Accelerating PDE Data Generation via Differential Operator Action in Solution Space

Huanshuo Dong [1] **Hong Wang** [1] **Haoyang Liu** [1] **Jian Luo** [1] **Jie Wang** [1*]

Abstract

Recent advancements in data-driven approaches, such as Neural Operator (NO), have demonstrated their effectiveness in reducing the solving time of Partial Differential Equations (PDEs). However, one major challenge faced by these approaches is the requirement for a large amount of high-precision training data, which needs significant computational costs during the generation process. To address this challenge, we propose a novel PDE dataset generation algorithm, namely **Diff**erential **O**perator **A**ction in **S**olution space (**DiffOAS**), which speeds up the data generation process and enhances the precision of the generated data simultaneously. Specifically, DiffOAS obtains a few basic PDE solutions and then combines them to get solutions. It applies differential operators on these solutions, a process we call 'operator action', to efficiently generate precise PDE data points. Theoretical analysis shows that the time complexity of DiffOAS method is one order lower than the existing generation method. Experimental results show that DiffOAS accelerates the generation of large-scale datasets with 10,000 instances by 300 times. Even with just 5% of the generation time, NO trained on the data generated by DiffOAS exhibits comparable performance to that using the existing generation method, which highlights the efficiency of DiffOAS.

1. Introduction

The Partial Differential Equation (PDE) is a fundamental mathematical model derived from various scientific areas including physics, chemistry, biology, engineering and so on (Zachmanoglou & Thoe, 1986).

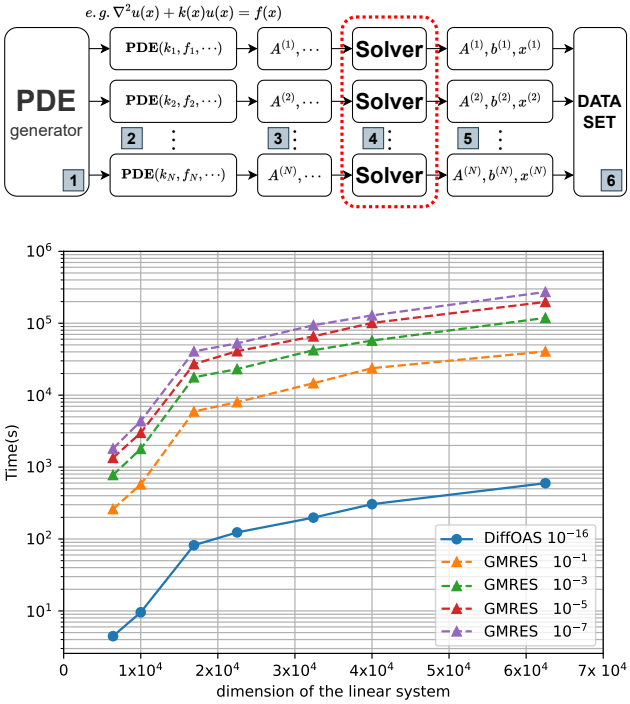

Figure 1. **Above.** Generation process of the PDE dataset. 1. Produce collection of random parameters derived from PDE. 2. Generate the relevant PDE using these parameters 3. Convert the PDE into linear equation systems using discretization methods. 4. Solve linear equations based on input parameters 5. Acquire solutions for the linear systems and translate them into solutions for the PDEs. 6. Compile the data into a dataset. **Below.** The generation cost of DiffOAS and GMRES varies with the dimension of the linear system. GMRES curves in the graph represent different truncation errors, where GMRES 10^{-5} indicates the algorithm's time cost with a truncation error of 10^{-5}. In contrast, our DiffOAS maintains machine precision of 10^{-16}. It can be observed that DiffOAS significantly speeds up the data generation process, achieving a speedup of up to $70,000$ times.

Traditionally, solving PDEs often relies on extensive domain expertise and computationally intensive methods. To reduce the solving time for PDEs, recent research has explored data-driven approaches to predict PDE solutions. One such approach is the use of Neural Operators (NOs) (Lu et al.,

[1]CAS Key Laboratory of Technology in GIPAS & MoE Key Laboratory of Brain-inspired Intelligent Perception and Cognition, University of Science and Technology of China. Correspondence to: Jie Wang <jiewangx@ustc.edu.cn>.

Proceedings of the 41^{st} International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

2019), which has achieved promising results in accelerating the solving process (Zhang et al., 2023).

However, the long running time and computation costs of generating the training datasets pose a great challenge to the training of NOs. Firstly, real-world applications often involve various types of PDEs, and the training of NO for a certain type of PDEs requires a large number of training instances. For example, when training a Fourier Neural Operator (FNO) (Li et al., 2020) for Darcy flow problems, it is common to require thousands of PDEs and their corresponding solutions under diverse initial conditions. Secondly, acquiring the corresponding solution functions of PDEs as labels for training the NO network poses a further obstacle. Existing algorithms typically rely on traditional PDE solvers, such as the finite difference method (FDM) (Strikwerda, 2004) and finite element method (FEM) (LeVeque, 2002). These algorithms, demonstrated in Figure 1, involve solving large-scale linear equation systems in the 4-th step *Solver*, with a high computational complexity of $O(n^3)$, where n represents the dimension of the linear system. Consequently, utilizing such algorithms can be time-consuming, with the computational cost of the *Solver* module often accounting for over 95% of the entire data generation process (Hughes, 2012). Thirdly, solving large-scale linear systems often involves the use of iterative methods. However, due to the presence of termination conditions, these methods inevitably introduce errors, which can potentially result in a degradation of the performance of the NO network. Furthermore, as shown in Figure 1, increasing the accuracy of solving linear systems leads to increased time costs. Therefore, these challenges of data generation have significantly hindered the real-world applications of NOs (Zhang et al., 2023).

To address the aforementioned challenges, we propose a novel and efficient PDE data generation method, named **Diff**erential **O**perator **A**ction in **S**olution space (**DiffOAS**). DiffOAS has two key advantages: it accelerates the data generation process and enhances the precision of the generated data simultaneously. DiffOAS replaces the process of solving linear systems in the *Solver* module in Figure 1 with operator actions. Initially, we generate a set of PDE solution functions that comply with the actual physical contexts, which serve as basis functions for the solution space. These basis functions are then appropriately combined to satisfy the PDE conditions and generate solution functions. The discretized operator and solution functions are used as inputs in the *Solver*, where the corresponding differential operators are applied to calculate the remaining data that satisfies PDE constraints. DiffOAS utilizes operator actions to avoid the process of solving linear equation systems, reducing the computational complexity by one order. As shown in Figure 1, the DiffOAS method can significantly accelerate PDE data generation, achieving a speedup of up to 300 times.

The distinct contributions of our work can be summarized as follows.

- We introduce a novel PDE dataset generation algorithm that utilizes differential operator actions. This algorithm generates a set of PDE solution functions that align with the physical background. By applying differential operators to combinations of these solution functions, it can generate large-scale PDE data.
- We have demonstrated in our theoretical analysis that our proposed algorithm is able to achieve mechanical precision at a low cost compared to existing generation methods, ensuring the accuracy of generated data
- We demonstrate that our proposed algorithm significantly reduces the computational complexity and data generation time, which addresses the long-standing challenge of the data-driven approaches for solving PDEs. Even with just 5% of the generation time, NO trained on the data generated by DiffOAS exhibits comparable performance to that using existing generation methods.

2. Related Work

2.1. Data-Efficient and learning-based PDE Algorithms

Data-efficient and learning-based algorithms have significantly impacted the realm of PDEs. Major advancements include the development of NOs such as the Fourier Neural Operator (FNO) (Li et al., 2020) and the Deep Operator Network (DeepONet) (Lu et al., 2019), both of which have considerably advanced the solving of PDEs. These models harness deep learning to unravel the complexities inherent in PDE systems, providing efficient alternatives to conventional methods. Additionally, there are also studies exploring the use of neural networks to accelerate the solution of linear equation systems, thereby speeding up the process of solving PDEs. The evolution of data-driven solvers, exemplified by the data-optimized iterative schemes in studies such as Hsieh et al. (2019); Yang et al. (2016); Li et al. (2023); Kaneda et al. (2023), highlights a trend towards merging machine learning with traditional computational techniques to enhance algorithmic efficiency.

