PNAS

RESEARCH ARTICLE | APPLIED MATHEMATICS

In-context operator learning with data prompts for differential equation problems

Liu Yang[a], Siting Liu[a,1], Tingwei Meng[a,1], and Stanley J. Osher[a,2]

Contributed by Stanley J. Osher; received June 16, 2023; accepted August 12, 2023; reviewed by Weinan E and Andrew M. Stuart

This paper introduces the paradigm of "in-context operator learning" and the corresponding model "In-Context Operator Networks" to simultaneously learn operators from the prompted 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, rather than a solution/operator approximator, 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 examples in the prompt are needed when learning a new operator. Our numerical results show the capability of a single neural network as a few-shot operator learner for a diversified type of differential equation problems, including forward and inverse problems of ordinary differential equations, partial differential equations, and mean-field control problems, and also show that it can generalize its learning capability to operators beyond the training distribution.

operator learning | meta-learning | in-context learning | differential equation | artificial intelligence

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

One typical approach aims to directly approximate the solution given a specific problem. Using deep learning to solve partial differential equations (PDEs) was first introduced in ref. 2 for high-dimensional parabolic equations, and further in ref. 3. The deep Galerkin method (4) imposes constraints on the neural networks to satisfy the prescribed differential equations and boundary conditions. The deep Ritz method (5) utilizes the variational form of PDEs and can be used for solving PDEs that can be transformed into equivalent energy minimization problems. The Physics-Informed Neural Networks (PINNs) (6) propose a deep neural network method for solving both forward and inverse problems by integrating both data and differential equations in the loss function. Weak Adversarial Network (7) leverages the weak form of PDEs by parameterizing the weak solution and the test function as the primal and adversarial networks, respectively. Ref. 8 solves high-dimensional mean-field game problems by encoding both Lagrangian and Eulerian viewpoints in neural network parameterization. APAC-net (9) 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. While transfer learning techniques can be employed to mitigate training costs by fine-tuning pretrained neural networks (10–19), they may not be adequate when there are substantial changes to the target function.

Later, efforts have been made to approximate the solution operator for a differential equation with different parameters or initial/boundary conditions. Early in refs. 20 and 21, shallow neural networks are used to approximate nonlinear operators. In ref. 22, the authors propose solving parametric PDE problems with deep neural networks. Ref. 23 introduces a Bayesian approach using deep convolutional encoder–decoder networks for uncertainty quantification and propagation in problems governed by stochastic PDEs. PDE-Net (24) 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) (25, 26) 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) (27, 28) utilizes the integral kernel in Fourier space to learn the solution operator. In ref. 29, the

Significance

This paper presents In-Context Operator Networks (ICON), a neural network approach that can learn new operators from prompted data during the inference stage without requiring any weight updates. Unlike existing methods that are limited to approximating specific equation solutions or operators and necessitate retraining for new problems, ICON trains a single neural network as an operator learner, eliminating the need for retraining or fine-tuning when encountering different problems. Numerical results demonstrate the efficacy of ICON in solving various types of differential equation problems and generalizing to operators beyond the training distribution. The proposed approach draws inspiration from successful in-context learning techniques used in natural language processing and has implications for artificial general intelligence in physical systems.

Author contributions: L.Y., S.L., T.M., and S.J.O. designed research; L.Y., S.L., and T.M. performed research; L.Y., S.L., and T.M. analyzed data; and L.Y., S.L., T.M., and S.J.O. wrote the paper.

Reviewers: W.E., Peking University; and A.M.S., California Institute of Technology.

The authors declare no competing interest.

Copyright © 2023 the Author(s). Published by PNAS. This article is distributed under Creative Commons Attribution-NonCommercial-NoDerivatives License 4.0 (CC BY-NC-ND).

[1] S.L. and T.M. contributed equally to this work.

[2] To whom correspondence may be addressed. Email: sjo@math.ucla.edu.

This article contains supporting information online at https://www.pnas.org/lookup/suppl/doi:10.1073/pnas.2310142120/-/DCSupplemental.

Published September 19, 2023.

Downloaded from https://www.pnas.org by SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY TECHNICAL UNIVERSITY TOWN on October 26, 2023 from IP address 103.220.9.62.

Downloaded from https://www.pnas.org by SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY TECHNICAL UNIVERSITY TOWN on October 26, 2023 from IP address 103.220.9.62.

authors propose a data-driven framework for approximating input–output maps between infinite-dimensional spaces for parametric PDEs, motivated by neural networks and model reduction. Physics-Informed Neural Operators (30) combines data and PDE constraints at different resolutions to learn solution operators for parametric PDEs. Other related work includes (31–35).

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, at least fine-tuned (30, 33–39), to approximate the new operator.

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 mapping 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 (40) and further scaled up in GPT-3 (41) has demonstrated the capability of large language models as few-shot learners. Here in-context learning refers to a learning paradigm where a generative language model performs a given task specified by the prompted "context", including task descriptions and a few examples related to that task. We refer the readers to ref. 42 for recent progress of in-context learning. Before in-context learning, NLP tasks are mainly dominated by the BERT-style pretraining plus fine-tuning paradigm (43), where a language model is pretrained to generate sentence embeddings and then fine-tuned for specific downstream tasks, typically with additional task-specific layers that map the sentence embeddings to the desired output. In-context learning gets rid of the limitations of the previous BERT-style pretraining plus fine-tuning paradigm 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 switch between or mix together multiple skills.

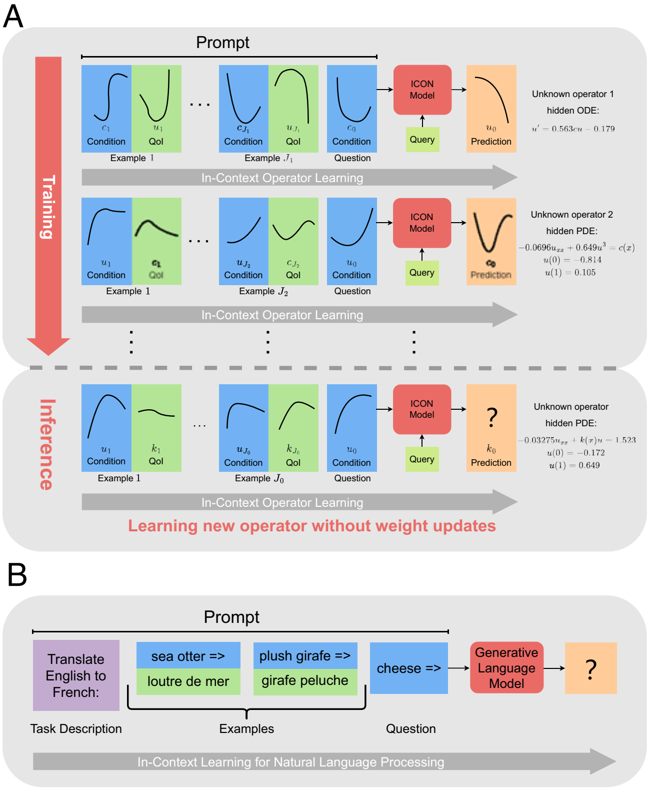

Fig. 1. Illustration and comparison of (*A*) In-Context Operator Networks (ICON) for operator learning and (*B*) in-context learning for NLP.

In this paper, we adapt and extend the idea of in-context learning to learn operators for differential equation problems.

We refer to the inputs of the operators as "conditions," and the operator outputs as "quantities of interest (QoIs)." An "example" consists of one pair of condition and QoI. In the previous paradigm of operator learning, the neural network is trained on examples 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. In this paper, during the inference stage, we instead have the trained neural network taking the examples and a new condition (namely "question condition") as input, and simultaneously completing the following two jobs: 1) learn the operator from examples, 2) apply the learned operator to the question condition and predict the corresponding QoI. We emphasize that there are no weight updates during the inference stage. We name the proposed paradigm as "in-context operator learning," and the corresponding model as "In-Context Operator Networks," or "ICON" in short.

