In-Context Operator Learning for Differential Equation Problems

Liu Yang[1], Siting Liu[1], Tingwei Meng[1], Stanley J. Osher[1*]

[1]Department of Mathematics, UCLA, Los Angeles, CA 90095, USA

Abstract

This paper introduces a new neural-network-based approach, namely IN-context Differential Equation Encoder-Decoder (INDEED), to simultaneously learn operators from data and apply it to new questions during the inference stage, without any weight update. Existing methods are limited to using a neural network to approximate a specific equation solution or a specific operator, requiring retraining when switching to a new problem with different equations. By training a single neural network as an operator learner, we can not only get rid of retraining (even fine-tuning) the neural network for new problems, but also leverage the commonalities shared across operators so that only a few demos are needed when learning a new operator. Our numerical results show the neural network's capability as a few-shot operator learner for a diversified type of differential equation problems, including forward and inverse problems of ODEs and PDEs, and also show that it can generalize its learning capability to operators beyond the training distribution, even to an unseen type of operator.

1 Introduction

The development of neural networks has brought a significant impact on solving differential equation problems. We refer the readers to [1] for the recent advancement in this topic.

One typical approach aims to directly approximate the solution given a specific problem. The Physics-Informed Neural Networks (PINNs) [2] propose a neural network method for solving both forward and inverse problems by integrating both data and differential equations in the loss function. Deep Galerkin Method (DGM) [3] imposes constraints on the neural networks to satisfy the prescribed differential equations and boundary conditions. Deep Ritz Method (DRM) [4] utilizes the variational form of PDEs and can be used for solving PDEs that can be transformed into equivalent energy minimization problems.

*Corresponding Author: sjo@math.ucla.edu

Weak Adversarial Network (WAN) [5] leverages the weak form of PDEs by parameterizing the weak solution and the test function as the primal and adversarial networks, respectively. [6] proposed a deep learning method based on the stochastic representation of high-dimensional parabolic PDEs. [7] solves high-dimensional mean-field game problems by encoding both Lagrangian and Eulerian viewpoints in neural network parameterization. APAC-net [8] proposes a generative adversarial network style method that utilizes the primal-dual formulation for solving mean-field game problems.

Despite their success, the above methods are designed to solve problems with a specific differential equation. The neural network needs to be trained again when the terms in the equation or the initial/boundary conditions change.

Later, efforts have been made to approximate the solution operator for a differential equation with different parameters or initial/boundary conditions. PDE-Net [9] utilizes convolution kernels to learn differential operators, allowing it to unveil the evolution PDE model from data, and make forward predictions with the learned solution map. Deep Operator Network (DeepONet) [10] designed a neural network architecture to approximate the solution operator which maps the parameters or the initial/boundary conditions to the solutions. Fourier Neural Operator (FNO) [11] utilizes the Fourier transform to learn the solution operator.

The above methods have successfully demonstrated the capability of neural networks in approximating solution operators. However, in these methods, "one" neural network is limited to approximating "one" operator. Even a minor change in the differential equation can cause a shift in the solution operator. For example, in the case of learning a solution operator mapping from the diffusion coefficient to the solution of a Poisson equation, the solution operator changes if the source term (which is not designed as a part of the operator input) changes, or a new term is introduced to the equation. Consequently, the neural network must be retrained.

We argue that there are commonalities shared across various solution operators. By using a single neural network with a single set of weights to learn various solution operators, we can not only get rid of retraining (even fine-tuning) the neural network, but also leverage such commonalities so that fewer data are needed when learning a new operator.

If we view learning one solution operator as one task, then we are now targeting solving multiple differential-equation-related tasks with a single neural network. Our expectation for this neural network goes beyond simply learning a specific operator. Rather, we expect it to acquire the ability to "learn an operator from data" and apply the newly learned operator to new problems.

Such ability to learn and apply new operators might be a very important part of artificial general intelligence (AGI). By observing the inputs and outputs of a physical system, a human could learn the underlying operator bridging inputs to outputs, and control the system according to their goals. For example, a motorcyclist can quickly adapt to a new motorcycle; a kayaker can quickly adapt to a new kayak or varying water conditions. If a human has expertise in both sports, they may be able to master jet skiing at their first few attempts.

We expect a robot with AGI able to adapt to new environments and tasks, just as a human would.

The paradigm of "learning to learn", or meta-learning, has achieved great success in the recent development of artificial intelligence. In natural language processing (NLP), in-context learning introduced in GPT-2 [12] and further scaled up in GPT-3 [13] has demonstrated the capability of large language models as few-shot learners. In-context learning gets rid of the limitations of the previous paradigm, i.e. pre-training plus fine-tuning, including (1) the need to fine-tune the neural network with a relatively large dataset for every new task, (2) the potential to overfit during fine-tuning which leads to poor out-of-distribution generalization, and (3) the lack of ability to seamlessly mix together or switch between multiple tasks and skills.

In this paper, we adapt the idea of in-context learning to learn operators in differential equations problems.