Furthermore, machine learning has achieved notable advancements in improving matrix preconditioning for PDE solving. (Greenfeld et al., 2019; Luz et al., 2020; Taghibakhshi et al., 2021) demonstrate the effectiveness of neural networks in refining multigrid preconditioning algorithms, thus streamlining the computational process. (Götz & Anzt, 2018) utilized Convolutional Neural Networks (CNNs) for the optimization of block Jacobi preconditioning algorithms, while (Stanaityte, 2020) developed corresponding Incomplete Lower-Upper Decomposition (ILU)

preconditioning algorithms leveraging machine learning insights.

2.2. Data Generation for PDE Algorithms

Training data-driven PDE algorithms requires large offline paired parametrized PDE datasets. Typically, the generation of PDE datasets is obtained by solving PDE problems employing traditional computational mathematics algorithms. In the field of computational mathematics, the numerical solution of complex PDEs generally involves converting the intricate equations into solvable linear systems using various discretization methods (Morton & Mayers, 2005), such as Finite Difference Method (FDM) (Strikwerda, 2004), Finite Element Method (FEM) (Hughes, 2012; Johnson, 2012), and Finite Volume Method (FVM) (LeVeque, 2002). These approaches ultimately result in the formation of large linear equation systems, which are usually solved using iterative methods (Liesen & Strakos, 2013) suited to the matrix properties (Golub & Van Loan, 2013), such as the Conjugate Gradient (CG) algorithm for SPD matrices (Hestenes et al., 1952), the Minimum Residual (MINRES) Method for symmetric matrices (Paige & Saunders, 1975), and the Generalized Minimum Residual (GMRES) Method for non-symmetric matrices (Saad & Schultz, 1986).

Although these traditional methods are effective, they are not exclusively designed for dataset generation, and using them independently for this purpose results in substantial computational costs. This has emerged as a significant barrier to the further advancement of data-driven approaches (Zhang et al., 2023; Hao et al., 2022). In tackling this challenge, research has led to the development of architectures that preserve symmetries and conservation laws (Brandstetter et al., 2022; Liu et al., 2023; Wang et al., 2024), enhancing model generalization and data efficiency. However, these advancements mainly focus on the optimization of the PDE solving algorithms themselves, without significantly altering data generation methods.

3. Preliminaries

3.1. Discretization for PDEs

Our main focus is the generation of PDE datasets, which are obtained by solving relevant PDE problems. Due to the complexity, continuity, and detailed boundary conditions of these PDE problems, discretized numerical methods such as FDM, FEM, and FVM are typically used to solve them (Strikwerda, 2004; Hughes, 2012; Johnson, 2012; LeVeque, 2002; Cheng & Xu, 2023).

Numerical methods discretize a partial differential equation problem by mapping it from an infinite-dimensional function space to a finite-dimensional space, resulting in a system of linear equations. To illustrate, we consider solving a two-dimensional inhomogeneous Helmholtz equation using the FDM, which transforms it into a linear equation system:

$$\nabla^2 u(x,y) + ku(x,y) = f(x,y), \tag{1}$$

using a 2×2 internal grid (i.e., $N_x = N_y = 2$ and $\Delta x = \Delta y$), the unknowns $u_{i,j}$ can be arranged in row-major order as follows: $u_{1,1}, u_{1,2}, u_{2,1}, u_{2,2}$. For central differencing on a 2×2 grid, the vector $\boldsymbol{b}$ will contain the values of $f_{i,j} = f(x_i, y_j)$ and the linear equation system $\boldsymbol{Ax} = \boldsymbol{b}$ can be expressed as:

$$\begin{bmatrix} -4+k & 1 & 1 & 0 \\ 1 & -4+k & 0 & 1 \\ 1 & 0 & -4+k & 1 \\ 0 & 1 & 1 & -4+k \end{bmatrix} \begin{bmatrix} u_{1,1} \\ u_{1,2} \\ u_{2,1} \\ u_{2,2} \end{bmatrix} = \begin{bmatrix} f_{1,1} \\ f_{1,2} \\ f_{2,1} \\ f_{2,2} \end{bmatrix}.$$

By employing various methods to generate function f and constant k, such as utilizing Gaussian Random Fields (GRF) or Uniform Random Distribution, we can derive inhomogeneous Helmholtz equations characterized by distinct parameters.

This represents a relatively simple PDE problem. However, for realistic physical simulations and more complex boundary conditions, numerically solving PDEs demands more specialized discretization methods and denser grid divisions. This results in a substantial increase in the size of the matrices within the linear systems, with matrix dimensions potentially growing from 10^3 to 10^6 or even larger. This results in significant computational costs during the dataset generation process.

3.2. Details of the Dataset

To introduce the content of the dataset, we will use a 2D Darcy Flow problem as an example (Li et al., 2020):

$$\begin{aligned} \nabla \cdot (a(x,y)\nabla u(x,y)) &= f(x,y) \quad && (x,y) \in D \\ u(x,y) &= 0 && (x,y) \in \partial D, \end{aligned}$$

where $D = [0,1]^2$. The functions $a(x,y)$, $f(x,y)$, and $u(x,y)$ represent parameter functions, solution functions, and right-hand side term functions. They are discretized on a $N \times N$ uniform grid $\Omega = \{(i/N, j/N), i,j = 0,1,\cdots,N\}$. So, the dimension of the matrix $\boldsymbol{A}$ obtained from the discrete PDE is given by $n = N \times N$. According to the given grid Ω, generate a dataset with features $F_k = (a_k(\Omega), f_k(\Omega))$ and target $T_k = (u_k(\Omega))$, where $k = 1, 2, ..., N_{samples}$. In existing data generation methods, the solution function $u(x,y)$ is obtained by solving the equation using $a(x,y)$ and $f(x,y)$, which can be constants, generated through GRFs, or other generation methods.

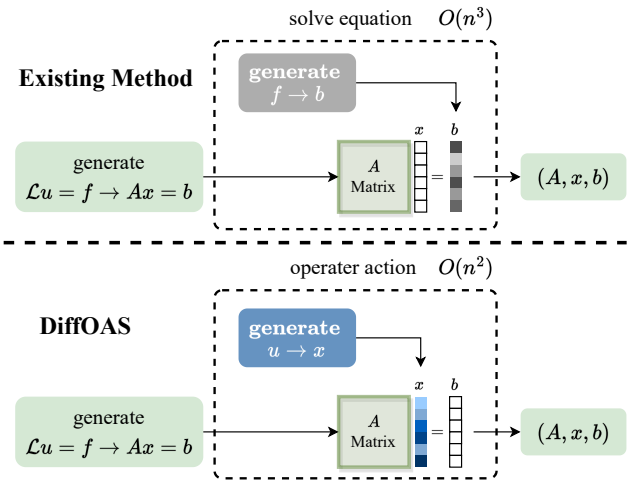

Figure 2. Overview of the model architecture. The process of DiffOAS and existing method: Firstly, Transforming a PDE into a linear equation system. Secondly, (**Above**) **Existing Method** generates f, input its discretized form $\boldsymbol{b}$ into the linear equation system, and solve $\boldsymbol{x}$ accordingly. (**Below**) **DiffOAS** generates u using basis functions, inputs its discretized form $\boldsymbol{x}$ into the linear equation system, and calculates $\boldsymbol{b}$ accordingly. Finally organizing the inputs and outputs of the linear equation system to create a complete PDE dataset.