Fig. 1*A* depicts the training and inference process of an ICON model. Parallel to this, Fig. 1*B* portrays an example of in-context learning for NLP, derived from ref. 41.[*] The analogy between in-context learning for NLP and in-context operator learning for scientific machine learning is clear. We embody the spirit of in-context learning by specifying operators in the prompted examples, rather than embedding specific operators into the neural network weights. In fact, the weights merely encapsulate the commonalities shared across operators, as well as the capacity to learn operators dynamically. As a result, in-context

[*] In ref. 41, "cheese =>" was designated as a "prompt." However, more recent literature often uses "prompt" to denote the natural language input provided to language models (44, 45) In alignment with this updated usage, we refer to "cheese =>" as a "question" to ensure clarity.

Downloaded from https://www.pnas.org by SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY TECHNICAL UNIVERSITY TOWN on October 26, 2023 from IP address 103.220.9.62.

operator learning exhibits several superior traits, including 1) the ability to learn new operators without fine-tuning, 2) reduced data requirements for learning a new operator, and 3) strong generalization to operators out of the training distribution.

It is important to note that extra efforts are required to adapt in-context learning to operator learning. A distinguishing feature of in-context operator learning is that the inputs and outputs are continuous functions, as opposed to the discrete tokens used in NLP. To overcome this challenge, we employ a flexible and universal approach by expressing these functions as sets of key-value pairs, where the keys represent function inputs, and the values correspond to the respective function outputs. These key-value pairs are packed into "data prompts," serving as inputs for ICON models, analogous to the way natural language prompts are used in language models. We also adopt a customized transformer encoder–decoder architecture (46), ensuring that 1) the number of examples is flexible, 2) the number and choice of key-value pairs for each condition/QoI function are flexible, 3) the learning process is invariant to the permutation of input key-value pairs, 4) the prediction of question QoI function is not limited to a preset collection of inputs but is applicable to any inputs, and 5) the prediction of question QoI function for different inputs can be performed in parallel. We will discuss the details in *Problem Setup* and *Methodology*.

There have been other works that use generative language models to solve science and mathematics-related tasks. For example, refs. 47 and 48 centers on mathematical reasoning and science question-answering tasks that combine textual, image, and tabular data in prompts. MyCrunchGPT (49) serves as an integration tool for various stages of scientific machine learning, utilizing the capability of ChatGPT to orchestrate the workflow in response to user prompts. The execution of machine learning tasks still necessitates the use of distinct methodologies such as PINNs, DeepONets, etc. In these works, although the tasks are related to science and mathematics, the focus of in-context learning is not on directly performing numerical scientific computations, but primarily applied to language models that carry out NLP tasks. Meta-learning is also used in refs. 50–52, where the task similarities are leveraged to boost the performance in new PDE tasks. However, in these works, it is inevitable that neural networks require fine-tuning when faced with new tasks. In this paper, we attempt to adapt the paradigm of in-context learning for numerical differential equation problems.

The remaining parts of the paper are organized as follows. In the next section, we introduce the problem setup of in-context operator learning. The detailed methodology of ICON is then presented, followed by the experimental results that demonstrate the capability of ICON in learning operators from examples and applying to question conditions during the inference stage. Moreover, we make discussions on several topics that we believe will enhance the reader's comprehension of the proposed method. Finally, we conclude the paper and discuss the limitations and future work.

Problem Setup

In this section, we introduce the problem setup of in-context operator learning.

An operator is defined as a mapping that takes either a single input function or a tuple of input functions and produces an output function. In this paper, we refer to the inputs of the operators as the "conditions," and the operator outputs as the "QoIs."

Take a one-dimensional ODE problem $u'(t) = \alpha u(t) + \beta c(t) + \gamma$ as an example. Given the parameters $\alpha, \beta, \gamma \in \mathbb{R}$, the forward problem learns the solution operator that maps from the control function $c\colon [0, T] \to \mathbb{R}$ and the initial condition $u(0)$, to the solution function $u\colon [0, T] \to \mathbb{R}$. In this case, $c\colon [0, T] \to \mathbb{R}$ and the initial condition $u(0)$ form the condition, and $u\colon [0, T] \to \mathbb{R}$ is the QoI. Note that while $u(0)$ is a number, we can still view it as a function on the domain $\{0\}$ to fit into the framework. Conversely, in the inverse problem, we aim to learn the operator mapping from the solution function $u\colon [0, T] \to \mathbb{R}$ to the control function $c\colon [0, T] \to \mathbb{R}$. In this scenario, the function u is considered as the condition, and the function c is the QoI.

In practical scenarios, it is often challenging to obtain an analytical representation of conditions and QoIs. Instead, we typically rely on observations or data collected from the system. To address this, we utilize a flexible and generalizable approach by representing these entities using key-value pairs, where keys are discrete function inputs, and values are the corresponding outputs of the function. Continuing with the example of the one-dimensional ODE problem introduced above, to represent the function $c\colon [0, T] \to \mathbb{R}$, we consider the discrete time instances as the keys, and the corresponding function values of c as the associated values. We use the key 0 and value $u(0)$ to represent the initial condition of u. It is important to note that the number of key-value pairs is arbitrary, the choice of keys is flexible, and they can vary across different functions.

The training data can be represented as $\{\{(\mathrm{cond}_i^j, \mathrm{QoI}_i^j)\}_{j=1}^{N_i}\}_{i=1}^{M}$, where each i corresponds to a different operator. For a given i, $\{(\mathrm{cond}_i^j, \mathrm{QoI}_i^j)\}_{j=1}^{N_i}$ represents a set of N_i condition-QoI pairs that share the same operator. In our setup, it is important to emphasize that the operators here are completely unknown, even in terms of the corresponding differential equation types. This aspect is aligned with many real-world scenarios where either the parameters of the governing equations are missing or the equations themselves need to be constructed from scratch.

During the inference stage, we are presented with pairs of conditions and QoIs, referred to as "examples," that also share an unknown operator. Additionally, we are given a condition called the "question condition." The objective is to predict the QoI corresponding to the question condition and the unknown operator. Note that the unknown operator in the inference stage may differ from the operators present in the training dataset, potentially even being out of distribution.

Methodology

In this section, we will provide a comprehensive overview of the method. Initially, we will explain the process of constructing neural network inputs, which includes prompts and queries. Subsequently, we will look into the neural network architecture. Furthermore, we will discuss the data preparation and training process. Finally, we will elaborate on the inference process.

Prompt and Query. The model is expected to learn the operator from multiple examples, each consisting of a pair of condition and QoI, and apply it to the question condition, making predictions on the question QoI. As the question QoI is a function, it is also necessary to specify where the model should make evaluations, i.e., the keys for the question QoI, which is referred to as the "queries" (each query is a vector). We group the examples and

Table 1. The matrix representation of the *j*-th example in solving the one-dimensional forward ODE problem (left) and *k*-th example in solving the one-dimensional inverse ODE problem (right)

		condition					quantity of interest (QoI)			
key	term	0	0	…	0	1	0	0	…	0
	time	t_1	t_2	…	t_{n_j-1}	0	τ_1	τ_2	…	τ_{m_j}
	space	0	0	…	0	0	0	0	…	0
value		$c(t_1)$	$c(t_2)$	…	$c(t_{n_j-1})$	$u(0)$	$u(\tau_1)$	$u(\tau_2)$	…	$u(\tau_{m_j})$
index		$\mathbf{e}_j$	$\mathbf{e}_j$	…	$\mathbf{e}_j$	$\mathbf{e}_j$	$-\mathbf{e}_j$	$-\mathbf{e}_j$	…	$-\mathbf{e}_j$

		condition				quantity of interest (QoI)			
key	term	0	0	…	0	0	0	…	0
	time	$\tilde{t}_1$	$\tilde{t}_2$	…	$\tilde{t}_{n_k}$	$\tilde{\tau}_1$	$\tilde{\tau}_2$	…	$\tilde{\tau}_{m_k}$
	space	0	0	…	0	0	0	…	0
value		$u(\tilde{t}_1)$	$u(\tilde{t}_2)$	…	$u(\tilde{t}_{n_k})$	$c(\tilde{\tau}_1)$	$c(\tilde{\tau}_2)$	…	$c(\tilde{\tau}_{m_k})$
index		$\mathbf{e}_k$	$\mathbf{e}_k$	…	$\mathbf{e}_k$	$-\mathbf{e}_k$	$-\mathbf{e}_k$	…	$-\mathbf{e}_k$