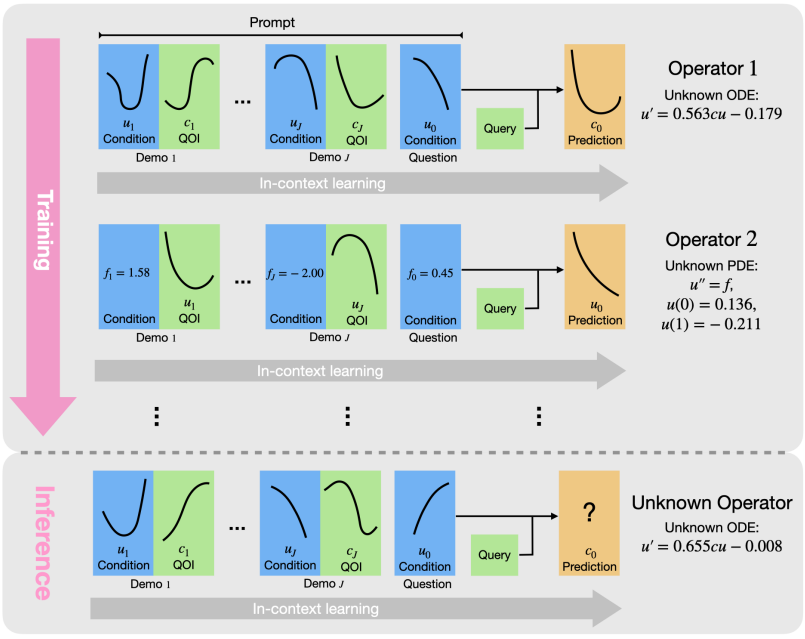

Figure 1: Training and inference of IN-context Differential Equation Encoder Decoder (INDEED).

To avoid confusion between the outputs of operators and neural networks, we refer to the inputs of the operators as "conditions", and the operator outputs as "quantities of interest (QOI)". A "demo" consists of one pair of condition and QOI.

In the previous paradigm of operator learning [10] [11], the neural network is trained on demos that share the same operator. During the inference stage,

it takes a new condition as input and predicts the QOI corresponding to the learned operator. There are other non-neural-network methods in the similar paradigm, such as SINDy [14] for solving inverse ODE problems via sparse regression over ODE parameters. In this paper, during the inference stage, we instead have the trained neural network taking the demos and a new condition (namely "question condition") as input, and simultaneously completing the following two jobs: (1) learn the operator from demos, (2) apply the learned operator to the question condition and predict the QOI. We emphasize that there are no weight updates during the inference stage. Since an encoder-decoder architecture is employed for the neural network, we name the proposed method as IN-context Differential Equation Encoder Decoder, or "INDEED" in short. The illustration of the training and inference of INDEED is shown in Figure 1.

The rest parts of the paper are organized as follows. In Section 2, we introduce the detailed methodology of INDEED. In Section 3, we present the experimental results, where we show the capability of INDEED in learning operators from demos and applying to question conditions during the inference stage. In Section 4, we try to answer the question why several demos are sufficient in learning the operator in the proposed method. In Section 5, we conclude the paper and discuss the limitations and future work.

2 Methodology

In this section, we will introduce the method in detail. In Section 2.1, we show how to build the neural network inputs, including prompts and queries. In Section 2.2, we illustrate the neural network architecture. We introduce the data preparation and training process in Section 2.3, and the inference process in Section 2.4.

2.1 Prompt and Query

The model is expected to learn the operator from multiple demos, each consisting of a pair of condition and QOI, and apply it to the question condition, making predictions on the question QOI. As the QOI is typically a function, it's also necessary to specify where the model should evaluate the QOI function, which is referred to as the "query". We group the demos and questions as "prompts", which together with the queries are the neural network inputs.

Take a one-dimensional ODE problem $u'(t) = \alpha u(t) + \beta c(t) + \gamma$ as an example. Given the parameters $\alpha, \beta, \gamma \in \mathbb{R}$, there exists a solution operator that maps from the function $c\colon [0,T] \to \mathbb{R}$ and the initial condition $u(0)$, to $u\colon [0,T] \to \mathbb{R}$. If we know the operator, then we can make predictions of $u\colon [0,T] \to \mathbb{R}$ with observations on $u(0)$ and $c\colon [0,T] \to \mathbb{R}$. However, in many real-world problems, the values of the parameters α, β, γ, even the structure of the ODE may not be available, hence the operator is unknown. In such cases, we need to infer the operator from pairs of conditions and quantities of interest, which are called "demos". After learning the operator, we can use it to make predictions on a new

condition, namely the "question condition". Here we remark that, the question and demos must share the same operator, in particular, the same parameters α, β, γ in this ODE problem.