4. Method

In this paper, we explore the most prevalent form of PDEs, namely the steady-state PDEs constrained by boundary conditions (Evans, 2022). These can generally be expressed in the following form:

$$\begin{aligned} \mathcal{L}(a(\boldsymbol{x}))u(\boldsymbol{x}) &= f(\boldsymbol{x}) \quad \boldsymbol{x} \in D \\ \mathcal{B}(b(\boldsymbol{x}))u(\boldsymbol{x}) &= g(\boldsymbol{x}) \quad \boldsymbol{x} \in \partial D. \end{aligned} \tag{2}$$

Here, $D \subseteq \mathbb{C}^n$ is the PDE's domain, with ∂D as its boundary, and $\boldsymbol{x} \in \mathbb{C}^n$. The operator $\mathcal{L}$, governed by the parameter function $a(\boldsymbol{x})$, is a partial differential operator. The solution function is $u(\boldsymbol{x})$, and $f(\boldsymbol{x})$ represents the source or forcing function on the right-hand side. The boundary operator $\mathcal{B}$, controlled by $b(\boldsymbol{x})$, dictates the equation's boundary conditions on ∂D, and $g(\boldsymbol{x})$ is the corresponding right-hand side function on ∂D.

In existing algorithms, the dataset generation process began with the random generation of other parameters for PDEs, followed by employing traditional PDE solvers to determine the corresponding solution functions $u(\boldsymbol{x})$. This approach necessitated the *Solver* module in Figure 1 to solve large linear systems of equations involving substantial matrices, as mentioned in the Preliminaries 3.1. As mentioned in Figure 1, existing method requires a substantial amount of time and often becomes a bottleneck in the dataset generation process.

As shown in Figure 2, unlike traditional methods, DiffOAS method initiates by generating solution functions $u(\boldsymbol{x})$. Subsequently, these functions are subjected to operator actions to derive other PDE parameters, such as $f(\boldsymbol{x})$, thereby facilitating dataset creation. Both methods generate data that adhere to their respective PDE constraints. However, our algorithm circumvents the significant computational expense and termination error associated with solving large matrix linear systems. This strategy substantially reduces computational overhead and enhances precision.

DiffOAS method comprises two primary steps:

1. **Solution Functions Generation**: This phase involves the low-cost generation of solution functions that comply with PDE conditions.

2. **Operator Action**: In this stage, operators are applied to the generated solution functions to obtain data that satisfies PDE constraints.

4.1. Solution Functions Generation

DiffOAS method specifically generates a series of basis functions either through a designated distribution that satisfies actual physical contexts (typically choosing between 10 to 50, taking N_{basis} as an example). This paper utilizes **existing method** to generate these basis functions. These basis functions form the foundational elements in the solution space for that particular physical distribution. We randomly weight and normalize these basis functions using Gaussian distribution. Additionally, we introduce a noise element ϵ that maintains the boundary conditions unaltered (further details are provided in the Appendix B.2), resulting in a novel solution function $u_{new}(\boldsymbol{x})$.

$$\begin{aligned} u_{\text{new}} &= \sum_{i=1}^{N_{basis}} \alpha_i u_i + \epsilon \qquad & \alpha_i &= \frac{\mu_i}{\sum_{j=1}^{N_{basis}} \mu_j} \\ & \mu_i \sim N(0,1) & & i = 1, 2, ..., N_{basis}. \end{aligned}$$

This method of weighting ensures that the newly formulated solution function $u_{new}(\boldsymbol{x})$ complies with the boundary conditions of the PDE, while also preserving its physical relevance. The incorporation of noise serves to enhance the complexity and applicability of the generated dataset. Through this methodology, we are capable of producing a myriad of physically meaningful solution functions at minimal expense, utilizing a set of basis functions that align with a physical distribution.

4.2. Operator Action

The operator $\mathcal{L}$ is a functional mapping within a Sobolev space associated with PDEs. This mapping transforms functions to functions as detailed below:

$$\begin{aligned} \mathcal{L} &: \mathcal{U} \to \mathcal{F} \\ &: u \mapsto f, \end{aligned} \tag{3}$$

where $\mathcal{U}$ and $\mathcal{F}$ represent the Sobolev spaces for the solution functions and the right-hand side functions of the PDE, respectively. Here, u is a solution function in space $\mathcal{U}$, and f is the corresponding right-hand side function in space $\mathcal{F}$ for the PDE problem (2) (Evans, 2022).

Specifically, as mentioned in the Preliminaries 3.1 regarding the Helmholtz equation (1), the Sobolev function spaces are defined as $\mathcal{U} = \mathcal{F} = \mathcal{W}^{1,2}([0,1]^2) = \mathcal{H}^1([0,1]^2)$, where the norm for $\mathcal{H}^1([0,1]^2)$ is given by

$$\|f\|_{\mathcal{H}^1} = \left(\int_{[0,1]^2} |f|^2 + |\nabla f|^2 \right)^{\frac{1}{2}}. \tag{4}$$

The operator corresponding to the Equation (1) is $\mathcal{L} = \nabla^2 + k$, and $\mathcal{L}u = (\nabla^2 + k)u(\boldsymbol{x})$ represents the action of the operator $\mathcal{L}$ on the solution function $u(\boldsymbol{x})$.

In numerical computations, when applying an operator to a function, we utilize discretization methods for PDEs as discussed in the Preliminaries 3.1. This process essentially projects the PDE from an infinite-dimensional Sobolev function space to a finite-dimensional linear space. It represents the differential operator $\mathcal{L}$ as a linear transformation $\boldsymbol{A}$, and functions are expressed in vector form: solutions u as vector $\boldsymbol{x}$, and the right-hand side functions f as vector $\boldsymbol{b}$. Thus, the PDE problem is transformed into a linear equation system:

$$\mathcal{L}u = f \rightarrow \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}. \tag{5}$$

In this context, the existing method used for dataset generation translates into solving large linear equation systems by finding $\boldsymbol{x}$ using $\boldsymbol{A}$ and $\boldsymbol{b}$. However, in our method, the operator action is converted into multiplying matrix $\boldsymbol{A}$ by vector $\boldsymbol{x}$ to obtain $\boldsymbol{b}$.

For the same problem, the computation required for a single matrix-vector multiplication is significantly less than that for solving the corresponding system of linear equations, and it does not introduce additional errors from the solving process. Therefore, our algorithm offers greater speed and higher precision.

5. Theoretical Analysis

5.1. Computational Complexity Analysis

5.1.1. Existing Method

Solving the corresponding linear systems constitutes the primary computational expense in numerical PDE solving (Hughes, 2012). Consequently, the computational cost of generating data points using the existing algorithm can be estimated by the cost of solving these linear systems. The GMRES, an iterative algorithm based on the Krylov subspace (Liesen & Strakos, 2013; Qin & Xu, 2023), is commonly used for large, sparse, and nonsymmetric linear systems. Iteratively constructing the Krylov space, this method projects the linear equation system of a large matrix onto a smaller subspace, effectively reducing computational complexity. The specific pseudocode is provided in Appendix A.

The most computationally intensive components are the matrix-vector multiplication and the orthogonalization process. Assuming that the matrix dimension is n and the current iteration number is denoted by $j = 1, 2, \cdots, m$ with a total of m iterations, where m represents the final dimension of the Krylov subspace.

The complexity of each iteration is primarily determined by $O(n^2)$ for the matrix-vector product and $O(j \times n)$ for the orthogonalization process. In the worst case scenario where m approaches n, the total complexity over m iterations can be approximated as $O(m \times n^2)$ for the matrix-vector products and $O(m^2 \times n)$ for the orthogonalization. The overall computational cost can be approximated by $O(m \times n^2 + m^2 \times n)$. Furthermore, in practical scenarios, the computational complexity is influenced by the matrix's sparsity and the algorithm's implementation specifics.

Assuming the dataset contains N data points, the computational cost of generating the dataset using the existing method can be approximated as $O(m \times n^2 \times N + m^2 \times n \times N)$.