In the left table, the condition consists of $c\colon [0,T] \to \mathbb{R}$ and the initial condition $u(0)$; and the QoI is $u\colon [0,T] \to \mathbb{R}$. We use $n_j - 1$ key-value pairs to represent c, one key-value pair for $u(0)$, and m_j key-value pairs for u. Note that in the first row, we use the indicator 0 and 1 to distinguish different terms in the condition, i.e., c and $u(0)$. The third row is populated with zeros since there are no spatial coordinates. $\mathbf{e}_j$ is the column index vector. In the right table, the condition is the function $u\colon [0,T] \to \mathbb{R}$ represented by n_k key-value pairs, the QoI is $c\colon [0,T] \to \mathbb{R}$ represented by m_k key-value pairs. All the values in the first row are 0 because we only have one term u in condition and one term c in QoI. Note that here in each table, the matrices only represent one example. The full prompt is the concatenation of examples and the question condition along the row.

question condition as "prompts," which together with the queries are the neural network inputs. The output of the neural network represents the prediction for the values of the question QoI, corresponding to the input queries.

Although alternative approaches exist, in this paper, we choose a simple method for constructing the prompts, wherein we concatenate the examples and the question condition to create a matrix representation. Each column of the matrix represents a key-value pair. Given that we will be using transformers (46), the order of columns in the prompt will not affect the outcome. Therefore, in order to distinguish the key-value pairs from different conditions and QoIs, we concatenate the key and value in each column with an index column vector. Suppose the maximum capacity of examples is J_m, for simplicity, we use index vector $\mathbf{e}_j$ for the condition in j-th example, and $-\mathbf{e}_j$ for the QoI in j-th example, where $\mathbf{e}_j$ is the one-hot column vector of size $J_m + 1$ with the j-th component to be 1. The index vector for the question condition is $\mathbf{e}_{J_m+1}$. We remark that for a large J_m, a more compact representation, such as the trigonometric position embedding used in NLP tasks, can be applied.

In order to cater to operators with varying numbers of input condition functions, and functions from different spaces, we restructure the keys in prompts and queries. Specifically, we assign the first row of the prompts/queries to indicate different function terms, the second row to denote temporal coordinates, the third row for the first spatial coordinate, and so forth. If certain entries are not required, we will populate them with zeros.

In Table 1, we show the matrix representation of one example used in the one-dimensional forward and inverse ODE problems aforementioned in *Problem Setup*. The prompt is simply the concatenation of examples and the question condition along the row.

In the end, we remark that the number of examples and key-value pairs may differ across various prompts. Transformers are specifically designed to handle inputs of different lengths. However, for the purpose of batching, we still use zero-padding to ensure consistent lengths. Such padding, along with appropriate masks that effectively ignore these zero paddings, has no impact on mathematical calculations.

Neural Network Architecture. We employ a customized transformer encoder–decoder (46) neural network architecture in our method, shown in Fig. 2.

Before entering the encoder, the columns of the prompt undergo an adjustment in their dimensionality facilitated by a shared linear layer and layer normalization (53). The architectural design of the transformer encoder adheres to the model proposed by ref. 46. Specifically, it comprises a stack of identical layers, each having two sublayers: a multihead self-attention mechanism, and a shallow fully connected feed-forward network with GELU activation (54). Each of these sublayers is wrapped by a residual connection (55), followed by layer normalization. The encoder merges information from all examples and the question condition within the prompt, generating an output matrix that represents an embedding of the operator and the question condition.

The decoder also comprises a stack of identical layers, each having a multihead cross-attention mechanism and a shallow fully connected feed-forward network with GELU activation. Similar to the encoder, each sublayer is wrapped by a residual connection, followed by layer normalization. Unlike the model in ref. 46, the self-attention layers are removed from the decode. We will discuss this later. The encoder's output embedding, after layer normalization, is utilized as the key and value inputs for the cross-attention mechanism within the decoder. Along with the embedding, the queries, i.e., keys of question QoI, are also injected into the decoder after a shared linear layer and layer normalization. They repeatedly pass through the cross-attention sublayers (serving as queries) and the feed-forward networks, eventually forming the decoder's output. In the end, an extra

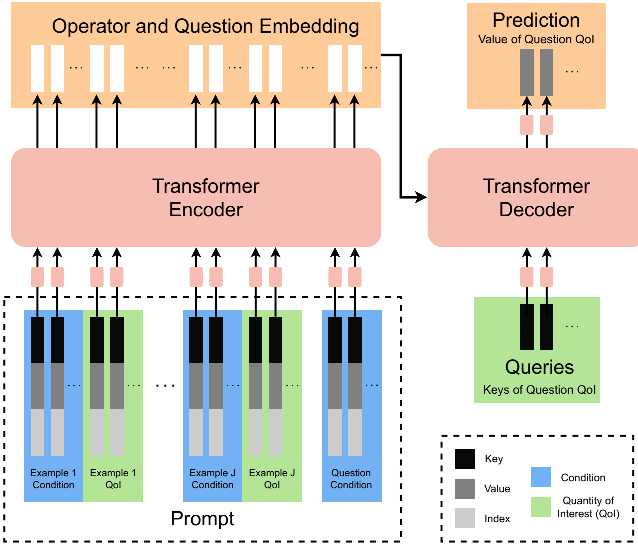

Fig. 2. The neural network architecture for ICON.

Downloaded from https://www.pnas.org by SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY TECHNICAL UNIVERSITY TOWN on October 26, 2023 from IP address 103.220.9.62.

Downloaded from https://www.pnas.org by SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY TECHNICAL UNIVERSITY TOWN on October 26, 2023 from IP address 103.220.9.62.

linear layer is applied to the decoder's output to match its dimensionality to that of the question QoI's value.

The transformer encoder–decoder utilized in this architecture shares similarities with the one used in computer vision for object detection tasks (56). In that case, the decoder takes the image embedding generated by the encoder and "object queries" as inputs, and each output from the decoder is then forwarded to a common feed-forward network that predicts the detection.

The transformer architecture plays a critical role in facilitating the adaptation of in-context operator learning. Its ability to process input sequences of any length and maintain invariance to the sequence permutation, aligns perfectly with the key-value representation for each condition/QoI function. First, it allows for variability in the number of examples. Second, it provides flexibility in the number and choice of key-value pairs for each condition/QoI function. Last, it ensures that rearranging the order of the key-value pairs does not influence the outcomes. Moreover, it is important to note that the self-attention layers are removed in our decoder. Therefore, with a fixed prompt, if we input n query vectors (or n keys of question QoI) into the model and receive n corresponding values as output, each value is exclusively determined by its corresponding query, unaffected by the others. Such independence enables us to design an arbitrary number of queries and make predictions in parallel, wherever we wish to evaluate the question QoI function.

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 parameters do
        Randomly generate $N$ pairs of conditions and QoIs;
        // These $N$ pairs of conditions and QoIs share the same operator
    end
end
```

Data Preparation and Training. Before training the neural network, we prepare data that contain the numerical solutions to different kinds of differential equation problems. The details of data generation are 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. The details of the training process are described in Algorithm 2.

Inference: Few-shot Learning without Weight Updates. After the training, we use the trained neural network to make predictions of the question QoI based on a few examples that describe the operator, as well as the question condition.

During one forward pass, the neural network finishes the following two tasks simultaneously: It learns the operator from the examples, 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 operator learner, and the training stage can be perceived as "learning to learn operators."

Algorithm 2: The training and inference of ICON.