Typically, both the condition and the QOI in a given problem are functions (or observations of functions), and may comprise various components. To represent these entities in a generalizable manner, we employ key-value pairs. Still taking the one-dimensional ODE problem introduced above as an example, the condition contains the function $c\colon [0,T] \to \mathbb{R}$ and the initial condition $u(0)$, while the QOI is the solution $u\colon [0,T] \to \mathbb{R}$. To represent the function $c\colon [0,T] \to \mathbb{R}$, we use the discretized time instants $t_1, \ldots, t_{n-1}$ as the keys, and the corresponding function values $c(t_1), \ldots, c(t_{n-1})$ as the values. Similarly, we use the key 0 and value $u(0)$ to represent the initial condition of u. To further distinguish $c\colon [0,T] \to \mathbb{R}$ and initial condition, we need to augment an additional dimension to the key, which serves as the indicator of $c\colon [0,T] \to \mathbb{R}$ and initial condition. For example, now we can use $(t_1, 0), (t_2, 0) \ldots, (t_{n-1}, 0)$ as the keys for $c\colon [0,T] \to \mathbb{R}$, and $(0, 1)$ as the key for $u(0)$. In total, we have n key-value pairs to represent the condition. In a similar manner, we can use m key-value pairs to represent the QOI $u\colon [0,T] \to \mathbb{R}$, with the discretized time instants $\tau_1, \ldots, \tau_m$ as keys and $u(\tau_1), \ldots, u(\tau_m)$ as values.

One condition-QOI pair forms one demo. To further distinguish the conditions and QOIs of different demos, we use a column index vector, denoted by e_j, for the j-th demo. To distinguish the condition and QOI of the same demo, we use a positive index vector e_j for the condition and a negative index vector $-e_j$ for the QOI. For simplicity, we adopt a one-hot column vector as e_j in this paper. However, for a large number of demos, a more efficient representation, such as the position embedding used in NLP tasks, can be applied. The size of the one-hot vector is equal to the number of demos plus one, with the additional one index reserved for the question condition.

Now, for each demo we create a matrix representation by concatenating the condition and QOI, where each of them consists of keys, values, and index vectors. In table 1, we show the matrix representation of the j-th demo for the above one-dimensional ODE problem.

	condition					QOI			
key	t_1	t_2	$\ldots$	t_{n_j-1}	0	τ_1	τ_2	$\ldots$	τ_{m_j}
	0	0	$\ldots$	0	1	0	0	$\ldots$	0
value	$c(t_1)$	$c(t_2)$	$\ldots$	$c(t_{n_j-1})$	$u(t_1)$	$u(\tau_1)$	$u(\tau_2)$	$\ldots$	$u(\tau_{m_j})$
index	e_j	e_j	$\ldots$	e_j	e_j	$-e_j$	$-e_j$	$\ldots$	$-e_j$

Table 1: The values of condition and QOI in the j-th demo in the example of solving the one-dimensional ODE $u'(t) = \alpha u(t) + \beta c(t) + \gamma$. The keys for QOI are padded with zeros to align with the keys for condition. The index vector e_j is a column vector.

The question condition is represented in the same key-value format as the demo conditions, with the number of key-value pairs being denoted as n_0. To

differentiate the question condition from the demo conditions, we set the index vector e_0 in the question condition to $(0....0, 1)^T$.

In the end, we concatenate the demos and the question condition to form a matrix representation, namely the "prompt".

The keys for the question QOI act as the "query", while the values for the question QOI are the targets for the neural network to predict, namely the "label" or "ground truth".

We make several remarks here:

- We emphasize that the demos and the question must share the same operator.
- We can design the queries based on where we wish to evaluate the question QOI function.
- The dimensions (or the number of rows) of keys may vary depending on the system and the corresponding operator. However, in order to form a matrix representation that can be fed into a single neural network, we need to pre-set the dimension for keys large enough, and pad the original keys (of different dimensions) with zeros. As an example, in Table 1, we padded the keys for QOI with zeros. We do the same for values as well, but the value dimension can be different from the key dimension.
- The number of key-value pairs in conditions and QOIs also varies. In other words, the length (or the number of columns) of the prompt and query varies. Although transformers are designed to handle variable length inputs, we still pad zeros in length for batching purposes. Such padding, combined with proper masks, does not affect mathematics.
- In principle, the number of demos may also vary across prompts. Our method can handle variable numbers of demos by switching to a more compact and flexible index representation, or simply setting the size of the one-hot index vector as one plus the max number of demos. For simplicity, we fix the number of demos in this paper.

2.2 Neural Network Architecture

We employ an encoder-decoder neural network architecture in our method. The encoder is a self-attention transformer, which takes the prompt as input. The decoder is a cross-attention transformer, whose input key and value are the output of the encoder, while the queries of the attention layers are the keys for the question QOI, i.e. the "query" introduced in Section 2.1. We expect the output of the decoder to predict the values for the question QOI, i.e. the "label" introduced in Section 2.1. The input and output of neural networks are shown in Figure 2. Such an encoder-decoder architecture is similar to the one used for object detection tasks in computer vision [15].

