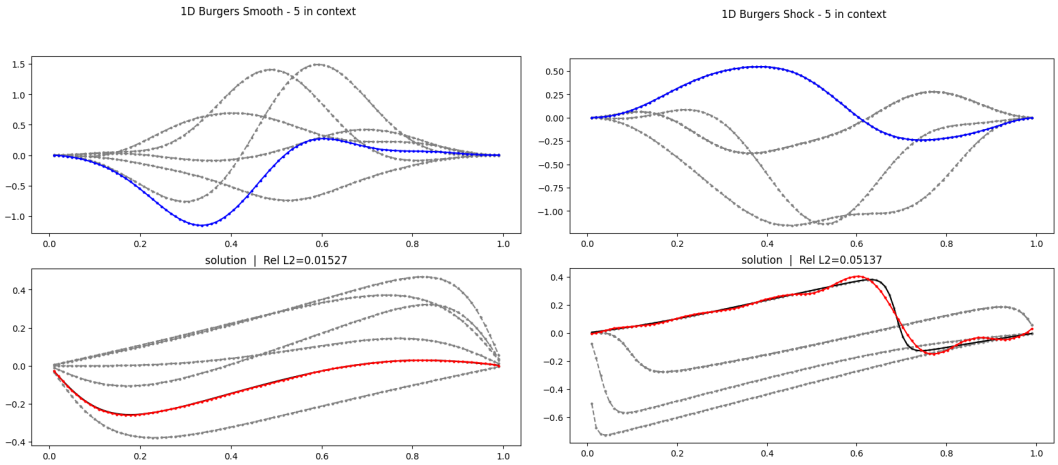


Figure 4: Representative testing examples for 1D Burgers, both with 5 in-context examples. Here, the upper graphs with the blue functions represent the in-context query functions, in this case the initial condition $u(x, 0)$. The gray functions are the in-context examples $u(x, 0)$, and the blue one is the final query. The lower graphs represent the 5 in-context solutions that were given to the network, followed by the network prediction in red, and compared with the true FDM solution in black.

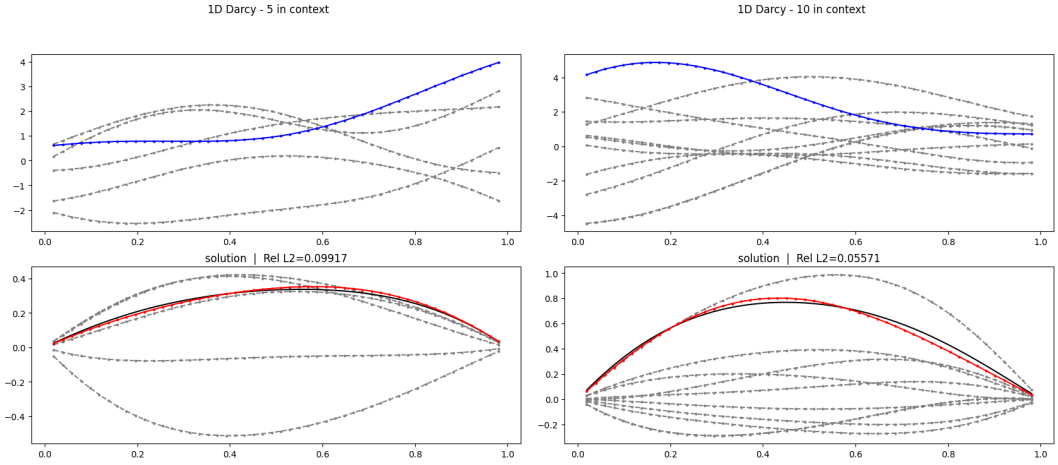


Figure 5: Representative testing examples for 1D Darcy with 5 and 10 in-context examples for different operators. The graph is formatted in the exact same manner as the one for Burgers' equation. Here, we see a noticeable reduction in L_2 error by simply increasing 5 more examples.