```
// Training stage:
for $i = 1, 2, \ldots,$ training steps do
    for $b = 1, 2, \ldots,$ batch size do
        Randomly select a type of problem and a set of parameters from dataset;
        Randomly set the number of examples $J$, and the number of key-value pairs in each condition and QoI of the examples and question;
        From $N$ pairs of conditions and QoIs, randomly select $J$ pairs as examples and one pair as the question;
        Build a prompt matrix, query vectors, and the ground truth using the selected examples 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 examples and a question condition, and design the queries;
Construct the prompt using the examples and question condition;
Get the prediction of the question QoI using a forward pass of the neural network.
```

Numerical Results

We designed 19 types of problems for training, each of which has 1,000 sets of parameters, so 19 × 1,000 = 19,000 operators in total. For each operator, we generate 100 condition-QoI pairs. In other words, $M = 1{,}000$, $N = 100$ in Algorithm 1 and 2.

We randomly select one to five examples when building the prompt during training. The number of key-value pairs in each condition/QoI randomly ranges from 41 to 50. Hence, the maximum prompt length is 550, consisting of five examples with a cumulative length of 500 and an extra question condition of 50. The neural network used in this paper has about 30 million parameters in total. Other details on configuration and training are in *SI Appendix*.

Problems. We list all 19 types of problems, as well as the setups for parameters and condition-QoI pairs in the data preparation stage (Algorithm 1) in Table 2. As for the implementation of the parameters, we present them in *SI Appendix*.

In-Distribution Operators. In this section, we show the testing errors for each of the 19 types of problems, with the distributions for parameters, conditions, and QoIs the same as in the training stage, i.e., in-distribution operator learning. The number of key-value pairs in each condition/QoI randomly ranges from 41 to 50, as in the training stage. 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 an example or a question, during testing.

We show several in-context operator learning test cases in Figs. 3 and 4.

In Fig. 5, we show relative errors with respect to the number of examples in each prompt for all 19 problems listed in Table 2. For each type of problem, we conduct 500 in-context learning cases, corresponding to 100 different operators, i.e., five cases for each operator. First, the absolute error is computed by averaging the

Table 2. List of differential equation problems

#	Problem description	Differential equations	Parameters	Conditions	QoIs
1	Forward problem of ODE 1	$\frac{d}{dt}u(t) = a_1 c(t) + a_2$	a_1, a_2	$u(0), c(t), t \in [0,1]$	$u(t), t \in [0,1]$
2	Inverse problem of ODE 1	for $t \in [0,1]$		$u(t), t \in [0,1]$	$c(t), t \in [0,1]$
3	Forward problem of ODE 2	$\frac{d}{dt}u(t) = a_1 c(t)u(t) + a_2$	a_1, a_2	$u(0), c(t), t \in [0,1]$	$u(t), t \in [0,1]$
4	Inverse problem of ODE 2	for $t \in [0,1]$		$u(t), t \in [0,1]$	$c(t), t \in [0,1]$
5	Forward problem of ODE 3	$\frac{d}{dt}u(t) = a_1 u(t) + a_2 c(t) + a_3$	a_1, a_2, a_3	$u(0), c(t), t \in [0,1]$	$u(t), t \in [0,1]$
6	Inverse problem of ODE 3	for $t \in [0,1]$		$u(t), t \in [0,1]$	$c(t), t \in [0,1]$
7	Forward damped oscillator	$u(t) = A\sin(\frac{2\pi}{T}t + \eta)e^{-kt}$	k	$u(t), t \in [0, 0.5)$	$u(t), t \in [0.5, 1]$
8	Inverse damped oscillator	for $t \in [0,1]$		$u(t), t \in [0.5, 1]$	$u(t), t \in [0, 0.5)$
9	Forward Poisson equation	$\frac{d^2}{dx^2}u(x) = c(x)$	$u(0), u(1)$	$c(x), x \in [0,1]$	$u(x), x \in [0,1]$
10	Inverse Poisson equation	for $x \in [0,1]$		$u(x), x \in [0,1]$	$c(x), x \in [0,1]$
11	Forward linear reaction–diffusion	$-\lambda a \frac{d^2}{dx^2}u(x) + k(x)u(x) = c$	$u(0), u(1), a, c$	$k(x), x \in [0,1]$	$u(x), x \in [0,1]$
12	Inverse linear reaction–diffusion	for $x \in [0,1]$, $\lambda = 0.05$		$u(x), x \in [0,1]$	$k(x), x \in [0,1]$
13	Forward nonlinear reaction–diffusion	$-\lambda a \frac{d^2}{dx^2}u(x) + ku(x)^3 = c(x)$	$u(0), u(1), k, a$	$c(x), x \in [0,1]$	$u(x), x \in [0,1]$
14	Inverse nonlinear reaction–diffusion	for $x \in [0,1]$, $\lambda = 0.1$		$u(x), x \in [0,1]$	$c(x), x \in [0,1]$
15	MFC g-parameter 1D → 1D	$\inf_{\rho,m} \iint c\frac{m^2}{2\rho}dxdt + \int g(x)\rho(1,x)dx$	$g(x), x \in [0,1]$	$\rho(t=0,x), x \in [0,1]$	$\rho(t=1,x), x \in [0,1]$
16	MFC g-parameter 1D → 2D	s.t. $\partial_t \rho(t,x) + \nabla_x \cdot m(t,x) = \mu\Delta_x \rho(t,x)$		$\rho(t=0,x), x \in [0,1]$	$\rho(t,x), t \in [0.5,1], x \in [0,1]$
17	MFC g-parameter 2D → 2D	for $t \in [0,1], x \in [0,1]$,		$\rho(t,x), t \in [0,0.5), x \in [0,1]$	$\rho(t,x), t \in [0.5,1], x \in [0,1]$
18	MFC ρ_0-parameter 1D → 1D	$c = 20$, $\mu = 0.02$,	$\rho(t=0,x)$	$g(x), x \in [0,1]$	$\rho(t=1,x), x \in [0,1]$
19	MFC ρ_0-parameter 1D → 2D	Periodic spatial boundary condition	$x \in [0,1]$		$\rho(t,x), t \in [0.5,1], x \in [0,1]$

differences between the predicted question QoI values and their corresponding ground truth values across all in-context learning cases. Then, the relative error is obtained by dividing the absolute error by the mean of the absolute values of the ground truth.

Across all 19 problems examined, it is evident from Fig. 5 that the average relative error remains below 6% even in the cases with a single example. The majority of the average relative falls around 2% when using five examples . This underscores the capacity of a single neural network to effectively learn the operator from examples and accurately predict the QoI for various types of differential equation problems. Furthermore, the error consistently decreases as the number of examples in each prompt increases for all 19 problems.

Functions of Superresolution and Subresolution. Even though the neural network is trained using 41 to 50 key-value pairs to represent conditions and QoIs, it demonstrates the ability to generalize to a significantly broader range of numbers without requiring any fine-tuning, including more key-value pairs (superresolution) or less key-value pairs (subresolution). FNO (27, 28) exhibits a similar capability, but in our paper, the generalization is attributed to the adaptability of transformers rather than the use of the integral kernel.

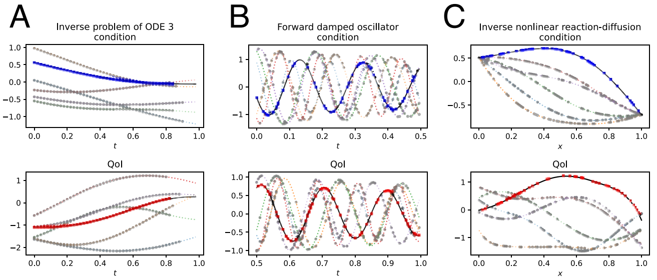

Fig. 3. Visualization of three in-context operator learning test cases for the selected differential equation problems. The problem type is shown in the title. The colored dotted line represents the hidden function of conditions and QoIs in examples, while the gray dots represent the sampled key-value pairs of the example conditions and QoIs used in the prompts. The blue dots represent the key-value pairs in the question conditions, sampled from the hidden function of the question condition in black solid lines. The neural network prediction of the question QoI is illustrated with red dots. One can see the consistency between the prediction and the ground truth (solid black lines). (*A*) inverse problem of ODE 3, (*B*) forward damped oscillator problem, and (*C*) inverse nonlinear reaction-diffusion problem.