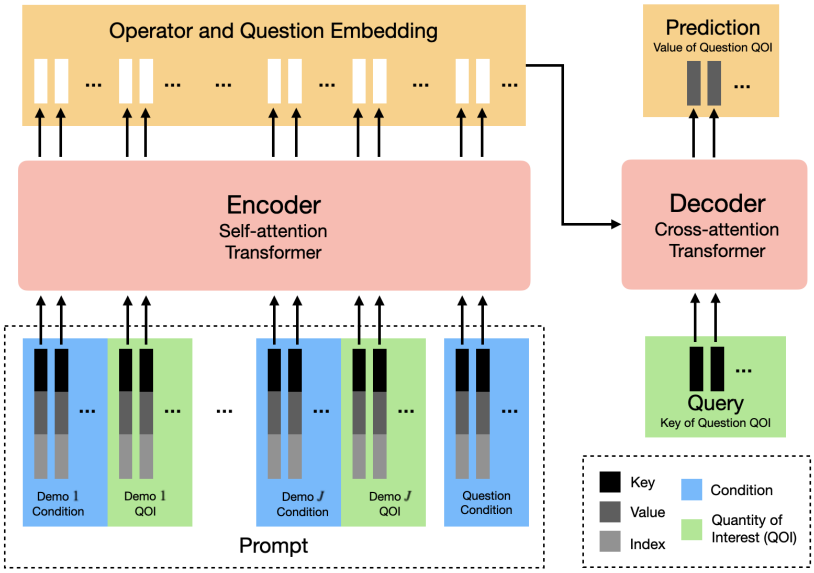

Figure 2: The neural network architecture for in-context differential equation encoder-decoder (INDEED).

The self-attention encoder mixes information from all demos and the question condition from the prompt, and the output can be interpreted as an embedding of the operator and the question. The order of columns in the prompt won't affect the result.

With a fixed prompt, if we input a query of n columns (i.e., n keys) into the model and receive n corresponding values as output, the i-th value is solely determined by the i-th key, independent of the others. This means we can make predictions on a query with arbitrary columns, or break a query into multiple pieces, making predictions one by one.

2.3 Data Preparation and Training

Before training the neural network, we prepare data which contains the numerical solutions to different kinds of differential equation problems. The details of data gereration is described in Algorithm 1.

In the training process, in each iteration, we randomly build a batch of prompts, queries, and labels (ground truths) from data. Note that different problems with different operators appear in the same batch. The loss function is the mean squared error (MSE) loss between the outputs of the neural network and the labels. If the values of the question QOI are padded with zeros, then we need to mask the padded values in the loss function. The details of the training process are described in Algorithm 2.

Algorithm 1: Data preparation.

```
for each type of problem do
    Randomly generate M sets of parameters;
    // Each set of parameters defines an operator
    for each set of paramters do
        Randomly generate N pairs of conditions and QOIs;
        // These N pairs of conditions and QOIs share the same
          solution operator
    end
end
```

Algorithm 2: The training and inference of in-context differential equation encoder-decoder (INDEED).

```
// Training stage:
for i = 1, 2, ..., training steps do
    for b = 1, 2, ..., batch size do
        Randomly select a type of problem and a set of parameters from
          dataset;
        From N pairs of conditions and QOIs, randomly select J pairs
          as demos and one pair as the question;
        Build a prompt, a query, and a label (the ground truth) using
          the selected demos and question;
    end
    Use the batched prompts, queries and labels to calculate the MSE
      loss and update the neural network parameters with gradients;
end
// Inference stage:
Given a new system with an unknown operator, collect J demos and a
  question condition, and design the query;
Construct the prompt using the demos and question condition;
Get the prediction of the question QOI using a forward pass of the
  neural network;
```

2.4 Inference: Few-shot Learning without Weight Update

After the training of the neural network, we use the trained neural network to make predictions of the question QOI based on a few demos that describes the solution operator, as well as the question condition.

Note that during one forward pass, the neural network finishes the following two tasks simultaneously: it learns the solution operator from the demos, and applies the learned operator to the question condition for predicting the question QOI. We emphasize that the neural network does not update its weights during such a forward pass. In other words, the trained neural network acts as a few-shot learner, and the training stage can be perceived as "learning to learn".

3 Numerical Results

To test the proposed method, we designed 13 types of problems, each of which has 1000 sets of parameters, so $13 \times 1000 = 13000$ operators in total. For each parameter set, we generate 100 condition-QOI pairs. To build one prompt, we randomly select 5 condition-QOI pairs for the demos, and 1 pair for the question. In other words, $M = 1000, N = 100, J = 5$ in Algorithm 1 and 2. In total, we have $13 \times 1000 \times 100^{(5+1)} = 1.3 \times 10^{16}$ possible prompts, which is much larger than the total of prompts used during training.

3.1 Problems

In this subsection, we list the details of all 13 types of problems, as well as the setups for parameters and condition-QOI pairs in the data preparation stage (Algorithm 1).

Problem 1 forward problem of ODE 1

In this problem, the governing dynamic is $u'(t) = a_1 c(t) + a_2$ over time interval $[0, 1]$.