5.1.2. DiffOAS Method

According to the introduction in Method 4, the DiffOAS method is divided into two steps: solution functions generation and operator action. In numerical computations, as explained in Preliminaries 3.1, we use discretization methods to represent PDE operators as matrices and implement operator actions by multiplying these discretized matrices with vectors corresponding to the solution functions.

In the process of generating the solution function, we utilize existing method to generate basis functions and construct the solution function space based on these functions. Since the computational cost of generating the solution function space from the basis functions is negligible, our focus will be solely on discussing the time complexity of generating the basis functions. Let l denote the number of basis functions, which is typically much smaller than the dataset size N (with l typically chosen to be between 10 and 50). The time complexity of this step can be expressed as $O(m \times n^2 \times l + m^2 \times n \times l)$.

The operator action step is a single matrix-vector multiplication operation. The time complexity for dense matrices is $O(n^2)$. In particular, the time complexity can be influenced by the sparsity of the matrix. If the dataset contains N data points, the computational cost of generating the dataset using our method is approximately $O(n^2 \times N)$.

Therefore, the overall computational complexity of our algorithm can be expressed as $O(n^2 \times N + m \times n^2 \times l + m^2 \times n \times l)$. By comparing the computational complexities of the existing method and our method, since l is a constant value and considerably smaller than N. m is mathematically of the same order as n (typically ranging between $n/20$ to $n/5$ in experiments). Consequently, our method theoretically provides a speedup of approximately $O(m) \approx O(n)$, which corresponds to an increase in speed by one order, where m and n are of the same order.

5.2. Error Analysis

In constructing datasets for physical problems, errors primarily arise from three sources: 1. Physical Modeling Error, which refers to the discrepancy between the PDE and the actual physical processes; 2. Discretization Error, which occurs due to the chosen numerical PDE methods and grid densities; 3. Data Generation Error, which is associated with the errors in generating solution functions that satisfy the grid constraints. In our analysis, we focus on the third type of error, assuming the accuracy of the PDE and fixed discretization methods.

Inaccuracies in solving linear systems are the main source of error in data generation using existing algorithms. For Krylov subspace algorithms, such as GMRES, to solve large sparse nonsymmetric linear systems, the magnitude of $h_{j+1,j}$ from the matrix $\boldsymbol{H}_j$ (generated during the Krylov subspace iteration) serves as a termination criterion for the iterations.

The GMRES algorithm is used to solve linear systems of the form $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ by minimizing the residual norm $\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|$. This is achieved through the construction of a Krylov subspace and the utilization of the Arnoldi process to generate an orthogonal basis for the subspace.

During the j-th iteration of the Arnoldi process, a Hessenberg matrix $\boldsymbol{H}_j$ and an orthogonal matrix $\boldsymbol{V}_{j+1}$ are generated, satisfying the equation $\boldsymbol{A}\boldsymbol{V}_j = \boldsymbol{V}_{j+1}\underline{\boldsymbol{H}}_j$. In this context, $\boldsymbol{V}_j$ contains the basis vectors of the Krylov subspace, and the augmented Hessenberg matrix $\underline{\boldsymbol{H}}_j$ includes an additional row $h_{j+1,j}$.

The residual of the j-th iteration's approximate solution $\boldsymbol{x}_j$, denoted as $\boldsymbol{r}_j = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_j$, can be expressed as $\boldsymbol{r}_j = \boldsymbol{V}_{j+1}(\beta \boldsymbol{e}_1 - \underline{\boldsymbol{H}}_j \tilde{\boldsymbol{y}})$. Here, $\beta = \|\boldsymbol{r}_0\|$ represents the norm of the initial residual, and $\boldsymbol{e}_1$ denotes the first unit vector in the standard basis of the corresponding space. The vector $\tilde{\boldsymbol{y}}$ an augmented version of $\boldsymbol{y}$, is obtained by appending an additional zero element to the solution vector $\boldsymbol{y}$, which is determined by the central minimization problem in GMRES.

The norm of the residual $\|\boldsymbol{r}_j\|$ can be expressed as $\|\boldsymbol{r}_j\| = \|\beta \boldsymbol{e}_1 - \underline{\boldsymbol{H}}_j \tilde{\boldsymbol{y}}\|$. By applying the triangle inequality, an estimation can be derived

$$\|\boldsymbol{r}_j\| \leq \|\beta \boldsymbol{e}_1 - \boldsymbol{H}_j \boldsymbol{y}\| + \|h_{j+1,j} y_j \boldsymbol{e}_{j+1}\|. \tag{6}$$

Given that $\|\beta \boldsymbol{e}_1 - \boldsymbol{H}_j \boldsymbol{y}\|$ is minimized through the iterative process, the major component of the residual arises from $\|h_{j+1,j} y_j \boldsymbol{e}_{j+1}\|$, which aids in the approximation

$$\|\boldsymbol{r}_j\| \leq |h_{j+1,j}| \|y_j\|. \tag{7}$$

The magnitude of $h_{j+1,j}$ in GMRES algorithm serves as an indicator for estimating the upper bound of the error in $\boldsymbol{r}_j$. A smaller value of $h_{j+1,j}$ typically indicates a reduced residual, suggesting a close approximation of $\boldsymbol{x}_j$ to the actual solution.

The computational cost of solving linear systems is directly influenced by the predefined error. We utilize the upper bound equation for error, Equation (7) to establish a requirement for the magnitude of $h_{j+1,j}$, continuing iterations when it is substantial and terminating them if it falls below a threshold. The precision requirement for the dataset varies depending on the inherent accuracy of the data-driven algorithm. For example, in algorithms like FNO, where the final error typically ranges from 10^{-2} to 10^{-4}, the relative error in the data should ideally be around 10^{-7} to 10^{-10} or lower to maintain the significance of training process.

In our method, the operator is actioned to the generated solution functions, essentially performing matrix-vector multiplication. The precision of this operation is governed by the machine epsilon of floating-point arithmetic in computers, typically, the error is around 10^{-16} to 10^{-17}, or even lower. Achieving this high level of precision is nearly impossible with existing method.

To demonstrate how larger data errors can impact model training, we conducted additional experiments detailed in Appendix C.2, where we trained models using datasets with varying error levels.

6. Experiment

In this chapter, we compare our proposed data generation method with existing data generation methods. Our code is available at `https://github.com/hs-dong/DiffOAS`.

6.1. Experimental Setup

Our analysis examines three main performance indicators, which are crucial for evaluating the effectiveness of data generation methods:

- Accuracy of the data.
- Time cost of generating data.

- Errors obtained from training on neural operator models.

In our experiments, we focus on testing two widely recognized and widely used neural operator models, which are the most prominent and common models in data-driven algorithms for PDE:

- FNO (Fourier Neural Operator) (Li et al., 2020)
- DeepONet (Deep Operator Network) (Lu et al., 2019)

We tested three different types of PDE problems that have important applications in science and engineering (detailed descriptions are listed in the Appendix B:

- Darcy Flow Problem (Li et al., 2020)
- Scalar Wave Equation in Electromagnetism (Zhang et al., 2022)
- Solute Diffusion in Porous Media (Mauri, 1991)

Baselines. The main time expense of existing data generation methods is solving a system of linear equations composed of large sparse nonsymmetric matrices (Hughes, 2012). We use an existing data generation method based on the **GMRES** algorithm as the solution and baseline for our study, utilizing scipy 1.11.4 (Virtanen et al., 2020). For detailed GMRES algorithmic information, please refer to Appendix A.

For constructing the FNO and DeepONet models, we employed 100 instances of test data generated using GMRES methods, The detailed settings are presented in Appendix B.1. The data generation process was performed on Intel(R) Xeon(R) Gold 6154 CPU @ 3.00GHz, while the model training took place on a GeForce RTX 3090 GPU with 24GB of memory.

6.2. Training Result

The main experimental results for all datasets and models are shown in Table 1. More details and hyperparameters can be found in Appendix B. Based on these results, we have the following observations.