In Fig. 6, we examine the neural network on problem 17, i.e., MFC g-parameter 2D → 2D, with the number of (randomly sampled) key-value pairs ranging from 10 to 500 in each condition/QoI. The average relative error is calculated in the same way as for the in-distribution operators, except that we make predictions and evaluate errors in the domain $(t, x) \in [0.5, 1] \times [0, 1]$, by setting the queries as grid points over the temporal-spatial domain. A case of three examples, and 50 key-value pairs is illustrated in Fig. 4.

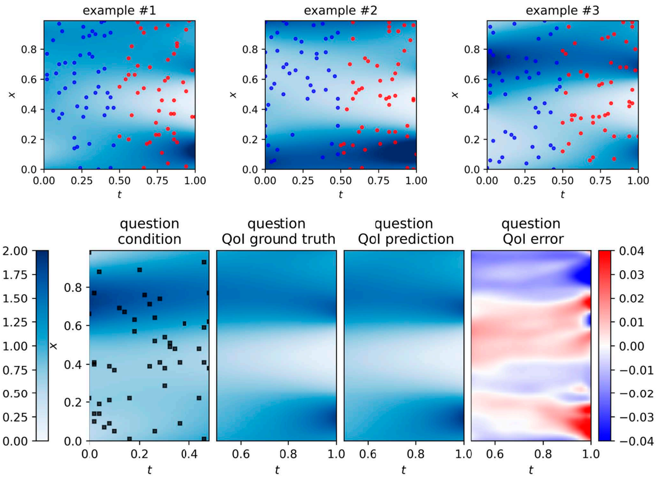

Fig. 4. Visualization of the in-context operator learning test case for problem 17: mean-field control (MFC) g-parameter 2D → 2D. On the *Top*, we show the three examples used when building the prompt, where the blue dots represent the sampled key-value pairs of conditions; the red dot represents the sampled key-value pairs of QoIs. At the *Bottom*, we present the question condition (black dot indicates the condition), ground truth, prediction, and the errors (difference between prediction and ground truth). Note here, we obtain the prediction of the density profile for $t \in [0.5, 1], x \in [0, 1]$ by setting the queries as grid points over the temporal-spatial domain. The examples and question conditions/QoIs share the same color bar.

Downloaded from https://www.pnas.org by SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY TECHNICAL UNIVERSITY TOWN on October 26, 2023 from IP address 103.220.9.62.

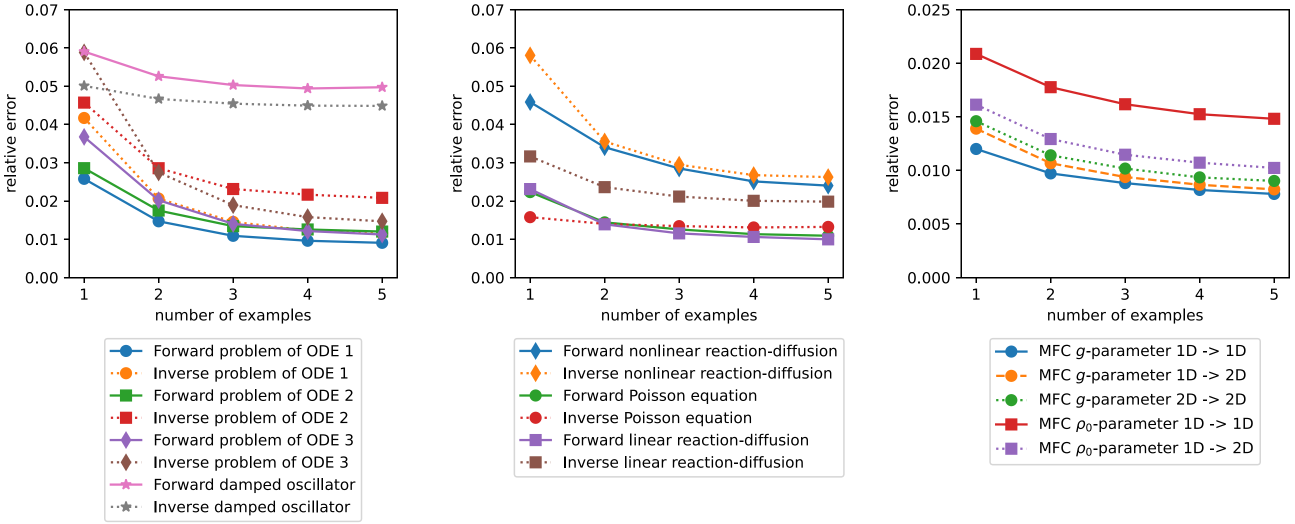

Fig. 5. Average relative in-distribution testing errors for all problems listed in Table 2. The error decreases with the number of examples in each prompt.

With a fixed number of examples in the prompt, the average relative error decreases with an increasing number of key-value pairs in each condition/QoIs, and finally converges below 1%, even for the case of a single example, i.e., one-shot learning.

Out-of-Distribution Operators. In this section, we examine the capability of the neural network in generalizing in-context learning to operators beyond the training distribution. Here, we emphasize that the term "out-of-distribution" does not refer to the conditions, but rather to the operator itself being outside the distribution of operators observed during training.

We conducted tests on four representative problem types, i.e., problems 5, 6, 11, and 12 in Table 2. During the training process of the forward and inverse problems of ODE 3, we randomly generated a_1, a_2, a_3 from uniform distributions $\mathcal{U}(-1, 1)$, $\mathcal{U}(0.5, 1.5)$ and $\mathcal{U}(-1, 1)$, respectively. Each triplet (a_1, a_2, a_3) defines an operator. Now, we expand the distribution to a much larger region. In order to evaluate and provide a visual depiction of the performance, we partitioned the region $[0.1, 3.0] \times [-3, 3]$ into a grid. The performance was then assessed by testing the (a_2, a_3) pair in each grid cell. a_1 continued to be randomly sampled from the distribution $\mathcal{U}(-1, 1)$. Specifically, we conducted 500 in-context learning cases in each cell, corresponding to 100 different operators and five cases with different examples and questions for each operator. Here, the number of examples is fixed as five, and the number of key-value pairs is fixed as the maximum number used in training. We calculate the relative error for each cell and depict the results in Fig. 7 *A* and *B*.

A similar analysis was applied to the forward and inverse problems of linear reaction–diffusion PDE problems. We divided the region of (a, c) into a grid, while keeping the boundary condition parameters $u(0)$ and $u(1)$ randomly sampled from $\mathcal{U}(-1, 1)$. The average relative errors are shown in Fig. 7 *C* and *D*.

It is evident that for all four problems, the neural network demonstrated accurate prediction capabilities even with operator parameters extending beyond the training region. This showcases its strong generalization ability to learn and apply out-of-distribution operators.

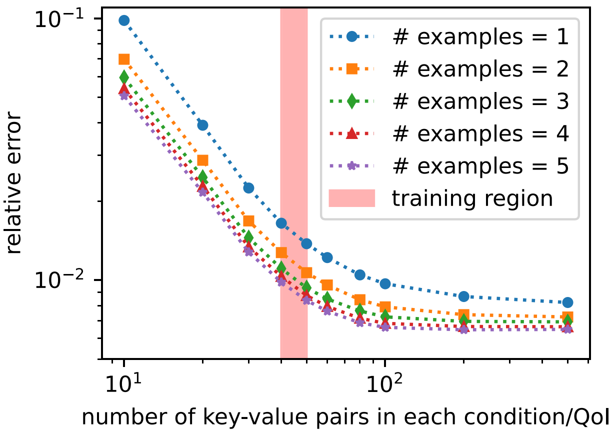

Fig. 6. Average relative testing errors for problem 17, i.e., MFC g-parameter 2D → 2D, with the number of key-value pairs ranging from 10 to 500 in each condition/QoI. As we increase the number of key-value pairs, the error decreases and finally converges below 1%. Note that the neural network is trained using 41 to 50 key-value pairs, represented by the narrow red region in the figure.

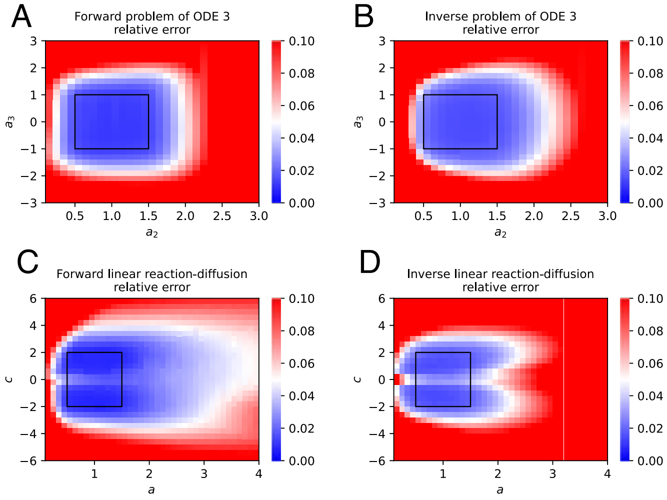

Fig. 7. Average relative error for out-of-distribution operators. The region of operator parameters utilized during training is indicated by a black rectangle. (*A*) forward problem of ODE 3, (*B*) inverse problem of ODE 3, (*C*) forward linear reaction-diffusion problem, and (*D*) inverse linear reaction-diffusion problem.

Downloaded from https://www.pnas.org by SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY TECHNICAL UNIVERSITY TOWN on October 26, 2023 from IP address 103.220.9.62.

Downloaded from https://www.pnas.org by SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY TECHNICAL UNIVERSITY TOWN on October 26, 2023 from IP address 103.220.9.62.

Generalization to Equations of New Forms. As discussed in ref. 41, one of the advantages of in-context learning over pretraining plus fine-tuning is the ability to mix together multiple skills to solve new tasks. GPT-4 (57) 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 preliminary evidence of the neural network's ability to learn and apply operators for equations of new forms that were never seen in training data.

In particular, we designed a new ODE $u'(t) = a_1 u(t)c(t) + bu(t) + a_2$ 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 forward and inverse problems for the new ODE and evaluate the performance of the neural network with $b \in [-0.3, 0.3]$. The other setups, including the distribution of a_1, a_2, and $c(t)$, are the same as in problems 3 and 4 (forward and inverse problem of ODE 2).

To study the influence of scaling up the training dataset, in Fig. 8, we show the average relative errors of neural networks trained with different training datasets. Here, we obtain the average relative error for each b in the same way as we did for each cell for the out-of-distribution operators. To reduce the computational cost, in this section, we train the same neural network as the one analyzed in other sections, but only with half batch size, for 1/5 training steps. We remark that in these new runs, the training datasets have different sizes, but the training steps and batch size are consistent. In other words, the neural network encounters the same number of prompts during training. The expansion of dataset types simply enhances the diversity of prompts.