- parameters: a_1, a_2
- condition: $c(t), u(0)$
- QOI: $u(t)$

$a_1 \sim \mathcal{U}(0.5, 1.5)$, i.e. randomly sampled from uniform distribution from $[0.5, 1.5]$, $a_2 \sim \mathcal{U}(-1, 1)$. $u(0) \sim \mathcal{U}(-1, 1)$. $c(t) \sim \mathcal{GP}(0, k(x, x'))$, i.e. sampled from a Gaussian process with zero mean and kernel $k(x, x')$, where

$$k(x, x') = \sigma^2 \exp(-\frac{1}{2l^2}(x - x')^2), \sigma^2 = 1, l = 0.5. \tag{1}$$

- condition key: $\begin{pmatrix} t_1 & t_2 & \dots & t_{n-1} & 0 \\ 0 & 0 & \dots & 0 & 1 \end{pmatrix}$, where $t_1, t_2, ..., t_{n-1} = 0, 0.02, ..., 0.96$
- condition value: $\begin{pmatrix} c(t_1) & c(t_2) & \dots & c(t_{n-1}) & u(t_1) \end{pmatrix}$

- QOI key: $\begin{pmatrix}\tau_1 & \tau_2 & \dots & \tau_m\end{pmatrix}$, where $\tau_1, \tau_2, ..., \tau_m = 0, 0.02, ..., 0.98$
- QOI value: $\begin{pmatrix}u(\tau_1) & u(\tau_2) & \dots & u(\tau_m)\end{pmatrix}$

Problem 2 inverse problem of ODE 1

In this problem, the governing dynamic and parameter setups are the same as in Problem 1, but the condition and QOI are changed to $u(t)$ and $c(t)$, respectively.

- condition key: $\begin{pmatrix}t_1 & t_2 & \dots & t_n\\ 0 & 0 & \dots & 0\end{pmatrix}$, where $t_1, t_2, ..., t_{n-1} = 0, 0.02, ..., 0.98$
- condition value: $\begin{pmatrix}u(t_1) & u(t_2) & \dots & u(t_n)\end{pmatrix}$
- QOI key: $\begin{pmatrix}\tau_1 & \tau_2 & \dots & \tau_m\end{pmatrix}$, where $\tau_1, \tau_2, ..., \tau_m = 0, 0.02, ..., 0.96$
- QOI value: $\begin{pmatrix}c(\tau_1) & c(\tau_2) & \dots & c(\tau_m)\end{pmatrix}$

Problem 3 & 4 forward and inverse problem of ODE 2

In this problem, the governing dynamic is $u'(t) = a_1 c(t) u(t) + a_2$ over time interval $[0, 1]$, with parameters a_1, a_2, where $a_1 \sim \mathcal{U}(0.5, 1.5)$, $a_2 \sim \mathcal{U}(-1, 1)$. The rest are the same as in Problem 1 and 2.

Problem 5 & 6 forward and inverse problem of ODE 3

In this problem, the governing dynamic is $u'(t) = a_1 u(t) + a_2 c(t) + a_3$ over time interval $[0, 1]$, with parameters a_1, a_2, a_3, where $a_1 \sim \mathcal{U}(-1, 1)$, $a_2 \sim \mathcal{U}(0.5, 1.5)$, $a_3 \sim \mathcal{U}(-1, 1)$. The rest are the same as in Problem 1 and 2.

Problem 7 forward problem of PDE 1

In this problem, the governing PDE is the Poisson equation $u_{xx} = c, u(0) = b_0, u(1) = b_1$ over the spatial interval $[0, 1]$.

- parameters: b_0, b_1
- condition: c
- QOI: $u(x)$

$b_0 \sim \mathcal{U}(-1, 1)$, $b_1 \sim \mathcal{U}(-1, 1)$. $c \sim \mathcal{U}(-2, 2)$.

- condition key: $\begin{pmatrix}x_1 & x_2 & \dots & x_n\end{pmatrix}$, where $x_1, x_2, ..., x_n = 0, 0.02, ..., 0.98$
- condition value: $\begin{pmatrix}c & c & \dots & c\end{pmatrix}$
- QOI key: $\begin{pmatrix}y_1 & y_2 & \dots & y_m\end{pmatrix}$, where $y_1, y_2, ..., x_m = 0, 0.02, ..., 0.98$
- QOI value: $\begin{pmatrix}u(y_1) & u(y_2) & \dots & u(y_m)\end{pmatrix}$

Problem 8 inverse problem of PDE 1

In this problem, the governing PDE and parameter setups are the same as in Problem 1, but the condition and QOI are changed to $u(x)$ and c, respectively.

- condition key: $\begin{pmatrix}x_1 & x_2 & \dots & x_n\end{pmatrix}$, where $x_1, x_2, ..., x_n = 0, 0.02, ..., 0.98$

- condition value: $\begin{pmatrix} u(x_1) & u(x_2) & \dots & u(x_n) \end{pmatrix}$
- QOI key: $\begin{pmatrix} 0 \end{pmatrix}$
- QOI value: $\begin{pmatrix} c \end{pmatrix}$

Problem 9 completion problem of PDE 1

In this problem, the governing PDE and parameter setups are the same as in Problem 1, but the condition and QOI are changed to u at some points $x_1, x_2, \dots, x_n$ and other points $y_1, y_2, \dots, y_m$, respectively.