Firstly, DiffOAS method consistently demonstrates remarkable acceleration compared to GMRES methods across datasets of all sizes, particularly in the case of Darcy Flow where the acceleration ratio can reach approximately 30 times. Furthermore, the time required for DiffOAS to generate different numbers of training instances remains relatively constant. The time required for our method to generate data can be divided into two parts: generating basis functions and applying operators. The time to generate basis functions depends on the number of basis functions and is independent of the number of training instances generated. The time to apply operators is directly proportional to the number of training instances generated. Therefore, the experimental results indicate that the generation of basis functions constitutes the main portion of the data generation time for our method. This implies that the DiffOAS method can generate a large amount of training data at a low cost, demonstrating the efficiency of our approach.

Secondly, in different PDE problems and with different neural operators, the dataset generated by DiffOAS method exhibits better performance compared to GMRES methods on DeepONet. By increasing the training set size, the error can reach the same order of magnitude on FNO. This indicates that our method, while accelerating data generation by 30 times, can achieve comparable performance to the data generated by existing method. Regarding the phenomenon of slightly higher errors in some problems compared to existing method, it could be attributed to the fact that the test set is obtained through numerical solutions of linear equations, which inherently have larger errors. In contrast, our method generates datasets with machine precision. Therefore, the larger errors in the predictions of models trained on our datasets on the test set could be attributed to errors in the test set itself and discrepancies between the test set and the exact solution.

To further illustrate the reasonableness of our method, we provide more detailed experiments in Appendix C.

6.3. Data Generation Time And Accuracy

The high generation speed and accuracy are the main advantages of our method. We tested the data generation time for generating 10,000 data points at different accuracies in seconds. The results are shown in Table 2. Further experimental details can be found in Appendix B.3.

The experimental results show that compared to the GMRES method, the DiffOAS method achieves a speedup of approximately 300 times in terms of the total time, while the time for operator actions can be accelerated by up to approximately $70,000$ times for a matrix size of $62,500$. Since the generation of basis functions in our method does not depend on the training data size, as we generate sufficiently large data, the speedup ratio will approach the acceleration ratio of the operator action part, as predicted by the theoretical analysis in Section 5.1. In other words, the DiffOAS method reduces the time for generating the dataset by one order, significantly enhancing the efficiency of data generation.

In the process of solving linear systems using the GMRES algorithm, as the accuracy requirement increases, the solution time significantly increases. When the accuracy of

Table 1. Compare the data generation time and training results on different models between our DiffOAS method and GMRES methods. The first column lists the method used to generate the dataset and the number of training instances. The first row represents the corresponding PDE problem and the corresponding length of the matrix sides. Bolding indicates that our algorithm outperforms existing method.

DATASET	DARCY FLOW 10000			WAVE 22500			DIFFUSION 62500		
	TIME(s)	FNO	DEEPONET	TIME(s)	FNO	DEEPONET	TIME(s)	FNO	DEEPONET
GMRES 1000	2.99E2	4.56E-3	6.82E-2	3.27E3	4.05E-4	6.12E-2	1.99E4	4.29E-3	7.70E-2
DIFFOAS 1000	8.97E0	2.15E-2	**6.82E-2**	1.60E2	1.71E-3	**4.57E-2**	5.99E2	1.76E-2	**6.40E-2**
DIFFOAS 5000	9.20E0	6.86E-3	**5.58E-2**	1.60E2	7.25E-4	**4.74E-2**	6.01E2	5.66E-3	**5.80E-2**
DIFFOAS 10000	9.49E0	**4.26E-3**	**5.62E-2**	1.60E2	6.52E-4	**4.72E-2**	6.03E2	4.79E-3	**6.04E-2**

Table 2. Compare the data generation time of our DiffOAS algorithm and GMRES with various accuracies, where the error of the DiffOAS algorithm is the machine precision $1\mathrm{E}-16$. 'TIME1' and 'TIME2' represent the total time and operator action time for data generation, respectively. The remaining parameters in the second row indicate the errors of the GMRES algorithm. 'SIZE' indicates the matrix size of the PDE problem.

SIZE	DIFFOAS		GMRES			
	TIME1	TIME2	1E-1	1E-3	1E-5	1E-7
2500	9.732E-1	2.687E-1	4.530E1	1.394E2	2.349E2	3.265E2
10000	9.582E0	5.931E-1	5.750E2	1.790E3	2.996E3	4.350E3
16900	8.192E1	8.066E-1	5.933E3	1.771E4	2.704E4	4.066E4
22500	1.236E2	1.004E0	7.994E3	2.305E4	4.086E4	5.251E4
32400	1.983E2	1.462E0	1.466E4	4.216E4	6.561E4	9.384E4
40000	3.051E2	1.943E0	2.371E4	5.739E4	1.010E5	1.285E5
62500	5.971E2	3.815E0	4.052E4	1.186E5	1.977E5	2.722E5

the GMRES algorithm is improved from $1\mathrm{E}-5$ to $1\mathrm{E}{-}7$, the time increases by approximately 50%. In contrast, our DiffOAS method achieves a precision of $1\mathrm{E}{-}16$. This indicates that improving the accuracy of the GMRES algorithm comes with expensive computational costs, while our algorithm guarantees data accuracy at machine precision through operator actions. Therefore, the data generated by the DiffOAS algorithm has much higher precision compared to the data generated by existing method.

We analyzed the relationship between acceleration ratio and matrix size in Table 3. By performing linear regression in the least squares sense on the largest five matrix sizes, we found that Pearson's r reached 0.927 on the GMRES algorithm with an accuracy of $1\mathrm{E}-7$. This indicates that the partial acceleration ratio due to the operator action in our algorithm is of the same order as N, implying a one order acceleration in our algorithm, consistent with the time Computational Complexity Analysis 5.1.

Our method tackles a common scenario: solving systems of linear equations involving large, sparse, non-symmetric matrices. The GMRES algorithm is the mainstream approach

Table 3. For each precision of the GMRES algorithm, a linear regression is performed on the ratio of the matrix size and the acceleration achieved by the DiffOAS method operator. The slope represents the slope of the fitted line, and Pearson's r is the Pearson correlation coefficient, where a value closer to 1 indicates a more reliable fit.

ACCURACY	SLOPE	PEARSON R
1E-1	0.079	0.715
1E-3	0.206	0.890
1E-5	0.380	0.867
1E-7	0.473	0.927

Table 4. Experimental results comparing datasets generated using different basis functions.

BASIS	FNO	DEEPONET
DIFFOAS	4.260%	5.629%
GRF	31.44%	88.08%
FOURIER	93.56%	96.98%
CHEBYSHEV	63.11%	68.23%

for addressing such problems. However, for certain types of equations, specifically when the system is symmetric positive definite, numerical algorithms like MINRES and Conjugate Gradient should also be considered for baseline comparisons. Consequently, we have included comparative experiments with these baselines in Appendix C.1.

6.4. Ablation Result

Finally, we conducted an ablation study to demonstrate the significant impact of the solution function generation method in our DiffOAS method on the quality of the dataset. In Table 4, we tested datasets generated using GRF, Fourier basis, and truncated Chebyshev functions as the basis functions. For specific details about the design of the basis functions, please refer to Appendix B.4.

Experimental results show that the training results of the data sets corresponding to the solution functions generated by other methods are quite poor on the FNO and DeepONet models. Even the best training results on FNO and DeepONet have errors of more than 30% and 60%, respectively. This shows that these data sets have little training value. Therefore, the solution function introduced in the DiffOAS method that conforms to the real physical distribution is necessary as a basis function and is crucial to ensuring the generation of high-quality datasets that can make accurate predictions.

7. Limitation and Conclusions

Limitation While our approach has shown promise in speeding up PDE dataset generation, there are areas for further exploration: 1. Our accelerated algorithm is designed for general PDE problems with asymmetric coefficient matrices, but it doesn't specifically target particular types of PDE problems. 2. While generating basis functions, we can enhance dataset quality by choosing specific functions from the generated set using optimization techniques.