We first train the neural network only with the datasets involving ODE 2 (both forward and inverse problems). Then, as a reference, we apply the "wrong" operator directly to the question condition. The "wrong" operator is defined as the one corresponding to ODE 2 $u'(t) = a_1 u(t)c(t) + a_2$ instead of the new ODE, with the same a_1 and a_2. Note that when $b = 0$, the new ODE is reduced to ODE 2, thus the error is zero. As another reference, we perform in-context operator learning with the same neural network, but replace the examples in the prompts with the ones corresponding to ODE 2, denoted as "wrong examples." We can see the neural network with "correct examples" performs no better than both references, indicating that the network can hardly generalize its capability of in-context operator learning beyond ODE 2.

We then gradually add more ODE-related datasets to the training data. It is encouraging to see that the error shows a decreasing trend as the training dataset becomes larger. When trained with all ODEs 1, 2, and 3, the neural network performs significantly better than the one merely trained with ODE 2. Such evidence shows the potential of the neural network to learn and apply operators corresponding to previously unseen equation forms, as we scale up the size and diversity of related training data.

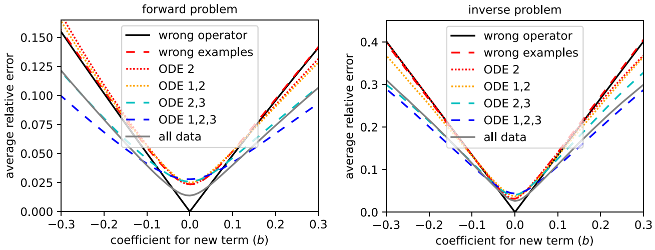

Fig. 8. Average relative errors for the new ODE, with the same neural network trained with different datasets. The error shows a decreasing trend as the training dataset becomes larger.

In the end, we also show the results of the neural network used in other sections, which is trained with the full dataset with a larger batch size for a longer time. The performance on the new ODE is not improved, which is reasonable, since the newly added data on the damped oscillator, PDEs, and MFC problems are not closely related to the new ODE.

Discussion

Why a Very Few Examples are Sufficient to Learn the Operator. We try to answer this question in the following aspects.

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

Second, the training operators and testing operators share commonalities. For example, for ODE problems, u's time derivative, u, and c satisfy the same equation at each time t. If the neural network captures such shared property during training, and also notices this property in the examples during inference, it only needs to identify the ODE, for which a few examples are sufficient.

Last, the operators in this paper are rather simple and limited to a small family, hence easy to identify with a few examples. It is likely that for a larger family of operators in training and testing, in-context operator learning requires more examples (especially for those complicated operators), as well as a larger neural network with more computation resources.

Differentiate Roles of Fine-Tuning. The fine-tuning approach is used in multiple ways in the field of NLP and scientific machine learning. It might be important to differentiate between various roles of fine-tuning.

The BERT-style (43) pretraining followed by the fine-tuning paradigm in NLP is one example. This strategy begins with pretraining a BERT-style neural network on a large-scale corpus to generate sentence embeddings. Following pretraining, the model is fine-tuned on specific downstream tasks, typically with additional task-specific layers that map the sentence embeddings to the desired output. The pretrained model handles a particularly challenging yet common aspect of various NLP tasks—creating a good embedding of the sentence, which significantly simplifies downstream tasks. Nonetheless, the pretrained model does not aim to solve the downstream tasks directly,[†] and each downstream task requires a task-specific version of model.

The recent advancements of in-context learning for generative large language models, such as GPT (41, 57) and LLaMA (58, 59), is a substantial paradigm shift in the realm of NLP. Instead of fine-tuning the pretrained model for individual downstream tasks, in-context learning utilize prompted task descriptions and examples to define tasks. While fine-tuning techniques are available for these generative large language models, their function differs from the traditional BERT-style fine-tuning. In fact, these models can tackle multiple tasks directly without any necessity for fine-tuning. The purpose of fine-tuning these generative large language models is not to enable task-specific adjustments, but mainly to enhance the model's proficiency in particular domains or types of tasks.

For scientific machine learning, the pretraining followed by fine-tuning paradigm is also proposed in the framework of

[†] Indeed, the pretrained BERT model only aims to predict the masked tokens and tell whether two input sentences are next to each other.

Downloaded from https://www.pnas.org by SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY TECHNICAL UNIVERSITY TOWN on October 26, 2023 from IP address 103.220.9.62.

solution approximation or operator approximation. In these cases, the neural network is trained to approximate a particular solution function (10–19) or operator (30, 33–39), and subsequently fine-tuned to approximate a similar one. This approach, however, shares a similar limitation as the BERT-style pretraining plus fine-tuning paradigm, in that the neural network must be fine-tuned individually for each distinct function or operator. The situation further deteriorates when closely comparing scientific machine learning with NLP. In the realm of NLP, creating an efficient sentence embedding simplifies the majority of downstream tasks, if not all. Contrarily, the functions and operators in scientific machine learning are considerably diverse, making it extremely difficult to define a universal "base function" or "base operator" that could be used as a starting point for approximating a broad variety of functions or operators. From this perspective, the task of developing "foundation models" (60), i.e., models trained on extensive data and adaptable to a wide range of downstream tasks, becomes daunting in this approach, even if the neural networks size is scaled up.

The proposed in-context operator learning transfers the paradigm of GPT-style models, instead of BERT-style pretraining plus fine-tuning, to scientific machine learning. As is shown in our experiments, the ICON model can directly learn a broad range of operators without any fine-tuning. However, just like GPT-style models, the ICON model can also be fine-tuned to specialize in a particular set of operators. Looking forward, we envision the development of a large-scale model trained on a substantial dataset under the in-context operator learning paradigm, functioning as a foundation model. This model could be used directly for a wide array of operator learning tasks, or alternatively, it could be fine-tuned to enhance its proficiency in dealing with a specific set of operators.