- condition key: $\begin{pmatrix} x_1 & x_2 & \dots & x_n \end{pmatrix}$, where $x_1, x_2, ..., x_n = 0, 0.01, ..., 0.24, 0.75, 0.76, ..., 0.99$
- condition value: $\begin{pmatrix} u(x_1) & u(x_2) & \dots & u(x_n) \end{pmatrix}$
- QOI key: $\begin{pmatrix} y_1 & y_2 & \dots & y_m \end{pmatrix}$, where $y_1, y_2, ..., y_m = 0.25, 0.26, ..., 0.74$
- QOI value: $\begin{pmatrix} u(y_1) & u(y_2) & \dots & u(y_m) \end{pmatrix}$

Problem 10 & 11 & 12 forward, inverse and completion problems of PDE 2

In this problem, the governing PDE is $-au_{xx} + cu = d, u(0) = b_0, u(1) = b_1$ over spatial interval $[0, 1]$. The parameters are a, d, b_0, b_1, where $a \sim \mathcal{U}(0.5, 1.5), d \sim \mathcal{U}(-2, 2), b_0 \sim \mathcal{U}(-1, 1)$, $b_1 \sim \mathcal{U}(-1, 1)$.

We have constant $c \sim \mathcal{U}(-2, 2)$ as the condition in the forward problem, or QOI in the inverse problem. The rest are the same as in Problem 7, 8 and 9.

Problem 13 time series prediction

we consider the time series prediction problem, where the hidden function is given by $u = A \sin(\omega t + \eta) + c(t)$ over time $[0, 1]$.

- parameters: $c(t)$
- condition: $u(t)$ over $[0, 0.5]$
- QOI: $u(t)$ over $[0.5, 1]$

$c(t) \sim \mathcal{GP}(0, k(x, x'))$ where $k(x, x')$ is defined in Problem 1. $A \sim \mathcal{U}(0.1, 0.2)$, $\omega \sim \mathcal{U}(0.1, 0.2)$, $\eta \sim \mathcal{U}(0, 2\pi)$, respectively.

In the time series prediction problem, each sample of $c(t)$ defines an operator. In each prompt, all demos are generated using the same $c(t)$. However, each demo has its own coefficients A, ω, η. The neural network is expected to learn $c(t), t \in [0, 1]$ from demos, and infer A, ω, η from the question condition, in order the make the prediction.

- condition key: $\begin{pmatrix} t_1 & t_2 & \dots & t_n \end{pmatrix}$, where $t_1, t_2, ..., t_n = 0, 0.01, ..., 0.49$
- condition value: $\begin{pmatrix} u(t_1) & u(t_2) & \dots & u(t_n) \end{pmatrix}$
- QOI key: $\begin{pmatrix} \tau_1 & \tau_2 & \dots & \tau_m \end{pmatrix}$, where $\tau_1, \tau_2, ..., \tau_n = 0.5, 0.51, ..., 0.99$
- QOI value: $\begin{pmatrix} u(\tau_1) & u(\tau_2) & \dots & u(\tau_m) \end{pmatrix}$

3.2 In-Distribution Testing Errors

In this section, we show the testing errors for each of the 13 types of problems, with the distributions for parameters, conditions and QOIs the same as in the training stage, i.e., in-distribution testing errors. By using different random seeds, we ensure that the testing data are different from the training data (although in the same distribution), and that each condition-QOI pair only shows once, either as a demo or a question, during testing.

In Table 2, we show the statistics for the absolute and relative errors. For each type of problem, we conduct 100 in-context learning cases (corresponding to different operators), and for each case, we denote the number of key-value pairs in question QOI as m, which varies for different problem types. The mean and two standard deviations of errors are taken over the total $100m$ values. The relative error is the absolute error divided by the mean of the absolute values of ground truths.

One can see that for all 13 problems, the average relative error is below 2%, mostly below 1%. This shows the capability of a single neural network in learning the operator from demos and predicting the QOI for the question condition, for a diversified type of differential equation problems.

problem	**absolute error**	**relative error**
Problem 1 (forward ODE 1)	0.00434±0.00933	0.00639±0.01374
Problem 2 (inverse ODE 1)	0.00512±0.01106	0.00657±0.01419
Problem 3 (forward ODE 2)	0.00526±0.01889	0.00814±0.02922
Problem 4 (inverse ODE 2)	0.00905±0.02064	0.01215±0.02772
Problem 5 (forward ODE 3)	0.00544±0.01439	0.00728±0.01925
Problem 6 (inverse ODE 3)	0.00666±0.01379	0.00869±0.01800
Problem 7 (forward PDE 1)	0.00152±0.00387	0.00397±0.01009
Problem 8 (inverse PDE 1)	0.00104±0.00173	0.00105±0.00175
Problem 9 (completion PDE 1)	0.00062±0.00104	0.00178±0.00299
Problem 10 (forward PDE 2)	0.00208±0.00599	0.00482±0.01387
Problem 11 (inverse PDE 2)	0.00505±0.01681	0.00488±0.01624
Problem 12 (completion PDE 2)	0.00072±0.00153	0.00184±0.00392
Problem 13 (time series prediction)	0.00368±0.00750	0.00417±0.00850

Table 2: Absolute and relative in-distribution testing errors, for each of 13 types of problems. We show the mean and two standard deviations, separated by ±.