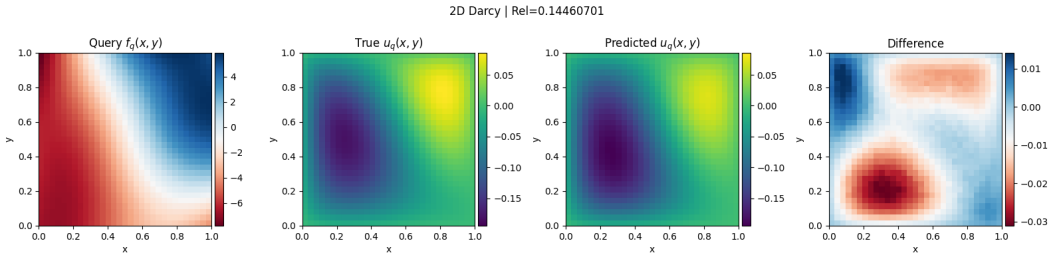


Figure 6: 2D Darcy with 5 in-context examples, relative error stated in title, we show the query function, true u, our network prediction, and the difference

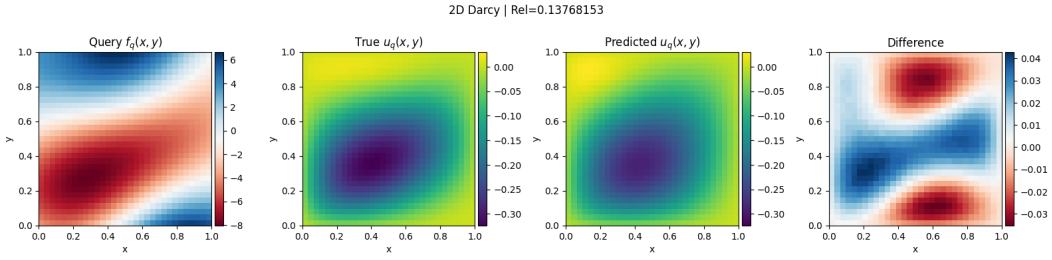


Figure 7: 2D Darcy with 10 in-context examples, a slight decrease in relative error, indicating the network still needs additional context

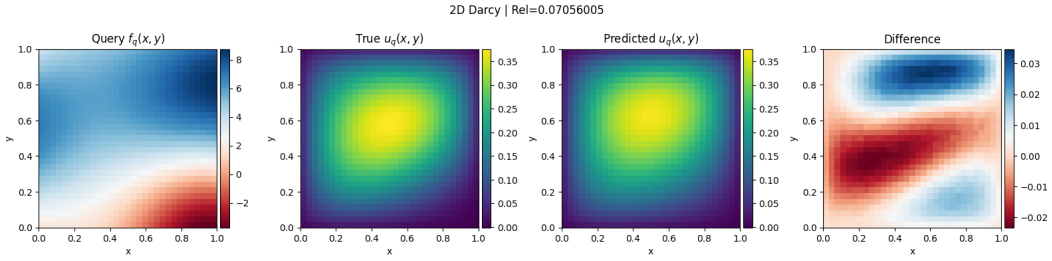


Figure 8: 2D Darcy with 20 in-context examples, a much larger decrease in relative error, showing that the network is benefitting from the additional context