Conclusions In this article, we introduce the DiffOAS algorithm. To our knowledge, this is the first PDE dataset generation method that does not require solving linear equations. Specifically, the algorithm consists of two parts: solution function generation and operator action. By generating solution functions using basis functions, we ensure that the generated data conforms to physical distributions. The operator action significantly accelerates the data generation process while maintaining mechanical precision in our generated data. The DiffOAS method ensures both speed and accuracy, alleviating a significant obstacle to the development of neural networks.

Acknowledgements

This work was supported in part by National Key R&D Program of China under contract 2022ZD0119801, National Nature Science Foundations of China grants U23A20388, 62021001, U19B2026, and U19B2044. This work was supported in part by Huawei as well. We would like to thank all the anonymous reviewers for their insightful comments.

Impact Statement

In this paper, we propose a new PDE dataset generation method that significantly improves the speed of data generation. This work has great potential in various practical and important scenarios, such as physics, chemistry, biology, and various scientific problems.

References

Brandstetter, J., Berg, R. v. d., Welling, M., and Gupta, J. K. Clifford neural layers for pde modeling. *arXiv preprint arXiv:2209.04934*, 2022.

Cheng, L. and Xu, K. Solving time-dependent pdes with the ultraspherical spectral method, 2023.

Evans, L. C. *Partial differential equations*, volume 19. American Mathematical Society, 2022.

Golub, G. H. and Van Loan, C. F. *Matrix computations*. JHU press, 2013.

Götz, M. and Anzt, H. Machine learning-aided numerical linear algebra: Convolutional neural networks for the efficient preconditioner generation. In *2018 IEEE/ACM 9th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (scalA)*, pp. 49–56. IEEE, 2018.

Greenfeld, D., Galun, M., Basri, R., Yavneh, I., and Kimmel, R. Learning to optimize multigrid pde solvers. In *International Conference on Machine Learning*, pp. 2415–2423. PMLR, 2019.

Hao, Z., Liu, S., Zhang, Y., Ying, C., Feng, Y., Su, H., and Zhu, J. Physics-informed machine learning: A survey on problems, methods and applications. *arXiv preprint arXiv:2211.08064*, 2022.

Hestenes, M. R., Stiefel, E., et al. Methods of conjugate gradients for solving linear systems. *Journal of research of the National Bureau of Standards*, 49(6):409–436, 1952.

Hsieh, J.-T., Zhao, S., Eismann, S., Mirabella, L., and Ermon, S. Learning neural pde solvers with convergence guarantees. *arXiv preprint arXiv:1906.01200*, 2019.

Hughes, T. J. *The finite element method: linear static and dynamic finite element analysis*. Courier Corporation, 2012.

Johnson, C. *Numerical solution of partial differential equations by the finite element method*. Courier Corporation, 2012.

Kaneda, A., Akar, O., Chen, J., Kala, V. A. T., Hyde, D., and Teran, J. A deep conjugate direction method for iteratively solving linear systems. In *International Conference on Machine Learning*, pp. 15720–15736. PMLR, 2023.

Kovachki, N., Li, Z., Liu, B., Azizzadenesheli, K., Bhattacharya, K., Stuart, A., and Anandkumar, A. Neural operator: Learning maps between function spaces. *arXiv preprint arXiv:2108.08481*, 2021.

LeVeque, R. J. *Finite volume methods for hyperbolic problems*, volume 31. Cambridge university press, 2002.

Li, Y., Chen, P. Y., Matusik, W., et al. Learning preconditioner for conjugate gradient pde solvers. *arXiv preprint arXiv:2305.16432*, 2023.

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.

Liesen, J. and Strakos, Z. *Krylov subspace methods: principles and analysis*. Numerical Mathematics and Scie, 2013.

Liu, N., Yu, Y., You, H., and Tatikola, N. Ino: Invariant neural operators for learning complex physical systems with momentum conservation. In *International Conference on Artificial Intelligence and Statistics*, pp. 6822–6838. PMLR, 2023.

Lu, L., Jin, P., and Karniadakis, G. E. Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint arXiv:1910.03193*, 2019.

Lu, L., Meng, X., Cai, S., Mao, Z., Goswami, S., Zhang, Z., and Karniadakis, G. E. A comprehensive and fair comparison of two neural operators (with practical extensions) based on fair data. *Computer Methods in Applied Mechanics and Engineering*, 393:114778, 2022.

Luz, I., Galun, M., Maron, H., Basri, R., and Yavneh, I. Learning algebraic multigrid using graph neural networks. In *International Conference on Machine Learning*, pp. 6489–6499. PMLR, 2020.

Mauri, R. Dispersion, convection, and reaction in porous media. *Physics of Fluids A: Fluid Dynamics*, 3(5):743–756, 1991.

Morton, K. W. and Mayers, D. F. *Numerical solution of partial differential equations: an introduction*. Cambridge university press, 2005.

Paige, C. C. and Saunders, M. A. Solution of sparse indefinite systems of linear equations. *SIAM journal on numerical analysis*, 12(4):617–629, 1975.

Perkins, T. and Johnston, O. A review of diffusion and dispersion in porous media. *Society of Petroleum Engineers Journal*, 3(01):70–84, 1963.

Qin, O. and Xu, K. Solving nonlinear odes with the ultraspherical spectral method. *arXiv preprint arXiv:2306.17688*, 2023.

Rahman, M. A., Ross, Z. E., and Azizzadenesheli, K. U-no: U-shaped neural operators. *arXiv preprint arXiv:2204.11127*, 2022.

Saad, Y. and Schultz, M. H. Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on scientific and statistical computing*, 7(3):856–869, 1986.

Stanaityte, R. *ILU and Machine Learning Based Preconditioning for the Discretized Incompressible Navier-Stokes Equations*. PhD thesis, University of Houston, 2020.

Strikwerda, J. C. *Finite difference schemes and partial differential equations*. SIAM, 2004.

Taghibakhshi, A., MacLachlan, S., Olson, L., and West, M. Optimization-based algebraic multigrid coarsening using reinforcement learning. *Advances in neural information processing systems*, 34:12129–12140, 2021.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

Wang, H., Hao, Z., Wang, J., Geng, Z., Wang, Z., Li, B., and Wu, F. Accelerating data generation for neural operators via krylov subspace recycling. *arXiv preprint arXiv:2401.09516*, 2024.

Yang, C., Yang, X., and Xiao, X. Data-driven projection method in fluid simulation. *Computer Animation and Virtual Worlds*, 27(3-4):415–424, 2016.

Zachmanoglou, E. C. and Thoe, D. W. *Introduction to partial differential equations with applications*. Courier Corporation, 1986.

Zhang, E., Kahana, A., Turkel, E., Ranade, R., Pathak, J., and Karniadakis, G. E. A hybrid iterative numerical transferable solver (hints) for pdes based on deep operator network and relaxation methods. *arXiv preprint arXiv:2208.13273*, 2022.

Zhang, X., Wang, L., Helwig, J., Luo, Y., Fu, C., Xie, Y., Liu, M., Lin, Y., Xu, Z., Yan, K., Adams, K., Weiler, M., Li, X., Fu, T., Wang, Y., Yu, H., Xie, Y., Fu, X., Strasser, A., Xu, S., Liu, Y., Du, Y., Saxton, A., Ling, H., Lawrence, H., Stärk, H., Gui, S., Edwards, C., Gao, N., Ladera, A., Wu, T., Hofgard, E. F., Tehrani, A. M., Wang, R., Daigavane, A., Bohde, M., Kurtin, J., Huang, Q., Phung, T., Xu, M., Joshi, C. K., Mathis, S. V., Azizzadenesheli,

K., Fang, A., Aspuru-Guzik, A., Bekkers, E., Bronstein, M., Zitnik, M., Anandkumar, A., Ermon, S., Liò, P., Yu, R., Günnemann, S., Leskovec, J., Ji, H., Sun, J., Barzilay, R., Jaakkola, T., Coley, C. W., Qian, X., Qian, X., Smidt, T., and Ji, S. Artificial intelligence for science in quantum, atomistic, and continuum systems. *arXiv preprint arXiv:2307.08423*, 2023.