3.3 Out-of-Distribution Testing Error

To show the capability of the neural network in generalizing in-context learning to parameters (or operators) beyond the training distribution, in this section, we study the out-of-distribution testing errors. In particular, we focus on Problems 1 and 2, i.e. the forward and inverse problems of ODE 1 $u'(t) = a_1 c(t) + a_2$.

During training, we have $a_1 \sim \mathcal{U}(0.5, 1.5)$ and $a_2 \sim \mathcal{U}(-1, 1)$. Each (a_1, a_2) pair defines an operator. Now, we extend the distribution of (a_1, a_2) to the

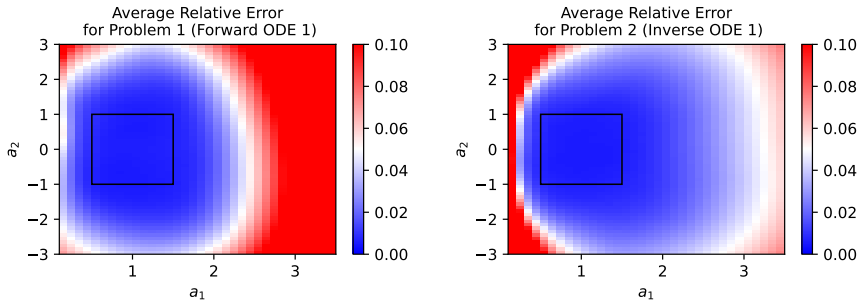

Figure 3: Relative out-of-distribution testing errors for Problem 1 and Problem 2. The black square shows the region of parameters a_1, a_2 used during training.

region of $[0.1, 3.5] \times [-3, 3]$, divide the region into a grid of 0.1 by 0.1 units, randomly sample (a_1, a_2) in each unit, and show the average relative errors in Figure 3.

From the figure, one can see that the neural network can make accurate predictions in the region of (a_1, a_2) far beyond the region used in training, especially for the inverse problem, showing great out-of-distribution generalization.

3.4 Generalization to New Type of Operator

As discussed in [13], one of the advantages of in-context learning over pre-training plus fine-tuning is the ability to mix together multiple skills to solve new tasks. GPT-4 [16] even showed emergent abilities or behaviors beyond human expectations.

Although the scale of our experiment is much smaller than GPT-3 or GPT-4, we also observed primitive evidence of the neural network's ability to learn and apply a new type of operator.

In particular, we designed a new ODE $u'(t) = a_1 u(t) c(t) + b u(t) + a_2$ over time interval $[0, 1]$, by adding a linear term $bu(t)$ to ODE 2, which is borrowed from ODE 3. In the new problem, b is also a parameter, and the operator is determined by (a_1, a_2, b). We study the inverse problem for the new ODE, i.e. $u(t)$ as the condition and $c(t)$ as the QOI. We study the performance of the neural network with $b \in [-1, 1]$. The other setups, including the distribution of a_1, a_2 and $c(t)$, are the same as in Problem 4.

We randomly sample (a_1, a_2) in $[0.5, 1.5] \times [-1, 1]$ (the same as in Problem 4) for a fixed b, and calculate the average relative errors with different b. The results are shown in Figure 3 (the red line). Note that when $b = 0$, the problem is reduced to Problem 4.

To study whether the neural network learned from demos about the new operator, we also test in-context learning with "wrong" demos. The wrong demos are built in the following way: we keep demo QOIs $c(t)$ unchanged, but

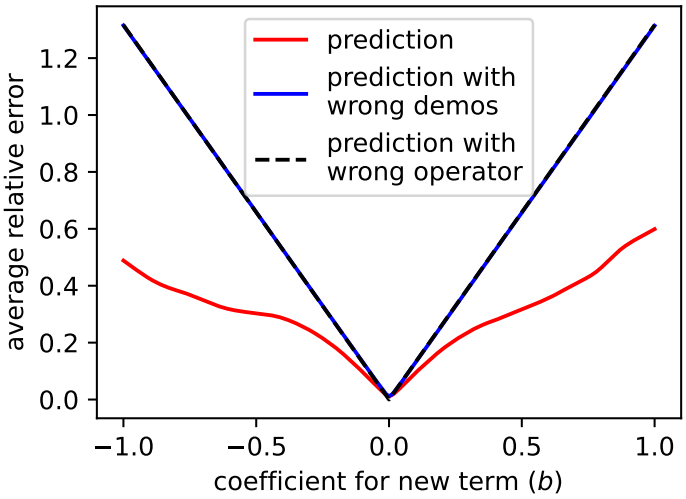

Figure 4: Relative testing errors for the new problem. The red line shows the prediction error w.r.t. b, i.e., the coefficient for the new term. The black dashed line and the blue line shows the errors of the the "wrong operator", or the operator learned from "wrong demos", both associated with the ODE without the new term.

replace demo conditions $u(t)$ with that induced from $u'(t) = a_1 u(t)c(t) + a_2$ (ODE 2) instead of $u'(t) = a_1 u(t)c(t) + bu(t) + a_2$ (the new ODE), using the same initial condition $u(0)$ and parameters a_1, a_2. The results of average relative errors with different b are shown in Figure 3 using the blue line. Note that when $b = 0$, the "wrong" demos become "correct", and thus the blue line and red line overlap.