Application of ICON in Small Scale. With training data in a narrow domain, a small language model below 10 million parameters can produce diverse, fluent, and consistent stories (61). In line with this, our experimental findings show that a small ICON model consisting of roughly 30 million parameters has the capacity to handle relatively straightforward synthetic operators. These findings indicate that for practical applications, a small-scale ICON model would suffice, if the objective is to master a limited range of operators instead of training a general-purpose foundation model.

Summary

In this paper, we proposed the paradigm of "in-context operator learning" and the corresponding model "ICON" to learn operators for differential equation problems. It goes beyond the conventional paradigm of approximating solutions for specific problems or some particular solution operators. Instead, ICON acts as an "operator learner" during inference, i.e., learns an operator from the given examples and applies it to new conditions without any weight updates.

Through our numerical experiments, we demonstrated that a single neural network has the capability to learn an operator from a small number of prompted examples and effectively apply it to the question condition. Such a single neural network, without any retraining or fine-tuning, can handle a diverse set of differential equation problems, including forward and inverse problems of ODEs, PDEs, and mean field control problems. Moreover, while the numbers of key-value pairs for representing the condition/QoI functions are limited to a narrow range during training, ICON can generalize its in-context operator learning ability to a significantly broader range during testing, with errors decreasing and converging as we increase the number of key-value pairs. Furthermore, ICON showed its capacity to learn operators with parameters that extend beyond the training distribution. In the end, our observations provide preliminary evidence of ICON's potential to learn and apply operators corresponding to previously unseen equation forms.

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 example 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 (57). We anticipate the possibility of witnessing such emergence in a large-scale operator learning network.

Data, Materials, and Software Availability. Code for data generation, neural network training, and results analysis has been deposited in https://github.com/LiuYangMage/in-context-operator-networks (62). All other data are included in the manuscript and/or *SI Appendix*.

ACKNOWLEDGMENTS. This work is partially funded by AFOSR MURI FA9550-18-502, ONR N00014-18-1-2527, N00014-20-1-2093 and N00014-20-1-2787. We would like to express our gratitude to ChatGPT for enhancing the wording during the paper-writing phase.

Author affiliations: [a]Department of Mathematics, University of California, Los Angeles, CA 90095

1. G. E. Karniadakis *et al.*, Physics-informed machine learning. *Nat. Rev. Phys.* **3**, 422–440 (2021).
2. W. E, J. Han, A. Jentzen, Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Commun. Math. Stat.* **5**, 349–380 (2017).
3. J. Han, A. Jentzen, W. E, Solving high-dimensional partial differential equations using deep learning. *Proc. Natl. Acad. Sci. U.S.A.* **115**, 8505–8510 (2018).
4. J. Sirignano, K. Spiliopoulos, DGM: A deep learning algorithm for solving partial differential equations. *J. Comput. Phys.* **375**, 1339–1364 (2018).
5. W. E, B. Yu, The deep Ritz method: A deep learning-based numerical algorithm for solving variational problems. *Commun. Math. Stat.* **6**, 1–12 (2018).
6. M. Raissi, P. Perdikaris, 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).
7. Y. Zang, G. Bao, X. Ye, H. Zhou, Weak adversarial networks for high-dimensional partial differential equations. *J. Comput. Phys.* **411**, 109409 (2020).
8. L. Ruthotto, S. J. Osher, W. Li, L. Nurbekyan, S. W. Fung, A machine learning framework for solving high-dimensional mean field game and mean field control problems. *Proc. Natl. Acad. Sci. U.S.A.* **117**, 9183–9193 (2020).
9. A. T. Lin, S. W. Fung, W. Li, L. Nurbekyan, S. J. Osher, Alternating the population and control neural networks to solve high-dimensional stochastic mean-field games. *Proc. Natl. Acad. Sci. U.S.A.* **118**, e2024713118 (2021).
10. S. Goswami, C. Anitescu, S. Chakraborty, T. Rabczuk, Transfer learning enhanced physics informed neural network for phase-field modeling of fracture. *Theor. Appl. Fract. Mech.* **106**, 102447 (2020).
11. X. Chen *et al.*, Transfer learning for deep neural network-based partial differential equations solving. *Adv. Aerodyn.* **3**, 1–14 (2021).
12. E. Haghighat, M. Raissi, A. Moure, H. Gomez, R. Juanes, A physics-informed deep learning framework for inversion and surrogate modeling in solid mechanics. *Comput. Methods Appl. Mech. Eng.* **379**, 113741 (2021).
13. M. Mattheakis, H. Joy, P. Protopapas, Unsupervised reservoir computing for solving ordinary differential equations. arXiv [Preprint] (2021). http://arxiv.org/abs/2108.11417 (Accessed 7 August 2023).
14. S. Chakraborty, Transfer learning based multi-fidelity physics informed deep neural network. *J. Comput. Phys.* **426**, 109942 (2021).
15. S. Desai, M. Mattheakis, H. Joy, P. Protopapas, S. J. Roberts, "One-shot transfer learning of physics-informed neural networks" in *AI for Science Workshop* (2022).

Downloaded from https://www.pnas.org by SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY TECHNICAL UNIVERSITY TOWN on October 26, 2023 from IP address 103.220.9.62.

16. Y. Gao, K. C. Cheung, M. K. Ng, SVD-PINNs: Transfer learning of physics-informed neural networks via singular value decomposition. arXiv [Preprint] (2022). https://doi.org/10.48550/arXiv.2211.08760 (Accessed 7 August 2023).
17. H. Guo, X. Zhuang, P. Chen, N. Alajlan, T. Rabczuk, Analysis of three-dimensional potential problems in non-homogeneous media with physics-informed deep collocation method using material transfer learning and sensitivity analysis. *Eng. Comput.* **38**, 5423–5444 (2022).
18. A. Chakraborty, C. Anitescu, X. Zhuang, T. Rabczuk, Domain adaptation based transfer learning approach for solving PDEs on complex geometries. *Eng. Comput.* **38**, 4569–4588 (2022).
19. C. Xu, B. T. Cao, Y. Yuan, G. Meschke, Transfer learning based physics-informed neural networks for solving inverse problems in engineering structures under different loading scenarios. *Comput. Methods Appl. Mech. Eng.* **405**, 115852 (2023).
20. T. Chen, H. Chen, Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE Trans. Neural Networks* **6**, 911–917 (1995).