A. Specific pseudocode of GMRES

The following computational procedure is adapted from (Golub & Van Loan, 2013)

Algorithm 1 Generalized Minimal Residual Method (GMRES)

Input: $\boldsymbol{A} \in \mathbb{R}^{n\times n}$, $\boldsymbol{b} \in \mathbb{R}^n$,$\boldsymbol{x_0}$ is the initial vector, tolerance $\epsilon > 0$, maximum iterations m
Output: Approximate solution $\boldsymbol{x}$

$\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$
$\beta = \|\boldsymbol{r}_0\|_2$
$\boldsymbol{v}_1 = \boldsymbol{r}_0/\beta$
for $j = 1$ to m **do**
  $\boldsymbol{w}_j = \boldsymbol{A}\boldsymbol{v}_j$
  for $i = 1$ to j **do**
    $h_{i,j} = (\boldsymbol{w}_j, \boldsymbol{v}_i)$
    $\boldsymbol{w}_j = \boldsymbol{w}_j - h_{i,j}\boldsymbol{v}_i$
  end for
  $h_{j+1,j} = \|\boldsymbol{w}_j\|_2$
  if $h_{j+1,j} = 0$ **then**
    break
  end if
  $\boldsymbol{v}_{j+1} = \boldsymbol{w}_j/h_{j+1,j}$
  Form or update the Hessenberg matrix $\boldsymbol{H}_j$
  Solve the least squares problem: $\min_{\boldsymbol{y}\in\mathbb{R}^j} \|\beta\boldsymbol{e}_1 - \boldsymbol{H}_j\boldsymbol{y}\|_2$
  Update the solution: $\boldsymbol{x}_j = \boldsymbol{x}_0 + \boldsymbol{V}_j\boldsymbol{y}$
  if $\|\boldsymbol{r}_j\|_2 < \epsilon$ **then**
    break
  end if
end for
$\boldsymbol{x} = \boldsymbol{x}_j$

B. Specific Experimental Details

B.1. Model set

FNO: We employ 4 FNO layers with learning rate 0.001, batch size 20, epochs 500, modes 12, and width 32.

DeepONet: We utilize branch layers: [50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50] and trunk layers: [50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50], with the activation function set to tanh. The learning rate is 0.001, batch size is 20, and the training process is performed for 500 epochs.

B.2. Data

B.2.1. Darcy Flow

In this research, we delve into two-dimensional Darcy flows, which are governed by the equation (Li et al., 2020; Rahman et al., 2022; Kovachki et al., 2021; Lu et al., 2022):

$$-\nabla \cdot (K(x, y)\nabla h(x, y)) = f(x, y), \tag{8}$$

where K represents the permeability of the medium, h denotes the hydraulic pressure, and f is a source term that varies, being either a constant or a function dependent on spatial variables.

For our experimental setup, the permeability field $K(x, y)$ and the source term $f(x, y)$ are generated by the Gaussian Random Field (GRF) methodology, with a time constant $\tau = 7$ and a decay exponent $\alpha = 2.5$.

In DiffOAS method, In the DiffOAS method, we use 30 solution functions obtained by solving as basis functions.

B.2.2. Scalar Wave Equation in Electromagnetism

In this research, we delve into a two-dimensional Helmholtz equation in electromagnetism, expressed as (Zhang et al., 2022):

$$\Delta\Phi(x,y) + k^2(x,y)\Phi(x,y) = S(x,y), \tag{9}$$

where $\Phi(x,y)$ is the electromagnetic scalar potential and $k(x,y)$ the spatially varying wavenumber. $S(x,y)$ is the source term representing electromagnetic wave origins.

This equation is fundamental in modeling electromagnetic wave propagation and interaction in various media. For our experimental setup, $k^2(x,y)$ and the source term $S(x,y)$ are generated by the Gaussian Random Field (GRF) methodology, specifically as $GRF(\tau = 3, \alpha = 2)/10$.

In DiffOAS method, In the DiffOAS method, we use 50 solution functions obtained by solving as basis functions.

B.2.3. Solute Diffusion in Porous Media

We investigate the process of solute diffusion in a two-dimensional porous medium, described by the following equation (Perkins & Johnston, 1963; Mauri, 1991):

$$\nabla \cdot (k(x,y)\nabla C(x,y)) + q(x,y)C(x,y) = f(x,y), \tag{10}$$

where $C(x,y)$ symbolizes the solute concentration, $k(x,y)$ signifies the diffusion coefficient varying spatially, and $q(x,y)C(x,y)$ represents the influence of internal or external sources/sinks. The function $f(x,y)$ acts as an additional source or sink, pinpointing regions of solute addition or removal.

This equation is key for modeling solute movement in heterogeneous porous media. For our experimental setup, the diffusion coefficient $k(x,y)$ is generated by $10 * GRF(\tau = 3, \alpha = 2)$, $f(x,y)$ is generated by $GRF(\tau = 3, \alpha = 2)$ and the influence of internal or external sources/sinks $q(x,y)$ is generated by uniform distribution $U[0,1]$.

In DiffOAS method, In the DiffOAS method, we use 50 solution functions obtained by solving as basis functions.

B.2.4. Noise

We multiply the GRF function by a matrix that decays towards the edges, and then multiply it by a parameter related to the norm of the solution function to generate noise.

B.3. Time test

In this experiment, we compare the time required to generate 10,000 data points for Darcy flow problems. The parameters used for dataset generation are consistent with Appendix B.2. The basis functions for the DiffOAS method are generated using the GMRES algorithm with an accuracy of $1\mathrm{E}-5$, resulting in a total of 30 basis functions.

B.4. Ablation

In the ablation experiment, we consider the training results of 10,000 training instances generated using several different basis functions on the Darcy flow problem. The basis functions are:

- **GRF**: The function generated by the $GRF(\tau = 7, \alpha = 2.5)$ using 30 solution functions as basis functions.
- **Fourier**: The function generated by periodic sine wave function on the two-dimensional plane using 100 solution functions as basis functions.
- **Chebyshev**: The function generated by truncated Chebyshev function using 100 solution functions as basis functions.

C. Additional Experiments

C.1. Time comparison Experiments

We conducted comparative experiments involving the SKR algorithm, MINRES, and Conjugate Gradient algorithms (Wang et al., 2024; Paige & Saunders, 1975; Hestenes et al., 1952). MINRES and Conjugate Gradient algorithms are often applied

to specific matrix types; for instance, the Conjugate Gradient method is tailored for symmetric positive-definite matrices. To maintain consistency in presenting results, we focused on scenarios involving symmetric positive-definite matrices:

Table 5. Considering the Darcy flow problem, appropriate boundary conditions are selected to ensure that the corresponding linear equation system is symmetric positive-definite. Each corresponding matrix has a size of 62500, and the table records the time taken by different algorithms to generate 1000 training data sets. The DiffOAS method's time denotes the duration for generating training data after obtaining basis functions.

METHOD	TIME(S)
DIFFOAS	3.893E-1
SKR	1.283E3
GMRES	1.974E4
MINRES	1.149E2
CONJUGATE GRADIENT	1.426E3

The experimental results show that the DiffOAS algorithm significantly outperforms other numerical algorithms in terms of computational time required for data generation. MINRES and Conjugate Gradient, optimized for specific matrix types, are faster than GMRES. Due to the large condition number of the matrices corresponding to the equations, MINRES demonstrates faster speeds than Conjugate Gradient. The SKR algorithm surpasses GMRES by optimizing the solution for multiple related linear equation systems. However, since it's not optimized for the symmetry of matrices, its speed still lags behind MINRES. Even though our method is not tailored for specific matrix types, it still generates data faster than numerically optimized algorithms (MINRES and Conjugate Gradient) designed for specific matrix types.