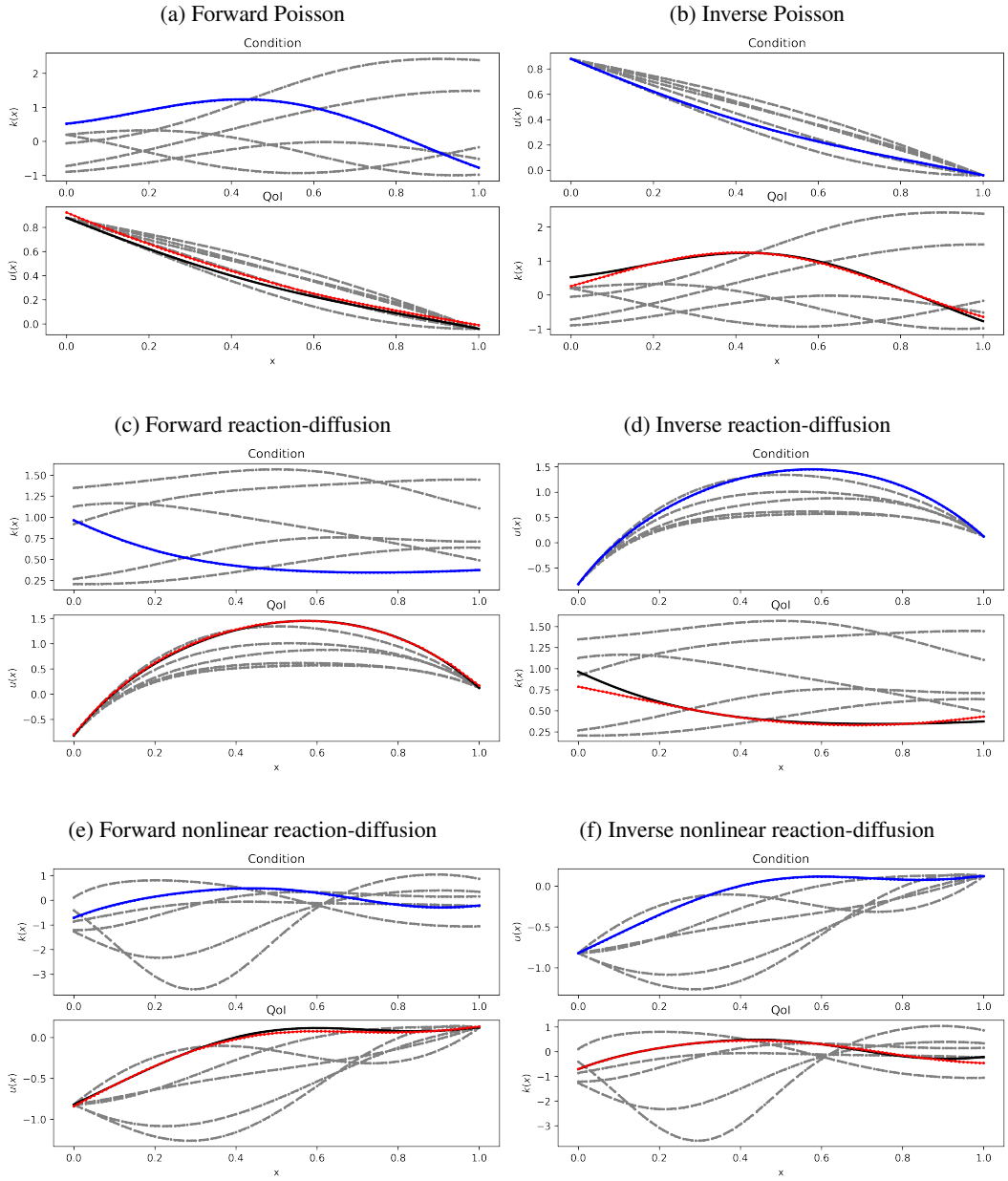


Figure 9: Sample results from the 12-task numerical experiment. In each case, the top plot contains the parameter functions while the bottom plot contains the corresponding solution functions. The gray lines indicate in each case the parameter-solution function pairs given in the prompt. The blue line indicates the query parameter function, while the red line indicates the corresponding solution predicted by the trained DeepOSets model. The black line indicates the exact solution corresponding to the query, using the appropriate equation.

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: The claims made in the abstract and introduction do accurately reflect the papers constribution and scope.

   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: Yes, we have added a discussion of the limitations of our results and experiments in the paper.

   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: [Yes]

Justification: We provide a detailed description of all assumptions, and a complete, detailed, and (as far as we can tell) correct proof.

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 provide a complete description of the experiments performed, including the data generation process. We also include the code used to run the experiments. This code will be publicly released, together will all of the data generated, upon publication.

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: All data and code used in the experiments will be publically released following the review process.

   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: All of these details are contained 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: All experiments are run multiple times with different random seeds. Error bars are reported in the paper and a detailed explanation of how they were calculated is provided in the Appendix.

   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: All details regarding the computing resources used are 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: No issues from the NeurIPS Code of Ethics arose for this work.

   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: [N/A]

    Justification: Our work studies theoretical and practical issues related to scientific computing using neural networks. As such, it is fundamental research which is not immediately applicable. For this reason, we don't forsee our work having a direct societal impact.

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: Out research poses no reasonably foreseeable risks.

    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: [N/A]

    Justification: No existing assets requiring licences are used.

    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: All code released will be well-documented.

    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: No crowdsouring or research with human subjects was involved.

    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: No crowdsouring or research with human subjects was involved.

    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: Our use of LLMs is not necessary for reproducing our experiments. For this reason, our usage of LLMs was not an important, original, or non-standard component of the research.

    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.