21. T. Chen, H. Chen, Approximation capability to functions of several variables, nonlinear functionals, and operators by radial basis function neural networks. *IEEE Trans. Neural Networks* **6**, 904–910 (1995).
22. Y. Khoo, J. Lu, L. Ying, Solving parametric PDE problems with artificial neural networks. *Eur. J. Appl. Math.* **32**, 421–435 (2021).
23. Y. Zhu, N. Zabaras, Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification. *J. Comput. Phys.* **366**, 415–447 (2018).
24. Z. Long, Y. Lu, X. Ma, B. Dong, "PDE-net: Learning PDEs from data" in *International Conference on Machine Learning* (PMLR, 2018), pp. 3208–3216.
25. L. Lu, P. Jin, G. Pang, Z. Zhang, G. E. Karniadakis, Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nat. Mach. Intell.* **3**, 218–229 (2021).
26. S. Wang, H. Wang, P. Perdikaris, Learning the solution operator of parametric partial differential equations with physics-informed DeepONets. *Sci. Adv.* **7**, eabi8605 (2021).
27. Z. Li *et al*., "Fourier neural operator for parametric partial differential equations" in *International Conference on Learning Representations* (2021).
28. N. Kovachki *et al*., Neural operator: Learning maps between function spaces with applications to PDEs. *J. Mach. Learn. Res.* **24**, 1–97 (2023).
29. K. Bhattacharya, B. Hosseini, N. B. Kovachki, A. M. Stuart, Model reduction and neural networks for parametric PDEs. *SMAI J. Comput. Math.* **7**, 121–157 (2021).
30. Z. Li *et al*., Physics-informed neural operator for learning partial differential equations. arXiv [Preprint] (2021). http://arxiv.org/abs/2111.03794 (Accessed 7 August 2023).
31. D. Kochkov *et al*., Machine learning-accelerated computational fluid dynamics. *Proc. Natl. Acad. Sci. U.S.A.* **118**, e2101784118 (2021).
32. G. Kissas *et al*., Learning operators with coupled attention. *J. Mach. Learn. Res.* **23**, 1–63 (2022).
33. S. Goswami, K. Kontolati, M. D. Shields, G. E. Karniadakis, Deep transfer operator learning for partial differential equations under conditional shift. *Nat. Mach. Intell.* **4**, 1155–1164 (2022).
34. M. Zhu, H. Zhang, A. Jiao, G. E. Karniadakis, L. Lu, Reliable extrapolation of deep neural operators informed by physics or sparse observations. *Comput. Methods Appl. Mech. Eng.* **412**, 116064 (2023).
35. A. Subel, Y. Guan, A. Chattopadhyay, P. Hassanzadeh, Explaining the physics of transfer learning in data-driven turbulence modeling. *PNAS Nexus* **2**, pgad015 (2023).
36. H. Wang, R. Planas, A. Chandramowlishwaran, R. Bostanabad, Mosaic flows: A transferable deep learning framework for solving PDEs on unseen domains. *Comput. Methods Appl. Mech. Eng.* **389**, 114424 (2022).
37. W. Xu, Y. Lu, L. Wang, "Transfer learning enhanced DeepONet for long-time prediction of evolution equations" in *Proceedings of the AAAI Conference on Artificial Intelligence* (2023), vol. 37, pp. 10629–10636.
38. Y. Lyu, X. Zhao, Z. Gong, X. Kang, W. Yao, Multi-fidelity prediction of fluid flow and temperature field based on transfer learning using Fourier Neural Operator. arXiv [Preprint] (2023). http://arxiv.org/abs/2304.06972 (Accessed 7 August 2023).
39. S. Subramanian *et al*., Towards foundation models for scientific machine learning: Characterizing scaling and transfer behavior. arXiv [Preprint] (2023). http://arxiv.org/abs/2306.00258 (Accessed 7 August 2023).
40. A. Radford *et al*., Language models are unsupervised multitask learners. *OpenAI Blog* **1**, 9 (2019).
41. T. Brown *et al*., Language models are few-shot learners. *Adv. Neural Inf. Process. Syst.* **33**, 1877–1901 (2020).
42. Q. Dong *et al*., A survey for in-context learning. arXiv [Preprint] (2022). http://arxiv.org/abs/2301.00234 (Accessed 7 August 2023).
43. J. Devlin, M. W. Chang, K. Lee, K. Toutanova, BERT: Pre-training of deep bidirectional transformers for language understanding. arXiv [Preprint] (2018). http://arxiv.org/abs/1810.04805 (Accessed 7 August 2023).
44. J. Wei *et al*., Chain-of-thought prompting elicits reasoning in large language models. *Adv. Neural Inf. Process. Syst.* **35**, 24824–24837 (2022).
45. P. Liu *et al*., Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Comput. Surv.* **55**, 1–35 (2023).
46. A. Vaswani *et al*., Attention is all you need. *Adv. Neural Inf. Process. Syst.* **30**, 6000–6010 (2017).
47. P. Lu *et al*., Dynamic prompt learning via policy gradient for semi-structured mathematical reasoning. arXiv [Preprint] (2022). http://arxiv.org/abs/2209.14610 (Accessed 7 August 2023).
48. P. Lu *et al*., Learn to explain: Multimodal reasoning via thought chains for science question answering. *Adv. Neural Inf. Process. Syst.* **35**, 2507–2521 (2022).
49. V. Kumar, L. Gleyzer, A. Kahana, K. Shukla, G. E. Karniadakis, CrunchGPT: A chatGPT assisted framework for scientific machine learning. arXiv [Preprint] (2023). http://arxiv.org/abs/2306.15551 (Accessed 7 August 2023).
50. Y. Chen, B. Dong, J. Xu, Meta-MgNet: Meta multigrid networks for solving parameterized partial differential equations. *J. Comput. Phys.* **455**, 110996 (2022).
51. X. Meng, L. Yang, Z. Mao, J. del Águila Ferrandis, G. E. Karniadakis, Learning functional priors and posteriors from data and physics. *J. Comput. Phys.* **457**, 111073 (2022).
52. X. Huang *et al*., Meta-auto-decoder for solving parametric partial differential equations. *Adv. Neural Inf. Process. Syst.* **35**, 23426–23438 (2022).
53. J. L. Ba, J. R. Kiros, G. E. Hinton, Layer normalization. arXiv [Preprint] (2016). http://arxiv.org/abs/1607.06450 (Accessed 7 August 2023).
54. D. Hendrycks, K. Gimpel, Gaussian error linear units (GELUS). arXiv [Preprint] (2016). http://arxiv.org/abs/1606.08415 (Accessed 7 August 2023).
55. K. He, X. Zhang, S. Ren, J. Sun, "Deep residual learning for image recognition" in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 770–778.
56. N. Carion *et al*., "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* (Springer, 2020), pp. 213–229.
57. OpenAI, GPT-4 technical report (2023).
58. H. Touvron *et al*., LLaMA: Open and efficient foundation language models. arXiv [Preprint] (2023). http://arxiv.org/abs/2302.13971 (Accessed 7 August 2023).
59. H. Touvron *et al*., Llama 2: Open foundation and fine-tuned chat models. arXiv [Preprint] (2023). http://arxiv.org/abs/2307.09288 (Accessed 7 August 2023).
60. R. Bommasani *et al*., On the opportunities and risks of foundation models. arXiv [Preprint] (2021). http://arxiv.org/abs/2108.07258 (Accessed 7 August 2023).
61. R. Eldan, Y. Li, TinyStories: How small can language models be and still speak coherent English? arXiv [Preprint] (2023). http://arxiv.org/abs/2305.07759 (Accessed 7 August 2023).
62. L. Yang, S. Liu, T. Meng, Code for in-context operator learning with data prompts for differential equation problems. GitHub. https://github.com/LiuYangMage/in-context-operator-networks. Deposited 7 August 2023.