It is worth noting that our method can also be combined with the SKR algorithm or other numerical algorithms. Our method mainly consists of two parts: basis function generation and training data generation after obtaining the basis functions. Through the following experiments, it can be observed that the time cost of generating data with the DiffOAS algorithm after obtaining basis functions is much lower than that of all existing methods, including the SKR algorithm. Therefore, the basis function generation becomes the primary time cost of DiffOAS. The SKR algorithm significantly improves the efficiency of generating basis functions. Due to the SKR algorithm's ability to significantly enhance the efficiency of basis function generation, its integration also leads to notable performance improvement in our method.

C.2. Error Experiments

We evaluated models trained with datasets of varying accuracies generated by the GMRES algorithm to demonstrate the impact of low-precision data on model training.

The experimental results indicate that when there are significant data accuracy errors, the errors noticeably impact the performance of the model. This suggests that obtaining high-accuracy data has a significant impact on the quality of the dataset.

Table 6. For the Darcy flow problem with a matrix dimension of 10,000, FNO was used as the testing model. The training outcomes of models trained on datasets generated by the GMRES algorithm, featuring truncation errors of 10^{-1}, 10^{-2}, and 10^{-5}, were evaluated.

TRAIN NUMBER	TOL=1E-1	TOL=1E-2	TOL=1E-5
100	1.25E-1	4.54E-2	4.41E-2
500	1.20E-1	1.14E-2	6.87E-3
1000	1.20E-1	1.04E-2	4.56E-3
5000	1.20E-1	9.60E-3	1.69E-3
10000	1.20E-1	9.43E-3	1.15E-3
20000	1.20E-1	9.85E-3	1.01E-3

C.3. Saturated Data

To demonstrate that our method achieves better training performance and faster generation speed compared to existing methods when the data volume reaches saturation, we use the DeepONet model as an example and consider the Darcy flow problem, where each corresponding matrix has a size of 10000. We use DeepONet as the testing model, with model parameters consistent with those in Appendix B.1.

Table 7. The first column lists the method used to generate the dataset and the number of training instances.

	TIME	ERROR
GMRES 500	1.50E2	7.85E-2
GMRES 1000	2.99E2	6.82E-2
GMRES 5000	1.47E3	7.08E-2
GMRES 10000	3.01E3	6.94E-2
DIFFOAS 500	8.94E0	8.62E-2
DIFFOAS 1000	8.97E0	6.82E-2
DIFFOAS 5000	9.20E0	5.58E-2
DIFFOAS 10000	9.49E0	5.62E-2

The experimental results indicate that for this problem, the upper bound 'h' for the dataset size in our DiffOAS method does not exceed 10000, while for the existing methods using GMRES, the upper bound 'h' for the dataset size is around 1000. At this point, we observe that the training performance of the datasets generated by our method surpasses that of the existing methods, and the time required to generate the datasets using our method is significantly lower than that of the existing methods.

C.4. Additional Experiments on FNO

To demonstrate that our method does not compromise the performance of the FNO model, we have supplemented the results of the Solute Diffusion in Porous Media (Diffusion) problem on the FNO model, along with the corresponding data generation time:

Table 8. The first column lists the method used to generate the dataset and the number of training instances.

DIFFUSION 62500	TIME	FNO
GMRES 1000	1.99E4	4.29E-3
DIFFOAS 1000	5.99E2	1.76E-2
DIFFOAS 5000	6.01E2	5.66E-3
DIFFOAS 10000	6.03E2	4.79E-3
DIFFOAS 15000	**6.05E2**	**3.47E-3**
DIFFOAS 20000	**6.07E2**	**3.39E-3**

The experimental results indicate that increasing the dataset size generated by our method leads to better performance compared to datasets generated by existing methods. Despite the larger volume of data we generated, the time required for data generation using our method remains significantly lower than that of existing methods.

C.5. Training Time

We have provided additional information on the training time, dataset generation speed, and training results for the Solute Diffusion in Porous Media (Diffusion) on the FNO model to demonstrate that our method accelerates the entire process of training neural operators.

The experimental results show that the total time for data generation and model training on the DiffOAS 20000 dataset is still lower than the data generation time for GMRES 1000. Additionally, we observed that by the 300th epoch of training on DiffOAS 20000, the model's error reaches 4.00E-3, which outperforms the results obtained by training GMRES for 500

Table 9. Each model was trained for 500 epochs. Additionally, we have provided the training result on the DiffOAS 20000 dataset for 300 epochs in the last row.

DIFFUSION 62500	GENERATE TIME (S)	TRAINING TIME(S)	FNO
GMRES 1000	1.99E4	7.42E2	4.29E-3
DIFFOAS 15000	6.05E2	1.08E4	3.47E-3
DIFFOAS 20000	6.07E2	1.45E4	3.39E-3
DIFFOAS 20000(300 EPOCH)	6.07E2	8.73E3	4.00E-3

epochs. At this point, the training time for the model is only 8.73E3 seconds. This indicates that our method effectively improves the efficiency of the entire model training process.

C.6. Relative Residual

To illustrate that the error in the generated data has a minor impact on the training results of the model, we compared the error in our generated data with the error obtained from training the neural network.

Error in the neural network: From the experimental results in Section 6.2, it is observed that the error obtained by the trained neural network does not fall below10^{-4}. The error is measured using the mean squared error (MSE) loss, defined as $Loss = \frac{||x_{NO}-x_{test}||}{||x_{test}||}$, where x_{NO} is the result obtained by the neural network and x_{test} is the result obtained by the GMRES algorithm.

Error in GMRES: In the experiments, the error used in the data generated by the GMRES algorithm is 10^{-5}, measured using the relative residual, defined as $error = \frac{||Ax_{test}-b||}{||x_{test}||}$, where x_{test} is the result obtained by the GMRES algorithm.

Considering the difficulty in obtaining analytical solutions, when comparing the errors of the neural network and the GMRES algorithm, we consider their relative residuals. We conducted experiments to retest the relative residuals of the results obtained by the neural network and the corresponding matrix norms:

Table 10. The truncation error of GMRES and the relative residuals of the model training results for the Darcy flow problem, Scalar Wave Equation in Electromagnetism, and Solute Diffusion in Porous Media.

	RELATIVE RESIDUAL
GMRES	<1.00E-5
DARCY FLOW 10000	5.08E0
WAVE 22500	7.31E0
DIFFUSION 62500	5.75E0

The experiments demonstrate that the relative residual of the neural network in this paper is significantly higher than the relative residual obtained by the GMRES algorithm. Therefore, the error generated by GMRES algorithm in our experiment has a minor impact on model training.

C.7. Basis Function

To show that our method can cover distribution biases by generating a large volume of data, we compared the minimum distance between each R.H.S. function in the test set and all R.H.S. functions in the training set for the Darcy flow problem. Denote the test dataset matrix as A and the training dataset matrix as B, with the distance calculated as $||A - B||_1/||A||_1$. The table records the average of all minimum distances for R.H.S. functions in the test set. We compare the datasets from Section 6.2, and several datasets with different basis functions from the ablation experiments in Section 6.4. "GMRES 1000" represents the dataset from existing methods, adhering to the real R.H.S. distribution.

The experimental results indicate that compared to datasets generated by other basis functions, our method produces training datasets with R.H.S. functions that closely resemble those in the test set. Moreover, increasing the volume of data reduces this distance, indicating that the basis functions employed by our method align well with the physical meaning. The

Table 11. The first column represents the datasets, and the second column shows the average of all minimum distances corresponding to the R.H.S. functions

DATASET	DARCY FLOW
GMRES 1000	1.12E-1
DIFFOAS 1000	3.20E-1
DIFFOAS 5000	2.95E-1
DIFFOAS 10000	2.90E-1
GRF	2.91E2
FOURIER	1.95E3
CHEBYSHEV	2.02E2

comparable results achieved by our method and datasets obtained by existing methods suggest that our method can cover the R.H.S. distribution of the test set through the generation of large datasets.