We also apply the "wrong" operator directly to the question condition. The "wrong" operator is defined as the one that maps from $u(t)$ to $c(t)$ corresponding to ODE 2 $u'(t) = a_1 u(t)c(t) + a_2$ instead of the new ODE. The results of average relative errors with different b are shown in Figure 3 using the black dashed line.

The black dashed line and blue line almost overlap, showing that the neural network can learn the "wrong" operator from "wrong" demos. Interestingly, the red line is significantly below them. This shows that the neural network managed to learn, although not perfectly, some information about the new ODE from "correct" demos.

4 Discussion

One may ask, why even five demos are sufficient to learn the operator? We try to answer this question as follows:

1. We actually only need to learn the operator for a certain distribution of question conditions, not for all possible question conditions.

2. Condition and QOIs for differential equation problems share commonalities that help to learn the operator. For example, for ODE problems, the time difference of u, u, and c satisfy the same equation at each time t, which is exactly the ODE up to some numerical errors. If the neural network captures such shared property during training, and also notices this property in the demos during inference, it only needs to identify the ODE, for which a few demos are sufficient.

3. The operators in this paper are rather simple and limited to a small family, hence easy to identify with a few demos. It is likely that learning more complicated operators requires more demos, as well as a larger neural network with more computation resources.

5 Summary

In this paper, we proposed IN-context Differential Equation Encoder Decoder (INDEED) to learn operators for differential equation problems. Instead of aiming to approximate the solution for a specific problem, or approximate a specific solution operator, in our method, the neural network aims to acquire the ability to "learn an operator from data" and apply the newly learned operator to new problems, during the inference stage without any weight update. Our numerical experiments showed the capability of a single neural network in learning the operator from a few demos and applying it to the question condition, for a diversified type of differential equation problems, including forward and inverse problems of ODEs and PDEs. Our numerical results also showed that the neural network can generalize its ability of in-context learning to parameters (or operators) beyond the training distribution. In the end, we observed primitive evidence of the neural network's ability to learn and apply an unseen type of operator.

The scale of our experiments is rather small. In the future, we wish to scale up the size of the neural network, the types of differential equation problems, the dimensions of keys and values, the length of conditions and QOIs, and the capacity of demo numbers. This requires further development of in-context operator learning, including improvements in neural network architectures and training methods, as well as further theoretical and numerical studies of how in-context operator learning works. In the field of NLP, for example in GPT-4, scaling up leads to emergent abilities or behaviors beyond human expectations [16]. We hope similar emergence appears in a large operator learning network.

Acknowledgement

This work is partially funded by AFOSR MURI FA9550-18-502, ONR N00014-18-1-2527, N00014-18-20-1-2093 and N00014-20-1-2787.

References

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

[2] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.*, 378:686–707, 2019.

[3] Justin Sirignano and Konstantinos Spiliopoulos. DGM: a deep learning algorithm for solving partial differential equations. *J. Comput. Phys.*, 375:1339–1364, 2018.

[4] Weinan E and Bing Yu. The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems. *Commun. Math. Stat.*, 6(1):1–12, 2018.

[5] Yaohua Zang, Gang Bao, Xiaojing Ye, and Haomin Zhou. Weak adversarial networks for high-dimensional partial differential equations. *J. Comput. Phys.*, 411:109409, 14, 2020.

[6] Jiequn Han, Arnulf Jentzen, and Weinan E. Solving high-dimensional partial differential equations using deep learning. *Proc. Natl. Acad. Sci. USA*, 115(34):8505–8510, 2018.

[7] Lars Ruthotto, Stanley J. Osher, Wuchen Li, Levon Nurbekyan, and Samy Wu Fung. A machine learning framework for solving high-dimensional mean field game and mean field control problems. *Proc. Natl. Acad. Sci. USA*, 117(17):9183–9193, 2020.

[8] Alex Tong Lin, Samy Wu Fung, Wuchen Li, Levon Nurbekyan, and Stanley J. Osher. Alternating the population and control neural networks to solve high-dimensional stochastic mean-field games. *Proc. Natl. Acad. Sci. USA*, 118(31):Paper No. e2024713118, 10, 2021.

[9] Zichao Long, Yiping Lu, Xianzhong Ma, and Bin Dong. Pde-net: Learning pdes from data. In *International conference on machine learning*, pages 3208–3216. PMLR, 2018.

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

[11] Zongyi Li, Nikola Borislavov Kovachki, Kamyar Azizzadenesheli, Burigede liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations*, 2021.

[12] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[13] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[14] Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the national academy of sciences*, 113(15):3932–3937, 2016.

[15] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I 16*, pages 213–229. Springer, 2020.

[16] OpenAI. Gpt-4 technical report, 2023.